Computer Science To Go (CS2Go): Developing a course to introduce and teach Computer Science and Computational Thinking to secondary school students

JAMES LOCKWOOD

A thesis submitted for the degree of

Master of Computer Science

Department of Computer Science

Maynooth University

May 2019

Supervisor: Dr. Aidan Mooney

Head of Department: Dr. Joseph Timoney

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Computer Science To Go (CS2Go) is a course designed to teach Transition Year Students about Computer Science and Computational Thinking. This project has been conducted over two years and this thesis charts the development of the course from the initial research stage, through the lesson creation sections to the testing and evaluation of the course material. Over 80 hours of engaging, informative and challenging material has been developed for use in the classroom.

Alongside the lesson plans, assessment and monitoring tools have been created, including a novel tool to assess students Computational Thinking skills. The content was tested in two major studies after an initial pilot study. This initial pilot study proved useful in constructing the full CS2Go course. Overall the course has been well received with teachers and students engaging well with the content. A web portal has also been created to allow for easy dissemination of all the CS2Go material. The further development of this web portal will turn CS2Go into a one-stop shop for teachers and educators hoping to find CS teaching material.

PUBLICATIONS

The publications that arose as a result of the research contained in this thesis are as follows:

Peer-Reviewed:

J. Lockwood and A. Mooney. "Computational Thinking in Secondary Education: Where does it fit? A systematic literary review." In: *International Journal of Computer Science Education in Schools* 2018 (2018), pp. 41–60

J. Lockwood and A. Mooney. "Developing a Computational Thinking Test using Bebras problems". In: *TACKLE: the 1st Systems of Assessments for Computational Thinking Learning workshop at EC-TEL 2018*. Vol. 2018. 2018

J. Lockwood and A. Mooney. "A Pilot Study Investigating The Introduction Of A Computer-Science Course at Second Level Focusing on Computational Thinking". In: *Irish Journal of Education* (2017), pp. 108–127

Workshops:

J. Lockwood, A. Mooney, and G. O'Mahony. "An Introduction to delivering Computational Thinking". In: *Computers in Education Society of Ireland Conference 2017*

J. Lockwood and A. Mooney. "Teaching computer science through unplugged puzzles and games". In: *Computers in Education Society of Ireland Conference 2018*

J. Lockwood and A. Mooney. "Computer Science 2 Go: Teaching Computer Science Through Unplugged Puzzles And Games". In: *8th Irish Conference on Game-Based Learning IGBL 2018 Cork City Ireland 28th and 29th June 2018*

ACKNOWLEDGMENTS

Firstly, I would like to thank my friends and family for their support and encouragement during these last two years. Thank you especially to my parents and my sister who spent hours proof-reading this final manuscript and also willingly took every test and challenge I threw at them.

Thank you to Dr. Aidan Mooney for his support and help throughout the two years as my academic supervisor. This project wouldn't have happened without you encouraging me to go down this path of research and study in the first place. Thank you for all the feedback and input you've give over the two years.

Thank you to my fellow postgraduate students in the Computer Science department who continually filled out forms and gave feedback whenever I asked. Special thanks to Keith Nolan, Emlyn Hegarty, Mark Noone and Keith Quille for all of your help and advice during the two years.

I couldn't have done this project without the support and interest of many teachers and other educational researchers who gave input to the project and more importantly tested CS2Go. Thank you to my own secondary school Drogheda Grammar School for their support of me and for allowing me to use their students as guinea-pigs. Thanks to all the students who gave feedback and took tests for the project, both in secondary schools and in Maynooth University.

Finally, thank you to Orlagh for putting up with me during the long nights of typing and for taking so much time to proof read and give feedback on this project as a whole. I hope more pictures of me in a cap and gown will make up for it!

DECLARATION

I confirm that this is my own work and the use of all material from other sources has been properly cited and fully acknowledged.

James Lockwood
September 2018

Part I

INTRODUCTION AND BACKGROUND MATERIAL

1 INTRODUCTION & BACKGROUND

1.1 INTRODUCTION

In this chapter, we will explain the rationale behind this project and the development of CS2Go. This will include an overview of the Irish Education System, the current state of Computer Science education in Ireland and around the world as well as discussing Computational Thinking. We will also describe previous work carried out in Maynooth University and how this project came about through that work, and how this work expands on it.

1.2 THESIS OVERVIEW

CS2Go is a fully contained Computational Thinking curriculum designed and built for Transition Year students and teachers. With the introduction of Computer Science in the Leaving Certificate curriculum and the short course in Coding now available at Junior cycle level it was felt that there was a market for another Computer Science offering at Transition Year level. CS2Go attempts to fill this gap with a focus on Computational Thinking.

The following sections provide a brief overview of what each chapter of this thesis will cover and ranges from the initial background work through to the creation and delivery of CS2Go, to the analysis of results, and finally presenting possible future enhancements.

1.2.1 CHAPTER 1 – INTRODUCTION & BACKGROUND

This chapter will discuss the motivations for and background to this project. We will examine Computer Science education in Ireland and around the world as well as looking at what Computational Thinking is and why it is important. A brief overview of the current work being undertaken within the Department of Computer Science at Maynooth University will be presented. The context of the project, its aims and how it furthers the work in the CS education space will also be discussed.

1.2.2 CHAPTER 2 – LITERATURE REVIEW

This chapter will present a literature review relevant to this thesis which aims to present an overview of what work has been carried out in the domain, as well as potential gaps and opportunities that still exist. This was the first stage undertaken in this project and the problems, opportunities and ideas that are presented in this chapter underpin CS2Go.

1.2.3 CHAPTER 3 – COURSE DEVELOPMENT

This chapter will focus on the development of the CS2Go course. The course was developed over an extended period of time, with most of it being complied in the first year of the study (academic year 2016-2017), before being enhanced with further topics and lessons during the second year (academic year 2017-2018). This chapter will describe the content which was developed including overviews of the individual lessons, example timetables and information on the rationale behind certain technology choices.

1.2.4 CHAPTER 4 – ASSESSMENT

This chapter will focus on one of the major components of developing the course. This component focused on how to analyse and assess its impact on students' Computation Thinking skills, their views of Computer Science as well as to see if students learned something from the lessons and enjoyed them. This chapter will describe the assessment and feedback tools used and designed for the course.

1.2.5 CHAPTER 5 – PILOT STUDY

This chapter presents an overview of an initial pilot study and related results from the study conducted in one secondary school in the spring term of 2017. The pilot study was designed to test a small sample of the material as well as the viability of the assessment metrics. Feedback and results from this study were used to inform any improvements and modifications that were needed before the main study during the 2017-18 school year.

1.2.6 chapter 6 – main school study

This chapter will focus on the main school study using CS2Go. During the 2017-18 school year a number of teachers in several schools taught a selection of the lessons to their classes. This chapter will give an overview of the feedback received from this study as well as results from the assessment metrics.

1.2.7 chapter 7 – first year study

This chapter will focus on an additional study carried out in Maynooth University with a first year undergraduate Computer Science class. The assessment tools described in Chapter 4 were used during the year and this chapter presents information on the CS course, the students and results from the different metrics.

1.2.8 chapter 8 – discussion and future work

This chapter will discuss the major findings and contributions from this project as well as lessons learned. It will also look at the future of CS2Go. The hope is that CS2Go will not end after the completion of this study. There is huge potential for it to be expanded and improved upon, as well as be examined in more detail. In this chapter we will discuss the various steps that can be taken to achieve these goals.

1.3 computer science education in ireland

1.3.1 the irish education system

The Irish education system is broken into three level, namely, primary, secondary and tertiary education. Education is compulsory from ages six to sixteen or until students have completed the first three years of secondary education. Primary school consists of eight years of study. The first two years of primary school are referred to as Junior and Senior infants and followed then by 1st to 6th class.

Although not compulsory until age six, the department of education report that 40% of all 4-year olds are enrolled in what is usually known as Junior Infants and almost all 5-year olds are enrolled in Senior Infants[1]. Schooling is done in a variety of

1 `https://www.education.ie/en/The-Education-System/Early-Childhood/`

settings, including mixed and single-sex schools, varying class and year-group sizes as well as public and private schools.

The six years of secondary school are referred to as 1st to 6th year[2]. Secondary school is split into the Junior and Senior Cycle. After completing three years of the Junior Cycle, students take the state-wide Junior Certificate Exams. Students may then enter the Senior Cycle which could be a two or three-year programme depending on the school. After completing this Senior Cycle, students take a state-wide exam known as the Leaving Certificate. In general, the subjects Mathematics, English and Irish are mandatory throughout primary and secondary school.

1.3.2 TRANSITION YEAR

As mentioned in Section 1.3.1 the Senior Cycle can generally take two or three years for students to complete. This difference is down to an optional Transition Year (TY) which can take place immediately after the Junior Cycle. TY provides an "opportunity for students to experience a wide range of educational inputs, including work experience, over the course of a year that is free from formal examinations"[3]. One of the benefits of TY is that many schools provide students with opportunities to study subjects that aren't on the Junior or Senior cycle curriculum.

1.3.3 COMPUTER SCIENCE IN IRELAND

In Ireland, as in other countries, Computer Science (CS), also referred to as computing or informatics, is not yet a state examination subject. Although steps to include it have been taken - it has just been introduced in the academic year of 2018-19 on a pilot basis in forty schools and will be examined for the first time in 2020 [73]. Prior to this, the only Computer Science course offered at second level was a Junior Certificate Coding short course [21]. This coding short course focuses not only on the writing of code but also on the problem solving aspect behind it. While this is a good first stage and one that may be beneficial to students in a variety of careers and paths in life, it is not close to a complete course on CS.

Research shows that an early introduction to computing is an advantage for students. It can build confidence in dealing with complexity and with open-ended problems [105]. Problem-solving skills can be extended and transferred [49] and students'

2 `https://www.education.ie/en/The-Education-System/`

3 `https://www.education.ie/en/The-Education-System/Post-Primary/`

analytical skills can be improved [55, 90]. It has been shown that students' self-efficacy for computational problem solving, abstraction, debugging and terminology can be increased [94]. It has also been found that teaching Computational Thinking (CT) can provide a better understanding of how programming is about solving a problem (not just coding) and that it can improve female students' attitudes and confidence towards programming [25]. One especially interesting finding is that exposure to CT can be used as an early indicator and predictor of academic success since CT scores have been found to correlate strongly with general academic achievement [39].

In Ireland, various attempts have been made to introduce CT into schools. One of these is the PACT programme which is a partnership between researchers in the Department of Computer Science at Maynooth University and teachers at selected primary and post-primary schools around Ireland. Its objective is to introduce students and teachers to CS through Programming and Algorithms, with the aim of improving CT skills in participating students. Now in its sixth year, the programme has been delivered in over 60 schools and to over 1000 students. The PACT team provides teachers with training and resources but it is the teachers who deliver the content to students. To date, most of the teacher-level instruction involves a short course in Python, an introduction to Algorithms, and the use of problems provided by organisations such as Bebras[4] to introduce both CT and computing concepts to students.

1.4 TRENDS IN COMPUTER SCIENCE EDUCATION AROUND THE WORLD

CS Education has been present in many forms for many decades. With the improvement and spread of technology over the past 20 years encompassing all areas of life, the demand for CS-competent workers has grown exponentially. To provide the training and skills required, a huge investment has been made in many countries to improve and expand CS education both in formal and informal education. This has been done through universities, government organisations, corporations as well as non-profit foundations. In most countries of the world, supply is still not meeting demand, with jobs such as software developers being in high demand and there just not being enough skilled graduates to fill the positions[5].

More time and focus is being given to introducing CS to students in all levels of education, as well as having more money invested to improving CS education. Recent

4 http://www.bebras.org

5 https://eu.usatoday.com/story/tech/talkingtech/2017/03/28/tech-skills-gap-huge-graduates-survey-says/99587888/

announcements by the governments of Chile[6] and the US[7] are just some examples of how government support to improve CS education is growing. Organisations such as code.org [8], Khan Academy [9] and CoderDojo[10] have expanded hugely in the past decade as they try to fill gaps left by formal education and provide teachers and students with the tools and resources to learn the skills desired by employers. In this section we will discuss a few of the major groups and organisations who are pushing CS education forwards and how they are doing this.

1.4.1 RESOURCES

In the last 20 years, a huge range of resources have been developed to enable educators to teach CS to students of different ages, abilities and interests. It is impossible to cover all of these. Even in the extensive literary review carried out only a handful could be studied and reviewed in detail. In this section we will briefly mention a few which directly impacted the design of CS2Go and which are popular around the world.

code.org

Code.org is a non-profit based in the US who believe that "every student in every school should have the opportunity to learn Computer Science". Through their annual *Hour of Code* campaign, they estimate that they have engaged 10% of all students in the world. They have designed a number of courses which are free and open source and are available online for schools and teachers. These courses ranges from primary school courses all the way up to university level courses. As well as building games, apps and teaching programming languages they have developed many "unplugged" activities to teach programming concepts and CT.

CS Unplugged

CS Unplugged[11] is a collection of free learning activities that teach CS through engaging games and puzzles that use cards, string, crayons and lots of running around. It was developed by the CS Education Research Group[12] at the University of Canterbury,

6 `https://bit.ly/2NQ3Ngw`
7 `https://bit.ly/2ugKvsS`
8 `https://code.org/`
9 `https://www.khanacademy.org/`
10 `https://coderdojo.com/`
11 `http://csunplugged.org/`
12 `http://cosc.canterbury.ac.nz/research/RG/CSE/)`

New Zealand, so that young students could interact with CS, experiencing the kinds of questions and challenges that Computer Scientists experience, but without having to learn programming first. In their paper, Bell, Duncan, and Atlas [7], who are some of the researchers responsible for the CS Unplugged project, give an initial overview of the project. They explore why CS Unplugged has become popular and describe different ways the resources have been adapted such as:

- The creation of videos of different activities
- The making of bracelets coded in binary
- Competitions
- Shows
- The adaption of CS Unplugged activities to different themes such as WWII
- Outdoor activities
- Online activities

A number of these lessons are used in the design of CS2Go, and the idea as a whole had a major impact on the lessons in CS2Go, especially in the Computer Science Concepts module.

Scratch

Scratch[13] is a visual programming language designed by MIT Media lab which was released in 2005. Scratch helps young people learn to think creatively, reason systematically, and work collaboratively through designing their own interactive stories, games, and animations. These creations and projects can then be shared on an online community, which at the time of writing has over 17 million registered users and over 20 million shared projects. Scratch has been used in many different ways to teach CT concepts.

App Inventor

MIT App Inventor[14] is a beginner's introduction to programming and app creation that transforms the complex language of text-based coding into visual, drag-and-drop building blocks. The simple graphical interface grants even an inexperienced novice the ability to create a basic, fully functional app within an hour of starting to use the tool.

13 `https://scratch.mit.edu/`
14 `http://appinventor.mit.edu/explore/`

1.4.2 ORGANISATIONS

There are many organisations who support, fund and equip various projects and groups who develop CS educational tools and resources. Groups such as Raspberry Pi, CoderDojo and code.org have been set up specifically to further CS education either worldwide or in their specific contexts. Large multi-national companies also put a lot of effort, money and resources into CS education. Google, Microsoft, Intel and many more both fund and develop training programs for teachers, tools to use (such as Arduinos) and give schools and students spaces to learn about CS (such as Microsoft's newly opened DreamSpace in Dublin[15]). As well as directly being involved, companies also sponsor research projects, conferences, individual schools and provide infrastructure as well as hardware and software to promote the use of IT in education more broadly.

1.4.3 EMPOWERING TEACHERS

One way which has proved popular to facilitate the teaching of Computer Science and Computational Thinking has been through teacher workshops. Studies have shown there are several ways in which teachers' enthusiasm for, knowledge of and ability to teach CT and CS in their classrooms can be improved/increased. Most popular seem to be day-long workshops and workshops that are heavily practical in nature. The ideas, tools and lessons that are given during these workshops seem to give teachers a greater understanding of what CT is and how it can be useful for their students, whilst also giving them very practical ways to implement this in a variety of contexts. Wide-reaching initiatives such as Google's CS4HS can help teachers that otherwise lack the skills and knowledge to teach CT topics.

Interestingly it seems that one significant barrier to CS and CT in education is teachers' and educators' misconceptions about what these are. One advantage of having teachers attend these training days and workshops is that these misconceptions and misunderstandings can be corrected, which is successfully done, according to the described studies. It can also be seen from studies that teachers' willingness and interest in teaching CS/CT is vital in its implementation in both primary, secondary and tertiary education.

15 https://www.rte.ie/news/business/technology/2018/0222/942659-microsoft-leopardstown/

1.4.4 HARD VS SOFT SKILLS

Historically, those who studied CS would have been taught how to program a computer from a manual or similar, and would become competent in the machine they were using. In the last 40 years, with the accelerated development of hardware and software, programming languages have developed which are cross-platform. This has led to CS students learning to program (usually with one specific language) and then adapting those skills to learn other languages and use them in other environments. The focus has generally been on programming skills alongside certain computer theory at the beginning of CS education. With the increased usage and availability of computers and other technologies, CS education has become more widespread and diverse. This can often lead to many students taking CS classes or modules who won't really need to know much more than the basics. As knowledge becomes more easily available, through online videos/courses/tutorials in particular, the teaching of a certain programming language or method can be done by the individual student at their own pace.

This leads to many good things as well as some obstacles. One of these is whether school and university courses should be teaching students so-called "Hard skills" or "Soft skills". A hard skill in CS would be, for example, teaching students Java programming or how to create websites using HTML. This type of teaching has its place, but it is our belief that we should be teaching CS students soft skills which enable them to develop the hard skills as they are needed. What this means practically is that we want to be developing students who are critical thinkers, problem-solvers, who are competent at analysing data and information and distilling conclusions from them.

This is one of the beliefs that underpins CS2Go. We do teach students CS skills and tools such as programming (through Scratch and Python) but a lot of the content, including programming modules, is designed to develop students' problem-solving or CT skills. This has lead to a large number of the lessons that have been developed not needing to be conducted on a PC or other device. This might seem counter-intuitive for a CS course but we believe that the concepts learned as well as the challenges presented to students will allow them to develop these "soft skills" and improve their ability to learn the "harder skills", such as programming, either through this course, or at a later stage. This methodology also allows CS2Go, we believe, to be of benefit to students who may never study CS, programming or anything related in the future, as problem-solving skills are useful in all other areas of education as well as in day-to-day life.

1.5 COMPUTATIONAL THINKING

Denning [26] suggested that Computational Thinking (CT) has been around since the 1950's as "algorithmic thinking", referring to the use of an ordered precise set of steps to solve a problem and where appropriate to use a computer to do this task. Papert [74] is credited as concretising CT in 1980 but it is since the contribution of Jeanette Wing [97], who popularised the term and brought it to the international community's attention, that more and more focus has been placed on CT within education. In her seminal paper, Wing outlined how she believed that all children should be taught CT, placing it alongside reading, writing and arithmetic in terms of importance. She further described it as representing a "universally applicable attitude and skill set everyone, not just computer scientists, would be eager to learn and use" [97].

Although academics have failed to agree on a universal definition of CT, Wing defines it as solving problems, designing systems, and understanding human behaviour, by drawing on the concepts fundamental to computer science. She states that CT it is not programming and that it means "more than being able to program a computer. It requires thinking at multiple levels of abstraction" [97]. In 2008 Wing posed a question to the computer science, learning sciences and education communities: "What are effective ways of learning (teaching) CT by (to) children?" [98]. This in turn raised further questions about what concepts to teach, the order in which these might be taught, and which tools should be used to teach them.

Based on these widely-accepted definitions of CT, for the purposes of this thesis we will define CT as "Using Computer Science concepts and the power of technology, combined with human creativity and ingenuity to solve problems". This definition encapsulates the idea of using CS concepts such as algorithms, data handling, efficiency etc. alongside the computational power that computers can give us. The idea is to harness these and combine them with human creativity and "outside-the-box" thinking to solve real-life problems.

In the meantime, a lot of work has been done around the world and across all levels of education to introduce CT into schools, colleges, after school clubs, mainly through CS or computing classes/courses. As CT is important to a computer scientist this makes sense; however, it should be noted that being able to think computationally, which includes skills such as decomposition, abstraction, algorithmic thinking and pattern matching, can be of benefit to all disciplines. Bundy [14] has made this point stating that CT concepts have been used in other disciplines and that the ability to think computationally is essential to every discipline.

The real benefits and importance of CT are currently hard to define. Some studies have shown potential for CT or problem-solving skills to be predictive of academic success in CS related courses [39, 55]. However, more study is needed to see whether teaching CT explicitly improves students' general problem-solving skills and whether it is just a sub-set of the general area of problem solving.

A wide array of topics have been used to introduce CT to students. In addition to explicitly teaching students what CT is [34, 54] students may be introduced to concepts such as abstraction [3, 85], modelling [16], algorithms [3, 29, 66], decomposition [3] and problem solving/critical thinking skills [83, 85].

1.6 PACT

As part of the movement to introduce CS into the Irish education system, researchers at the Department of Computer Science, Maynooth University, Ireland, designed the PACT programme. PACT is an acronym for Programming $^\wedge$ Algorithms $\approx$ Computational Thinking. Originally the hope was to introduce Irish secondary school students, and teachers, to Computer Science through both Programming and Algorithms, with the goal of improving the vital skill of CT in participating students. The PACT group now also work with primary schools and use CT as a means to bring CS to schools. Now in its sixth year, the PACT program has been delivered in over 60 schools and to over 1000 students.

1.7 THE CS2GO PROJECT

1.7.1 GENERAL AIMS

With the introduction of programming to the Leaving Certificate curriculum and the call from administrators and governments to include more Computer Science content in schools, we have developed a comprehensive course for Transition Year students. By teaching Computer Science topics including programming, web development and non-computer-based (unplugged) activities, our goal for this course is to develop students' CT skills as well as their knowledge of Computer Science.

We therefore see the main aims of this project as being:

- To introduce students to Computer Science, what it is, how it can affect their lives and how they can be involved

- To improve students' view of Computer Science. This includes changing views on the gender imbalance in CS and stereotyped "nerd" view
- To improve students' Computational Thinking and problem solving skills. We hypothesise that this will make them aware of a problem solving process and how it can be beneficial in many subjects and areas of life
- To teach students CS concepts such as Algorithms, Cryptography, Sorting/Searching, etc. with a focus not just on the concepts themselves but on real-world applications
- To give students an introduction to programming so that they can write their own basic programs i.e. variables, input, operators, loops and conditional statements

2 LITERATURE REVIEW

2.1 INTRODUCTION

At the commencement of the project, we needed to decide and research what the course content should be, how it should be presented and what tools to use. This was done through conversations with teachers and other educators who have experience in CS and general education. In addition a systematic literature review was undertaken to help influence the course development. This chapter presents the findings of this review in relation to second level CS education and Computational Thinking.

The hope was that this review, undertaken during October and November 2016, would also be of benefit to other educators, researchers, teachers and industry members who hope to introduce this vital 21st century skill to the next generation of learners.

2.2 RESEARCH QUESTIONS

The first stage of the systematic literature review was to design the research questions to be addressed. To this end, several research questions were defined to cover the potential studies involved in looking at CT in secondary education. For the purposes of this review, we took secondary education to be from age 11/12 until the completion of final country or state-wide exams at aged 17/18. The research questions decided upon were as follows:

- RQ1: What topics/subjects have been used to teach CT?
- RQ2: What tools/methods have been developed/used to teach CT?
- RQ3: What methods/tests/tools exist to test students' Computational Thinking ability/improvement?
- RQ4: Why is Computational Thinking important for educational institutions to incorporate into their curriculum? i.e. What benefits does CT have?

- RQ5: What problems/barriers are in place to incorporating Computational Thinking? Or what difficulties exist to introducing Computational Thinking into schools?

2.3 METHOD

2.3.1 INTRODUCTION

The systematic literature review process used was based on Kitchenham's method [48] as applied to software engineering. This method of performing a review was chosen as the process is well documented and is derived from review processes that were previously well established in the medical community. Kitchenham outlines how to identify the need for the review, how to develop a strict protocol to follow for the review and how to report the findings from the review. The following steps are listed in the method:

- Identify the need for a systematic literature review and define your research questions. Addressed in Section 2.1 & Section 2.2.
- Carry out an exhaustive search for studies. Described in Section 2.3.2, Section 2.3.3 and Section 2.3.4.
- Assess quality of accepted studies. Discussed in Section 2.3.5 and Appendix A.
- Extract data from accepted studies. Presented in Section 2.3.6.
- Summarise and synthesise study results. Discussed in Section 2.5.

The correct application of these steps leads to a rigorous, exhaustive and reproducible meta-review [48].

2.3.2 SEARCH TERMS

In this study, one primary search term was used. This was "Computational Thinking". Secondary terms were used along with this and were as follows: "school", "secondary school", "high school", "post-primary school", "middle school", "second level", "benefits", "assessment", "test".

2.3.3 RESOURCES SEARCHED

Using the search terms introduced above, an extensive search of the following databases was carried out between October and November 2016:

- ACM Digital Library
- IEEE Xplore
- ERIC
- Google Scholar

2.3.4 DOCUMENT SELECTION

Inclusion and exclusion criteria were developed as recommended in the systematic review guidelines [48]. Texts were included that:

- Directly answered one or more of the research questions
- Were related to the teaching of Computational Thinking in second level educational institutions

Studies were excluded that:

- Were in the form of a book or grey literature (opinion pieces, technical reports, blogs, presentations etc.)
- Were not published in reputable (i.e. peer-reviewed) sources
- Did not answer any research questions
- Were not published in English

The first step was to eliminate papers based on their titles using the above criteria. Abstracts of papers that made it through this stage were then read with further cuts being made based on whether the content of the paper was relevant. The papers that made it through these stages were then studied in detail and assessed as described in Section 2.3.5.

2.3.5 QUALITY ASSESSMENT

The following questions were taken from Kitchenham's framework [48] and each paper was analysed using them. The most relevant questions were taken from a set of 18 and applied to this review. These questions are:

- Question 1: How credible are the findings?
- Question 2: How well does the evaluation address its original aims and purpose?
- Question 3: How well was data collection carried out?
- Question 4: How clear and coherent is the reporting?
- Question 5: How has knowledge or understanding been extended by the research?

A scoring system was developed to analyse each paper based on these questions and this scoring system is described next (For a full overview of the quality assessment one should consult Table A.1 in Appendix A):

- Question 1: Y (yes), the findings are very credible due to where the study is published, P (partly), the findings are credible but the source is questionable, N (no), the findings not the source are credible.
- Question 2: Y (yes), the evaluation addresses the original aims and objectives, P (partly), the evaluation addresses the original aims and objectives implicit, N (no), the evaluation does not address the original aims and objectives.
- Question 3: Y (yes), the data collection was carried out well and outlined clearly, P (partly), the data collection was carried out well but not outlined clearly, N (no), the data collection was not carried out well.
- Question 4: Y (yes), the paper was clear and coherent, P (partly), the paper was somewhat clear and coherent, N (no), the paper was not clear and coherent.
- Question 5: Y (yes), knowledge or understanding has been extended, P (partly), knowledge or understanding have been somewhat extended, N (no), knowledge or understanding has not been extended.

The scoring used was as follows. Where a question received a Y, a score of 1.0 was applied, where a P was received a score of 0.5 was applied and where a N was received a score of 0.0 was applied. Papers that did not receive a rating of three or above were excluded.

2.3.6 ANALYSIS OF PAPERS

Figure 2.1 shows the year the papers that were analysed in this literary review were published. It can be seen that little research was carried out on CT relating to second level education prior to 2010. The amount of studies has generally increased in recent years. We suggest that the year 2016 is lower due to the timing of the searches, with some papers not having been published and/or made available at the time of search.

Figure 2.1: Year of papers published

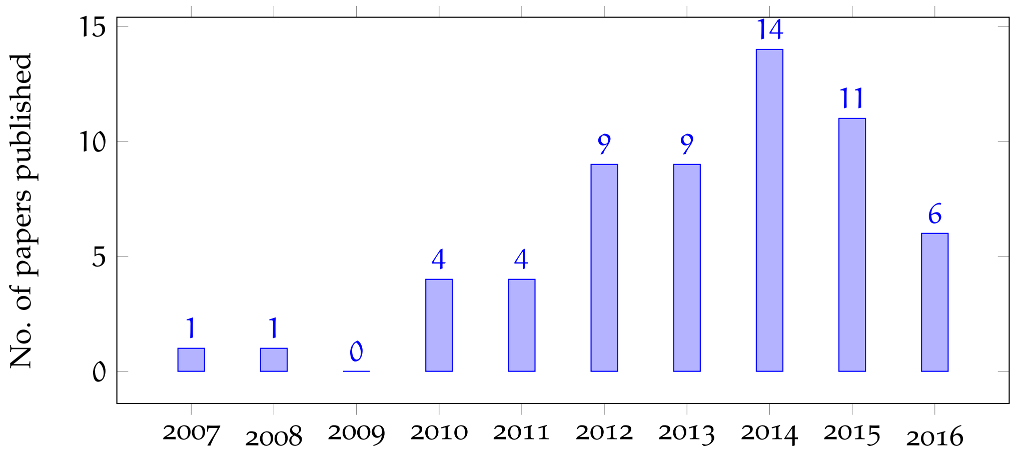


Figure 2.2 highlights the number of the papers included in the review by the database they were found in. One thing of interest is that no papers were included in the final corpus from ERIC. This was in part due to the small number found in the initial searches, only nine unique papers were returned using the search terms. We feel that this might show the need for more education researchers to be involved in studies on CT in secondary schools. Note: Many papers found in ACM and IEEE were also found through Scholar, the number listed for Scholar includes only those found exclusively there.

Figure 2.2: Number of papers sourced from each database

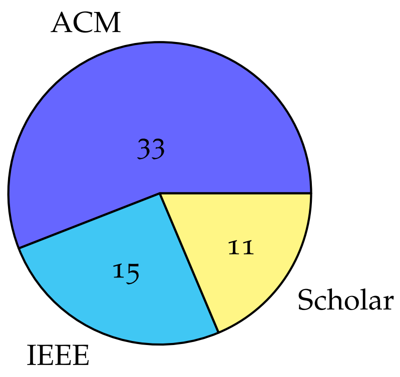


Table 2.1 and Table 2.2 provide further analysis of the papers included in this study. Table 2.1 provides a summary of all the papers including a general topic area and publishing location. These topic areas were assigned by us as reviewers and is given

to show the general focus of the paper. However, it may be that other topics are also covered in the papers and they are not limited to the topics given here.

Table 2.1: Overview of papers

Reference	Year	Publishing Location	Topic Area
Ahamed et al. [1]	2010	Conference	Teacher workshop, activity description, survey
Atmatzidou and Demetriadis [3]	2016	Journal	Teaching methodologies, results, definition of CT
Bargury et al. [4]	2012	Conference	Course design, teacher info
Basawapatna et al. [5]	2014	Conference	Learning methods
Basu, Kinnebrew, and Biswas [6]	2014	Conference	Assessment, tool, results
Blum and Cortina [10]	2007	Bulletin	Teacher workshop, survey
Brancaccio et al. [11]	2015	Conference	E-learning, teacher training
Carvalho et al. [15]	2013	Workshop	Challenges to CT, CT attempts in education
Caspersen and Nowack [16]	2013	Conference	Course design and learning methodologies
Chiprianov and Gallon [18]	2016	Conference	implementation, discussion
Cho et al. [19]	2014	Conference	Teacher workshop, survey
Cortina et al. [20]	2012	Conference	Teacher workshop, survey
Curzon [22]	2013	Conference	Activity description and outreach summary
Curzon et al. [23]	2014	Conference	Teacher workshop, survey
Davies [25]	2008	Conference	Teaching methodology, results
Falkner, Vivian, and Falkner [28]	2015	Conference	MOOC, course design, survey
Folk et al. [29]	2015	Conference	Course examples, teacher training

Continued on next page

Table 2.1 – continued from previous page

Reference	Year	Publishing Location	Topic Area
Fronza, El Ioini, and Corral [30]	2015	Conference	Course description, results
Grover and Pea [35]	2013	Conference	Activity description, learning method, results
Grover, Cooper, and Pea [34]	2014	Conference	Assessment, curriculum design, survey, tests
Grover, Pea, and Cooper [36]	2015	Journal	Course design, assessment, results
Haddad and Kalaani [39]	2015	Conference	Assessment, results
Imberman, Sturm, and Azhar [44]	2014	Journal	Tools
Jenkins, Jerkins, and Stenger [46]	2012	Conference	Teacher workshop
Kalelioglu, Gülbahar, and Kukul [47]	2016	Journal	Literary Review
Koh et al. [50]	2014	Conference	Assessment
Koh et al. [49]	2013	Conference	Course description, questions on it
L'Heureux et al. [52]	2012	Conference	Course design, activity design, survey
Lee et al. [53]	2011	Magazine	Activity suggestions
Li, Hu, and Wu [54]	2016	Conference	Activity description, survey
Lishinski et al. [55]	2016	Conference	Assessment, results
Lye and Koh [63]	2014	Journal	Literary review
Mannila et al. [64]	2014	Conference	CS education worldwide, survey
Mensing et al. [65]	2013	Conference	Tools, activity description
Mooney et al. [66]	2014	Conference	Course design, survey

Continued on next page

Table 2.1 – continued from previous page

Reference	Year	Publishing Location	Topic Area
Morreale and Joiner [68]	2011	Journal	Teacher workshop, survey
Morreale et al. [70]	2012	Journal	Teacher workshop, survey
Morreale, Joiner, and Chang [69]	2010	Journal	Teacher workshop, survey
Nesiba, Pontelli, and Staley [71]	2015	Conference	Course examples, teacher training, results
Pokorny and White [76]	2012	Journal	Teacher workshop, survey
Repenning, Webb, and Ioannidou [80]	2010	Conference	Tools
Ribeiro et al. [81]	2013	Workshop	Definition of CT, challenges to CT
Roscoe, Fearn, and Posey [83]	2014	Conference	Course description
Shailaja and Sridaran [85]	2015	Journal	Definition of CT, how can it be taught?
Sherman and Martin [86]	2015	Journal	Assessment, results
Sysło and Kwiatkowska [87]	2014	Conference	Activity design
Van Dyne and Braun [90]	2014	Conference	Course design, assessment, results
Vieira and Magana [92]	2013	Conference	Teachers workshop
Voogt et al. [93]	2015	Journal	Definition of CS, How can it be taught?
Webb and Rosson [94]	2013	Conference	Activity design, evaluation
Werner et al. [96]	2012	Conference	Assessment, results

Continued on next page

Table 2.1 – continued from previous page

Reference	**Year**	**Publishing Location**	**Topic Area**
Wolz et al. [99]	2010	Conference	Curriculum design, activity design, pilot study, survey
Wolz et al. [100]	2011	Journal	Teacher & student workshop, survey
Worrell, Brand, and Repenning [101]	2015	Conference	Assessment, teaching methodology
Yadav et al. [103]	2014	Journal	Teacher module, survey
Yadav et al. [102]	2011	Conference	Trainee teachers
Yevseyeva and Towhidnejad [105]	2012	Conference	Activity design
Zur Bargury [106]	2012	Conference	Curriculum description and evaluation

Table 2.2 provides a list of all the publishing locations of the papers that are included.

2.4 RELATED WORK

There have been several literature style reviews and overviews written on the current state of CT in schools in specific countries as well as suggestions for frameworks for CT. The following section presents several of these papers which were found during the search process. Some conclusions that can be drawn from these papers are as follows:

- Visualisation of the output of programming helps students learn CT [63]
- Having a scaffolding process and giving students an authentic problem helps students [63]
- The inclusion of CT aspects in the curriculum is relevant in all countries [64]
- CT concepts can be taught through various subjects [64]

- There are many positives to informal activities, however, they are not usually integrated into curriculum, so students may not see the connections of CT concepts. There is also no consensus on what should be taught [64]
- The top five skills related to CT, as found in the literature, are [47]:
  - Abstraction
  - Algorithmic thinking
  - Problem solving
  - Pattern recognition
  - Design-based thinking
- There are serious challenges in defining CT [93]. Suggested research gaps include:
  - Explore more class-based interventions [63]
  - Explore more studies in computational practices and computation perspectives [63]
  - Examine the programming process [63]
  - There is no accepted or well-known definition of CT and an agreed-upon definition would be helpful to educators [47]
  - More work is needed in relation to integrating CT in education and integrating a CT curriculum [93]

2.5 FINDINGS

2.5.1 RQ1: WHAT TOPICS/SUBJECTS HAVE BEEN USED TO TEACH CT?

A huge range of topics have been used to teach CT to students. This includes explicitly teaching what CT is [4, 36, 54, 106] and introducing them to concepts such as abstraction [3, 16, 83, 85], modelling [16], algorithms [3, 29, 34, 36, 37, 66, 71], decomposition [3] and problem solving/critical thinking skills [83, 85].

CT has also been taught through a wide range of CS topics, these include robotics [3, 4, 18, 83, 106], web development [4, 106], searching and sorting [29, 54, 105], circuit boards [83], logic [16, 36] and product design/software engineering [4, 16, 83, 106].

As well as teaching CT independently, or through CS, there have been many efforts made to teach it in existing second-level subjects. One paper [71] discusses the DISSECT project and they present one approach in which CT practices are combined with composition and literature through a 12th grade English Literature course. One example of this is a unit on the play Macbeth. Using a drag and drop comic creation tool called ToonDoo [45] students had to use algorithmic thinking to create a story board of the scene. Another example in the English domain is a summer school and after school program in Interactive Journalism [99, 100]. This was designed for middle school students and teachers to develop a competency and interest in CT. Students and teachers conducted research and interviews into how to develop news stories that are presented using Scratch animations, text and video. Others [87] discuss how CT concepts can be incorporated into traditional school mathematics. Examples of the topics and how they say they are linked to maths include:

- Representation of numbers – polynomials
- Reduction and composition - Given the sides of a triangle, is it a valid triangle?
- Approximation - rounding errors - quadratic equation

Other examples of the links CT has to Mathematics and English include Cartesian Coordinates and blogging [65].

One interesting study was a report on the FACT (Foundations for Advancing Computational Thinking) curriculum, developed for middle school by Stanford University, USA. The curriculum was trialled both as an online version and face-to-face and it was found that the online version worked as well if not better than the face-to-face version [36]. With the rising popularity of online courses and Massive Open Online Courses (MOOC), this is worth considering when developing any content.

It was also found that a collaborative classroom design helped students retain a high level of information in situations where they may not be in the class for the whole semester. This design also allowed late-comers to be integrated quite easily [101].

It can be seen from the presented papers that introducing CT does not have to be done exclusively through new courses or even through CS. CT is a skill that can be used in a possibly surprising range of disciplines and can benefit students studying in any area. The ability to break down a problem and develop a manageable solution is one that all students will find useful in both their academic and work lives. However, it must be acknowledged that CS is the most obvious place to teach CT to students. All the skills that make up CT (abstraction, algorithmic thinking etc.) are vital to a Computer Scientist. It is important though, that students who have never studied CS learn these skills, not only for their studies but also for their future careers.

2.5.2 RQ2: What tools/methods have been developed/used to teach CT?

Programming is the most popular way of implementing CT skills as well as being a great tool to show students how CT can be applied to real-world problems. Programming concepts such as conditionals [4, 16, 34, 36, 37, 83, 85, 106], iteration [4, 16, 34, 36, 37, 54, 85, 106] and data/information handling [4, 16, 34, 36, 37, 106] are often taught and can be either add-ons to CT or even perhaps key parts of it.

Secondary school programming has been taught using a variety of technologies including Scratch [4, 36, 52, 54, 94, 106], spreadsheets [4, 106], Python [11, 66, 85], Java [85], BASIC [85] and Visual Basic [85]. It is worth noting that it has been shown that using a graphical programming language (Scratch, for example) or making simple simulations might allow algorithmic skills essential to CT to be taught in a way that is not as intimidating to students as a textual language (Java, Python etc.) [5, 36, 83].

CT skills have also been taught using other "technologies" such as Minecraft [83], Printcraft [83] and unplugged activities [22, 29, 54] which include the popular "CS Unplugged" activities, magic shows, hunting for a thief in CCTV footage and even a physical "Doll-house" with its own unique set of rules. These kinds of activities usually promote interaction and one course which used App Inventor specifically emphasised interaction as important when teaching CT [35]. In their course, they included whole-class discussions and questions as well as working in pairs to develop apps. One thing noted about App Inventor is that it was found to compare favourably with Scratch and Alice but offers the benefits of having something very tangible to show [35]. Fronza, El Ioini, and Corral [30] also used App Inventor during their week-long summer school called Mobilise. Through this course, they aimed to use the "curiosity of students for developing mobile apps to introduce and teach CT via programming mobile applications". Both groups found that students enjoyed designing apps and the familiarity with apps can help to engage students [30, 35].

Another method of developing CT is through game design. One major group who consider this are the Scalable Game Design (SGD) group who are in the University of Colorado, Boulder, USA. They have created a spin-off company called AgentSheets Inc.[16] which has developed AgentCube and AgentSheets. These are tools that let people create their own agent-based games and simulations and publish them on the Web through a user-friendly drag-and-drop interface. They state that building games teaches students Computer Science concepts, logic, and algorithmic thinking. One

16 http://www.agentsheets.com/

study of note presents a checklist for "computational thinking tools" [80]. They claim that to have an impact, a CT tool used in K-12 should fulfil the following conditions:

- Have a low threshold (should be easy to learn)
- Have a high ceiling (should allow sophistication)
- Scaffolds Flow (progressing in sophistication should be as straight forward as possible)
- Enables transfer (can students apply what they learn to other scenarios)
- Supports equity (is it interesting & engaging)
- Systematic and sustainable (they need to be used)

A variety of tools have been developed to assist the teaching of CT. Although several of these are in the early stages of development, it is encouraging to see efforts to make CT fun and accessible to students of all ages, genders and abilities. The benefits for educators are many and include a variety of options of how to integrate CT into their classrooms. Whether in a computer lab, a regular classroom or outside, in a one-on-one session or with a class of 30+ there is a tool out there which will suit educators' needs; and if there isn't then the evidence suggests that there might well be soon.

2.5.3 RQ3: WHAT METHODS/TESTS/TOOLS EXIST TO TEST STUDENTS CT ABILITY / IMPROVEMENT?

One form of assessment that has been designed for secondary schools is discussed by Werner et al. [96]. In their course, students were taught using Alice and engaged with CT in a three-step progression called Use-Modify-Create [53]. At the end of the semester, students were given up to 30 minutes to individually complete the "Fairy Assessment". They designed this "Fairy Assessment" as an Alice program which they hoped would analyse thinking algorithmically and making effective use of abstraction and modelling. The assessment involved solving three tasks which occurred during the playback of a narrative scenario in Alice. Failure in any of the three tasks did not result in the inability to complete the other two and they believed that for students to perform well they would "have to understand the narrative framework of the story underlying the program and to understand existing program instructions which create the framework". They found a large variety of results but in general their findings suggest that the Fairy assessment is a promising strategy for assessing CT because it is "motivating"; only 30 of the 311 participants did not attempt to modify the program.

Another assessment was developed to calculate CT levels in students' models [6]. These models were developed in "Computational Thinking in Simulation and Model-Building" (CTSiM) which is a learning environment that combines CT with middle school science. They designed pre- and post-tests to measure both science content and CT skills. The models students developed were evaluated by comparing them against the "expert model" for that activity, which gave a "correctness" value.

Regarding the assessment of App Inventor projects, one paper introduced a rubric for analysing "mobile computational thinking" (MCT) [86]. They define MCT as a "superset of CT..., where the device changes location and context with its user". They claim that existing CT assessment tools do not cover these new ideas and therefore they tested their rubric in a mixed-major course that taught App Design using App Inventor. They measured 14 different properties.

This comprised of six "general CT" concepts:

- Naming
- Procedural abstraction
- Variables
- Loops
- Conditionals
- Lists

and eight MCT concepts:

- Screen interface
- Events
- Component abstraction
- Data persistence
- Data sharing
- Public web services
- Accelerometer
- Orientation sensors & location awareness

Each of these properties were rated on a "2-to 4-point scale, with increasing points representing more sophistication with the concept being measured". Detailed examples of this scaling system are presented in their paper.

The SGD group have also developed several ways of assessing students' CT skills, specifically when using their AgentSheets/Cubes game design software. Basawapatna et al. [5] wanted to see if students could recognise "Computational Thinking Patterns" (CTP), which they defined as "abstract programming patterns that enable agent interactions not only in games but also in science simulations". Some examples of CTP's are generation, absorption, diffusion and transportation, which they describe in more detail in the paper. The SGD group have also built on this work by developing a Cyberlearning tool to help teachers see which high-level concepts students have mastered and which they are struggling with as students code in real-time [50]. The system is called REACT (Real Time Evaluation and Assessment of Computational Thinking) and it displays the CTP's that students are currently implementing. REACT also shows the CTP's students haven't used, as well as the correctness of previously implemented patterns. REACT offers a variety of visualisation tools and is designed to be used with SGD teams AgentSheets & AgentCubes software.

Work relating to the testing and assessing of CT is in its infancy. Most of the examples presented in this section are in the early stages of development. Tools and methods do exist such as the "Fairy Assessment" and the tools developed by the Scalable Design Group but there is a need for more research in this area if CT is to become a common skill taught in schools. From the papers presented here as well as our experiences in teaching CS to students, we believe an adaptable system in which unique problems are produced for students to solve would be a hugely beneficial resource. This would allow for a range of problems that test all areas of CT but require no specific knowledge of the problem area. The Bebras[17] problems aim to do this and so it is our belief that these problems can be used effectively as a central part of a CT assessment. An assessment based on these problems was developed during this project is described in detail in Chapter 4.

2.5.4 RQ4: WHY IS CT IMPORTANT FOR EDUCATIONAL INSTITUTIONS TO INCORPORATE INTO THEIR CURRICULUM? I.E. WHAT BENEFITS DOES CT HAVE?

Research shows that the earlier we can introduce students to computing, the sooner we can get them attracted to the field and that it can have several side effects such as building confidence in dealing with complexity and dealing with open-ended problems [105]. Problem-solving skills can be extended and transferred [49] as well as

17 https://www.bebras.org/

improving students' analytical skills [55, 90]. In relation to these, it has been found that students' self-efficacy for computational problem solving, abstraction, debugging and terminology can be increased [94]. It has also been found that teaching CT can provide a better understanding to people that programming is about solving a problem and not just the code and improve female students' attitudes and confidence towards programming [25].

One especially interesting finding is that CT can be used as an early indicator and predictor of academic success and that CT scores correlate strongly with general academic success [39]. Another finding to note is that it was found that CT skills significantly improved as training proceeded and that students developed the same level of CT skills at the end, independent of age (junior high (15-year-olds) vs high vocational (18-year-olds)). The authors administered tests after four classes and after ten classes and found a significant improvement in results. They suggest that this shows CT skills need a considerable number of sessions to develop [3]. It was also found that it can be difficult for students to transfer from one platform to another. Sometimes sticking to one and using scaffolded examples is a good option [94].

Although some encouraging signs have been seen, CT and research into it are still in the early stages. Therefore, long-term effects, as well as additional benefits still need to be researched further. The above findings are encouraging and show that CT is a beneficial skill to all, though more research is required before the extent of the impact of teaching CT can have on students is known.

2.5.5 RQ5: WHAT PROBLEMS/BARRIERS ARE IN PLACE TO INCORPORATING CT? OR WHAT DIFFICULTIES EXIST TO INTRODUCING CT INTO SCHOOLS?

Although becoming more commonplace, the introduction of CT into second level education has been slow. This could be down to many reasons and some provided in the literature include:

- Lack of infrastructure i.e. a lack of computers [15]
- Rearrangement of the curriculum i.e. should CT be a new subject or incorporated into others? [15, 81]
- Government Policies are also cited as a potential issue and it is remarked that it is important that governments build workgroups to lead the inclusion of CT into education [11, 81]

- People, and traditions in learning, mean that people might be reluctant to take up CT as they might be hard to convince that it is a skill they don't already possess [81]
- Students may not show much interest in learning [66]. One suggestion to counter this is via "stealth teaching". This is where the technical concepts are hidden to begin with and the student's interest is caught by the topic first [105]

Two other major roadblocks are the qualification and backgrounds of instructors i.e.
1) Do they need a computing degree? [4, 15, 81, 106] and,
2) Strategies to disseminate CT i.e. workshops, textbooks etc. [4, 15, 106]

For these two barriers a number of factors come into play. Firstly, in the Irish context, with CS being introduced into the Leaving Certificate there is a need for teachers to be trained and equipped to teach CS now, rather than waiting for them to complete a CS degree. CS training for teachers can be done in many different ways and this is discussed in more detail below. From researching different educational contexts, there is a clear move to have more and more teachers study CS during their teaching qualifications, as would be done with any of the other traditional sciences. However, the push to introduce CS to students as soon as possible has lead to the development of a wide variety of tools, textbooks and training's to allow teachers to up-skill.

This leads to the question of how to give teachers, both trained and currently training, the skills, resources and knowledge to incorporate CT into their classrooms. To this end, many training workshops have been run and below are studies found on these during this literary review.

The main aims of the workshops seem to be to inform teachers about what CT is, how it can be useful and then give them practical training to help integrate it into their classrooms. Information on what CT is, why it should be taught are a common starting point [10, 23, 68–70, 92]. Other topics included career opportunities [10, 70, 76] and females in CS [76].

Most workshops appear to be multi-day events, usually occurring in the summer. Most common were two to four daylong [1, 10, 19, 20, 92] workshops though a number were also a series of sessions [23, 68–70]. What all had in common was that they contained a lot of practical, hands-on sessions where teachers are given a chance to try out different methods of teaching CT. Another way this has been done is through MOOC [28]; modules were developed and then videos made on those topics. They also used a Google+ group to foster collaboration.

A common theme was teaching educators the different ways in which CT can be taught through different programming languages and technologies. These included

Scratch [19, 44, 76, 92], Alice [20, 68–70], Python [1, 20, 46], Java [20, 68–70] and App Inventor [19, 44]. Another popular method was showing examples of "Unplugged" activities, where CT is taught without the need for computers [10, 19, 23, 44, 68–70, 76, 103]. Examples included magic shows [23], games [23] and roleplaying [103]. This also includes teaching CT to those with no experience or through different subjects. Examples of this include DNA strings and matching algorithms [10], food [10], Harmonic Oscillation simulation [68], friction simulation [1], Towers of Hanoi [102], debugging (how to fix a lamp) [102, 103] and mathematical topics [1, 46]. A final common aspect of these workshops is that many offered the teachers an opportunity to design lesson plans with the help of their colleagues as well as the workshop staff [1, 19, 20, 28, 76, 92].

One slightly different course is a one-week module designed for trainee teachers [102, 103]. This did not include training in programming languages or technologies but focussed on day-to-day examples. The students were introduced to ideas such as problem identification, decomposition, algorithms and debugging. They were given examples such as teaching algorithms through kinaesthetic activities and were shown a recursion example using the Towers of Hanoi [103].

Some problems arose after these courses, which are important to highlight to teachers wanting to integrate CT into their educational institutes. One paper noted a lack of support from IT departments (for example when wanting to use Scratch) and the difficulty in and lack of time to refine lesson plans and develop competency in the material [19, 76]. This shows the need for the whole school/group to buy into the idea of CT and to understand what it is and its potential benefits. Another issue raised which links in with support from the whole school is that one group of teachers felt a classroom assistant would be beneficial [19].

Other findings from these workshops were a change in educators' views of what CT is [10, 69, 70, 76, 92, 102, 103]. This usually meant a shift from either no knowledge of CT or thinking CT is programming (or at least heavily related to it) to an understanding of its application as a way of thinking/problem solving which can be applied to many parts of education and life. The first point, of teachers lacking understanding of what CT is, and even what CS is, is especially important for researchers and curriculum developers to note. Without an understanding, teachers will struggle to understand why they should take time to learn and teach these concepts. It also gives rise to the need for the area to come to a consensus of what we mean by CT, without a definition it is hard for integration and change to occur. Also, the practical sessions are often seen as hugely beneficial [10, 28, 70] and give teachers the confidence and opportunities to introduce the lessons and ideas discussed into their classes [28, 70, 76, 103].

From these various papers and studies there are several ways in which teachers' enthusiasm for, knowledge of and ability to teach CT and CS in their classrooms can be improved/increased. The most common seem to be day-long workshops and workshops that are heavily practical in nature. The ideas, tools and lessons that are given during these workshops seem to give teachers a greater understanding of what CT is and how it can be useful for their students, whilst also giving them very practical ways to implement this in a variety of contexts. Wide-reaching initiatives such as Google's CS4HS can help teachers that otherwise lack the skills and knowledge to teach CT topics. Interestingly, it seems that one significant barrier to CS and CT in education is teachers' and educators' misconceptions about what these are. One advantage of having teachers attend these training days and workshops is that these misconceptions and misunderstandings can be corrected, which is successfully done in most of the described studies. It can also be seen from these papers that teachers' willingness and interest in teaching CS/CT is vital in its implementation in secondary education.

2.6 DISCUSSION

From the papers found and discussed in this review, the following findings are of significance:

- More work needs to be done on the assessment of CT
- CT is becoming more common place, but problems still exist
- Teacher workshops are vital and very effective
- Programming is key to advancing CT skills, but the basics and introduction can be done without computers and this needs to be done more
- CT appears to have benefits, but further research is needed, especially on long-term effects

We will now briefly discuss each of these findings.

2.6.1 MORE WORK NEEDS TO BE DONE ON THE ASSESSMENT OF CT

Currently, there are several tool-specific assessment methods, such as those developed for Agentsheets, plus adapted methods such as Bebras problems. It is hard to judge their effectiveness at assessing CT and more work is required. This research gap demands attention as the widespread acceptance and introduction of CT requires educators to be able to assess students' abilities in this, as well as allowing researchers and educators to be able to assess their own courses and curricula.

2.6.2 CT IS BECOMING MORE COMMON PLACE, BUT PROBLEMS STILL EXIST

The introduction of CT, whether through CS, independently, or through other subjects, is becoming more common place. This is encouraging from both a CS perspective (as it suggests more students will be introduced to CS) and an educational perspective (as it could lead to more analytically capable students). However, problems such as defining CT, disagreements on whether programming is an essential part of CT, and whether CT should be taught alongside pre-existing subjects, part of CS, or as a standalone module/course must be researched further.

2.6.3 TEACHER WORKSHOPS ARE VITAL AND VERY EFFECTIVE

Teacher workshops that aim to inform, teach and instruct teachers about CT are the topic of many of the studies presented here. It can be seen from these studies that workshops are extremely effective at changing teachers' perceptions of what CT is and how it can be taught. They are also important in equipping teachers who don't have a technical background in CS to know how to teach CT through concepts such as programming. The potential for more long-term and formal teacher training is something that researchers should look at in the coming years.

2.6.4 PROGRAMMING IS KEY TO ADVANCING CT SKILLS, BUT THE BASICS AND INTRODUCTION CAN BE DONE WITHOUT COMPUTERS AND THIS NEEDS TO BE DONE MORE

It can be seen from the number of studies, courses and workshops that incorporate programming in one guise or another that it goes hand-in-hand with teaching CT. However, it should be noted that CT skills, and even CS concepts, can be taught without programming and even without computers. Programming is one way in which CT skills can be evaluated as a student's ability to program a solution to a problem shows that they can use key skills such as abstraction, algorithmic thinking and decomposition. It is important that as work continues that the balance between using programming and teaching students to Computational Thinkers is struck. Although a useful skill, programming is more domain specific whereas it is widely agreed that CT is a skill that can be used across the educational spectrum and everyday life.

2.6.5 CT appears to have benefits, but further research is needed, especially on long-term effects

Due to the research of CT being in its infancy, very few long-term studies have been undertaken to see whether the teaching/learning of CT at different stages has an impact on things like academic success, career or study paths or even long-term problem-solving skills. Although it is commonly viewed as an essential skill that all students should be learning, so far, the evidence to support the inclusion of CT alongside reading, writing and arithmetic is limited, and more longer-term studies must be conducted to support this claim.

2.7 Conclusion

The aim of this literature review was to influence the development of CS2Go and provide other teachers and researchers with an overview of the current research and work around teaching CT at second level education. As stated in the introduction, CT is an important skill that we should be teaching students of all ages. To that end, the research questions posed in this paper were selected with the hope that educators from all contexts will be able to find examples of how CT can be incorporated into their classroom. Whether it is through a dedicated CT course/module, an already existing subject or just as a one-off event, CT can be taught in a fun and engaging way.

In addition, whilst learning CT, students can gain vital skills that can be applied across the curriculum as well as in daily and work life. There are also several different tools and ways in which one can apply them in classrooms, and although more work is needed, there are ways in which CT can be assessed. However, there are potential obstacles and difficulties when attempting to integrate CT. These include a lack of trained teachers and potential difficulties with government policy or school administrators. This literary review will also form the basis for a curriculum designed to teach CT and the hope is that providing teachers with a curriculum will lessen other common problems. These problems include preparation time and a fear in teaching these concepts, usually based around a lack of understanding. Until these problems are resolved it will be difficult for the widespread introduction of CT to take place.

2.8 publications released since the completion of the systematic literature review

Due to the nature of this project, the majority of the literature review was conducted, as previously stated, in October and November 2016. Since then papers and studies on CT and CS Education have been published in journals and conferences around the world. Some of these are referenced throughout the following chapters. Below is a list of selected publications released since the review was undertook. This is by no means an exhaustive list and the quality of these papers has not been analysed in as much detail as those in the review.

[31] Francisco José García-Peñalvo. "Computational Thinking". In: (2018)

[13] Francisco Buitrago Flórez et al. "Changing a generation's way of thinking: Teaching computational thinking through programming". In: *Review of Educational Research* 87.4 (2017), pp. 834–860

[27] Peter J Denning. "Remaining trouble spots with computational thinking". In: *Communications of the ACM* 60.6 (2017), pp. 33–39

[17] Guanhua Chen et al. "Assessing elementary students' computational thinking in everyday reasoning and robotics programming". In: *Computers & Education* 109 (2017), pp. 162–175

[8] Marina Umaschi Bers. *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge, 2017

[104] Aman Yadav et al. "Computational thinking in teacher education". In: *Emerging Research, Practice, and Policy on Computational Thinking*. Springer, 2017, pp. 205–220

[51] Özgen Korkmaz, Recep Çakir, and M Yaşar Özden. "A validity and reliability study of the Computational Thinking Scales (CTS)". in: *Computers in Human Behavior* 72 (2017), pp. 558–569

[84] Olgun Sadik, Anne-Ottenbreit Leftwich, and Hamid Nadiruzzaman. "Computational Thinking Conceptions and Misconceptions: Progression of Preservice Teacher Thinking During Computer Science Lesson Planning". In: *Emerging Research, Practice, and Policy on Computational Thinking*. Springer, 2017, pp. 221–238

[75] Bjarke Kristian Maigaard Kjær Pedersen et al. “Towards playful learning and computational thinking—Developing the educational robot BRICKO”. in: *Integrated STEM Education Conference (ISEC), 2018 IEEE*. IEEE. 2018, pp. 37–44

[40] Ting-Chia Hsu, Shao-Chen Chang, and Yu-Ting Hung. “How to learn and how to teach computational thinking: Suggestions based on a review of the literature”. In: *Computers & Education* 126 (2018), pp. 296–310

[41] Aleata Hubbard. “Pedagogical content knowledge in computing education: a review of the research literature”. In: *Computer Science Education* (2018), pp. 1–19

[72] Tom Neutens and Francis Wyffels. “Bringing Computer Science Education to Secondary School: A Teacher First Approach”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM. 2018, pp. 840–845

[95] Mary Webb et al. “Computer science in K-12 school curricula of the 2lst century: Why, what and when?” In: *Education and Information Technologies* 22.2 (2017), pp. 445–468

[38] Yasemin Gülbahar and Filiz Kalelioğlu. “Competencies of High School Teachers and Training Needs for Computer Science Education”. In: *Proceedings of the 6th Computer Science Education Research Conference*. ACM. 2017, pp. 26–31

Table 2.2: Publishing location

Location	No. of papers
ACM Annual Southeast Regional Conference	1
ACM Inroads	1
ACM Transactions on Computing Education	2
Australasian Computing Education Conference	2
Baltic Journal of Modern Computing	1
Computer Science Education	1
Computers in Human Behavior	1
Constructionism and Creativity	1
Education and Information Technologies	1
Emerging eLearning Technologies & Applications	1
Frontiers in Education (FIE)	7
Integrated STEM Conference (ISEC)	1
International Computer, Software & Applications Conference	1
International Conference on Engaging Pedagogy (ICEP)	1
International Conference on Interactive Technologies & Games	1
International Journal of Advanced Networking & Applications	1
ITiCSE	5
Journal of Computing Sciences in Colleges	6
Robotics and Autonomous Systems	1
SIGCSE	12
SIGITE	2
Society for IT & Teacher Education International Conference	1
International Conference on Intelligent Tutoring Systems	1
Visual Languages and Human-Centric Computing	2
WiPSCE	3
Workshop-School on Theoretical Computer Science	2

Part II

DEVELOPING THE COURSE

3 COURSE DEVELOPMENT

3.1 introduction

As described in Chapter 1, the main aim of this project was to develop a course to introduce students to Computational Thinking and Computer Science. The aim was to do this through developing a series of lessons which would allow teachers to take the developed content and easily use it within their own class. The hope was to make the course enjoyable and accessible, which would allow students to be introduced to topics in CS which they may have never encountered before. The course content was chosen based on the literature reivew described in Chapter 2. Along with this, in September 2016, teachers who had previously been involved with the PACT group, as well as others (including trainee teachers) were asked for their ideas on course design and content. This feedback, in conjuction with input from our group members and previous experience of teaching CS, led to the aims of the course described previously in Chapter 1. In this chapter the rationale behind the chosen course topics and developed lessons will be described as well as brief descriptions of each of the topics.

3.2 external tools and resources used in cs2go

In this section we will look at the external tools, resources and methods used to teach the various modules and lessons that have been developed for CS2Go. This includes online tools, previously designed activities and software developed for educational purposes. As CS Education is a growing area, there are a wide variety of top-quality resources available to use. It was also clear from the literature review, as presented in Chapter 2, that a huge variety of tools and methods had been used to teach CS and CT. Rather than "re-inventing the wheel" these have been incorporated into CS2Go to further different topics and lessons. Credit is given to the original creators wherever possible, a list of these can be found in Appendix B. When developing new content, or incorporating other ideas, the checklist for "Computational Thinking tools" was used when choosing tools and designing lessons (see Section 2.5.2). Specifically the conditions of "have a low threshold", "scaffolds flow" and "supports equity", were key in the development of content.

3.2.1 COMPUTER SCIENCE CONCEPTS

The lessons within the *Computer Science Concepts* module are all designed to be run in an open-space classroom and require no computers other than one for the teachers to project details to a screen. The lessons also often require activity sheets which will have to printed or prepared prior to the class. During the course of the literature review, and from previous teaching experiences, many of these type of activities were discovered. It was felt that this would be a very beneficial way to begin the CS2Go course. Many students in the Irish context, as is the case across much of the world, have not interacted with CS as a subject before. Introducing CS through interactive and hands-on activities which don't require PC's can allow all schools, teachers and students to engage with CS. This can then be used as a base to go onto more topics such as programming. For this module, several outside sourced activities are used. There is a fantastic and growing amount of resources available for teaching CS to both primary and secondary school students without the need for PC's. All credit is given to the original designer when used and permission was sought on top of the already existing creative commons licensing on all the resources.

The topics chosen for this section, which are shown in Table 3.1, were based off of the literature review, available resources (described in the following subsections) and previous teaching experience. Topics such as Algorithms, Variables and User Input, Introduction to CS and Introduction to CT were all topics that were found to occur in similar courses around the world (see Section 2.5.1. Topics such as Cryptography and Searching and Sorting were seen as interesting and engaging as well as being key in other similar courses such as the CS Unplugged content. Others were based of of previous teaching experience and ideas that were thought up during the course development stage.

CS Unplugged

CS Unplugged[18] is a collection of learning activities that teach CS using "engaging games and puzzles that use cards, string, crayons and lots of running around". It was developed by the CS Education Research Group at the University of Canterbury, New Zealand. They developed the resources so that young students could interact

18 https://csunplugged.org/en/

with Computer Science and could experience the kinds of questions and challenges that computer scientists experience, but without having to learn programming first. During the literature review a number of studies on the CS Unplugged activities were read and critiqued. The findings were positive with the lessons being enjoyable and informative and this lead to many of the activities being incorporated into the CS2Go course. It also led to the development of similar "unplugged" lessons.

Teaching Computing London

Teaching Computing London is a partnership between Queen Mary University of London and King's College London[19]. It is a resource hub which is supported by and contributed to by many different groups. They provide a large repository of activities and lessons which teachers are free to use in their classrooms. As with the CS Unplugged section a number of papers from the participants of Teaching Computing London were read during the literature review. The results and insights in these papers influenced the design of this course to include more unplugged lessons.

code.org

Code.org[20] is a non-profit based in the US who believe that "every student in every school should have the opportunity to learn computer science". Through their annual *Hour of Code* campaign, they estimate that they have engaged 10% of all students in the world. They have designed a number of courses which are free and open source and are available online for schools and teachers. This ranges from primary schools courses all the way up to university level courses. As well as building games, apps and teaching programming languages they have developed many "unplugged" activities to teach programming concepts and Computational Thinking.

Bebras

Bebras[21] is an "international initiative aiming to promote CS and computational thinking among school students at all ages". The Bebras competition is held annually in many countries. Bebras problems are designed to introduce CS concepts and to test CT skills that do not require any prior technical knowledge. They are designed to be done using pen and paper but can also be made into interactive games, each question is short and focusses on one Computer Science topic. The Bebras problems have been

19 `https://teachinglondoncomputing.org/`
20 `https://code.org/`
21 `https://www.bebras.org/`

used extensively by the PACT group and this, along with studies found during the literature review (and described more in Chapter 4), lead to their inclusion not only in some of the lessons but also as the basis for some of the course assessment.

3.2.2 programming

Two of the modules designed for CS2Go are programming modules. The first is using Scratch and is designed to be an introduction not only to Scratch but also to programming concepts. The second uses Python, specifically Python Turtle. This is designed to build on any acquired knowledge from the Scratch module but can also be taught separately. As discussed in Section 2.6.4, programming is one of the key skills to help improve CT. This, along with the findings on each tool (Scratch and Python) discussed below, lead to the development of these two modules. Alongside this, personal experience in teaching both Python and Scratch to beginner programmers of all ages, including the target TY age group, led to these two tools to be chosen.

Scratch

Scratch[22] is a visual programming language designed by the MIT Media Lab's Lifelong Kindergarten group, which was originally released in 2005, with Scratch 2.0 being released in 2013. Scratch helps young people learn to think creatively, reason systematically, and work collaboratively through designing their own interactive stories, games, and animations. It is available in both browser and desktop form and this allows schools and students to be flexible in how they use it. For those with poor broadband or internet connections then they can download the desktop version and avoid potential network issues. For those where this is not a problem, students can use the online version and can easily access the online community, remix templates and projects as well as access their own projects from home. If a school opts for the desktop version then students can take their programs home by saving them onto a USB drive, emailing them or storing them in a cloud drive.

As of 28th August 2018, there have been over 33 million projects shared and over 30 million register users. In July 2018 there was over 300,000 new users, nearly 500,000 new projects and over 3.1 million comments[23]. Although targeted at primary school aged children, it is our belief from previous experiences and from feedback from teachers and other educators that Scratch can still be a valuable tool for secondary

22 https://scratch.mit.edu/

23 https://scratch.mit.edu/statistics/

school aged students. It has been shown that using Scratch can allow students to learn programming concepts in a less-threatening way than a textual language [5, 36, 83]. Scratch is also a very powerful language and, although often used to make games, there is very little to limit the programs that can be designed. From the literature review there were many examples of how Scratch has been used in a variety of educational contexts (see Section 2.5.2). These findings including practical teaching experience made using Scratch an obvious choice for the introductory programming module of CS2Go.

Python

Python is a scripting programming language which was first released in 1991. Since then it has grown in popularity and usage both in industry and as an introductory programming language. Reasons for this include its high-level syntax, this allows the concepts to be easily taught without being bogged down in syntax with a language like C or Java. It is multi-platform and so can be used across all operating systems and, as they state it "plays well with others" which allows it to be used across many different systems. It is also relatively easy to write quite powerful applications and functions without massive amounts of code; this allows learners to make quick progress. It is also a highly desirable language in industry with firms such as Google and Industrial Light & Magic using it in their applications. Recent surveys show that Python is increasing in popularity within employers and as a first programming language in educational institutions [88]. Python was also one of the common programming languages used in studies found during the literature review (see Section 2.5.2).

The suggested software tool for the CS2Go Python module is a website, `repl.it`. Whilst researching and investigating how best to teach the Python module several options were found. One option is to have students install Python 3 using Anaconda. We had previously used this to teach students in a CS001 course which used similar material. The CS001 module is an introductory CS module which is part of Maynooth Univeristy's Certificate in Science Course [24]. It is designed for students to take before entering a full-time degree program in science. In the CS001 module students cover topics such as the history of computers, operating systems and algorithms before spending a significant amount of time on programming.

Anaconda allows all the modules and libraries that would be needed to be installed in one go, and it also provides an IDE called Spyder which could be used. One draw-

24 `maynoothuniversity.ie/study-maynooth/undergradute-studies/courses/certificate-science`

back to this (or any installation of software) is that some schools and students might not have the facilities to do this. If schools only have older machines, then installing a large program might not be possible and this would prevent them from using this module of the course. The other issue is if students don't have PC's or laptops at home that can run Python efficiently then the same problems occur. Another issue is the grading and marking of assignments. If students save them as .py file on either local machines or an internal system, this will most likely require the teacher to download and possible run all these files individually, which is a time-consuming process.

We wanted to avoid this problem and so searched for browser-based solutions. One which was recommended was `repl.it` which is hosted in the cloud and so this allows you to log in and work on your programming sessions/problems anywhere. They allow for teacher and student registration which means that teachers can organise their class and keep track of student's progress. Exercises (such as the one's designed for this course[25]) can be designed and published in the classroom and students can then attempt them. Once they've submitted a solution their teacher will be notified. Teachers can then run and grade the students' work. This could be assigning it a pass mark or sending it back to the student with feedback and tips on how to improve it. For certain languages auto-grading is also available. `repl.it` also allows for Python turtle which was a necessity for this course, however it needs to be manually graded.

Another feature that was helpful was allowing teachers to collaborate on classrooms which allows multiple teacher accounts to access the class. This could mean, for example, if a teacher needed help grading assignments, multiple teachers could do this. This feature used to be free but currently has been removed and is only available for schools that sign up to a pricing plan. There is also now a limit of 30 students per class for a free account.

3.2.3 WEB DEVELOPMENT

Mozilla Thimble[26] was chosen as the tool to introduce students to web development and website design. Thimble is a full-featured code editor that runs in the browser. It is designed to "help new coders create their own sites and web-based projects using HTML, CSS & JavaScript". It is completely free and allows students to publish their sites live on the web. As it runs in the browser it is easy for students to continue working on their projects at home and in the classroom. It also easily allows students

25 `repl.it/community/classrooms/87698`
26 `https://thimble.mozilla.org/en-US/`

to "remix" projects, that is, to take other peoples projects and edit them. It also allows students to add files and has a tutorial feature that allows teachers to make their own tutorials which students can work through.

3.3 LENGTH & ORDER

The original plan was to develop a course that would be 20 hours long. This would provide teachers with enough content for anywhere from 4-20 weeks depending on the frequency and length of their classes. It would also allow enough content that topics could be picked depending on the teacher's prior knowledge and experience as well as the facilities available to them. However, considerably more content was developed than this over the course of the research. A total of approximately 85 hours of content was created. This has allowed more topics to be covered and in more depth. It also allows teachers to be more flexible in choosing the lessons they wish to teach. Details relating to the created lessons and associated times can be seen in Table 3.1.

The number of lessons refers to the number of separate lesson plans that were developed to teach each topic. In general, no lesson plan is shorter than 40 minutes (the minimum class slot in secondary schools), with the aim for most to be about an hour in length. An exception to this would be the Python and Scratch Projects as well as website design.

As explained in more detail in Section 3.4.3, the Python labs are "question-only" exercise sheets, not full lessons. The exercises are based on the topics which would be taught prior to the students attempting the lab sheets. For ease of viewing, Table 3.1 presents the lessons in the order taught, followed by the lab which corresponds to those lessons. For example, the first two lessons are "*Introduction*" and "*Variables & Expressions*" respectively, and these are followed by Lab 1.

The Intended length column in Table 3.1 describes the expected amount of time taken to teach each lesson. For example, although there is up to five hours of activities for *Cryptography*, if designing a timetable, teachers would be recommended to spend two hours on this topic. This is just a guideline for teachers and they are free, and encouraged, to choose lessons which suit their class, content and knowledge level.

The Approx. maximum length column in Table 3.1 is an estimate of the maximum amount of time that could be spent on a topic or lesson. This is informed through teaching the lesson. For example, different teachers have shown that lessons in the programming modules can take much longer than planned or anticipated. This is neither a bad nor good thing, as all students and classes will work at their own pace. For lessons which have not been tested, the lengths are inferred either from similar lessons (such as other Scratch lessons), advice from teachers and other experts as well as personal experience.

Table 3.1: CS2Go Course Content

Topic	**No. of lessons**	**Intended Length**	**Approx. maximum length**
Computer Science Concepts	12	10 hours	14 hours 35 mins
Algorithms	2	2 hours	2 hours
Cryptography	4	2 hours	5 hours
Data & Binary	1	1 hour	1 hour 15 mins
Finite State Machines	1	1 hour	1 hour 20 mins
Introduction to CS	1	1 hour	1 hour
Introduction to CT	1	1 hour	1 hour
Searching and Sorting	1	1 hour	2 hours
Variables and User Input	1	1 hour	1 hour
Scratch	5	11-13 hours	26 hours
Cat and Mouse	1	1 hour	5 hours
Guessing Game	1	2 hours	4 hours
Polygon Drawer	1	2 hours	4 hours
Pong	1	2 hours	4 hours
Project	1	4-6 hours	9 hours
Python	15	23 hours	39 hours
1) Introduction	1	30 minutes	1 hour
2) Variables and Expressions	1	30 minutes	1 hour
Lab 1	1	1 hour	2 hours
3) Strings	1	30 minutes	1 hour
4) Keyboard Input	1	30 minutes	1 hour
Lab 2	1	2 hours	3 hours
5) Turtle Graphics	1	1 hour	2 hours

Continued on next page

Table 3.1 – continued from previous page

Topic	No. of lessons	Intended length	Approx. maximum length
Lab 3	1	2 hours	3 hours
6) Loops	1	1 hour	2 hours
Lab 4	1	2 hours	3 hours
7) Selection	1	1 hour	2 hours
Lab 5	1	2 hours	3 hours
8) Functions	1	1 hour	2 hours
Lab 6	1	2 hours	3 hours
Project	1	6 hours	10 hours
Web Development	4	2 hours 45 mins	5 hours
Databases	1	1 hour	1 hour 30 mins
Making a Basic Website	1	1 hour	2 hours
Adding a Menu	1	30 minutes	1 hour
Adding a List	1	15 minutes	30 minutes
Total Course	36	47-49 hours	84.5 hours

3.3.1 sample timetables

Due to the amount of content developed and at the request of teachers (for guidelines and help in structuring the lessons), several sample timetables were developed which structured the time into the different sections. Most of these timetables do not give a lesson-by-lesson plan but recommend the order and structure of using the different modules. These were based on experiences gained teaching the course during the pilot study, as well as being sensitive to different learning styles, students engagement and interest, as well as a school's facilities. These timetables are presented in Table 3.2, Table 3.3, Table 3.4, Table 3.5, Table 3.6, Table 3.7 and Table 3.8. All of the times and amount of classes suggested in these timetables are approximations and are meant as a guide. Some classes will complete activities quicker than others, as will students within a class. Teachers are encouraged to be flexible, adaptable and encourage self-led learning, especially in the programming modules.

Table 3.2: Sample timetable for a class with 6-8 hours availability

Class 1	Introduction to Computer Science (including View of Computer Science Survey only)
Class 2	Introduction to Computational Thinking
Class 3	Algorithms 1 or 2 (could do both)
Class 4	Searching and Sorting
Class 5	Cryptography 1 or 2 (could do both)
Class 6	Scratch
Class 7	Scratch (final assessment - View of Computer Science survey)

In Table 3.2 the shortest (6-8 hours) recommended usage of the course is outlined. This covers some of the fundamental concepts within Computer Science and allows students to begin programming using Scratch. Due to the short nature of this timetable, the students wouldn't need to complete the Bebras problem solving test as it is unlikely that any significant change would occur over the duration of the course.

Table 3.3: Sample timetable for a class with 8 hours availability

Class 1	Introduction and assessments (this would include view of CS survey and the Bebras Problem solving test which takes 35 mins)
Class 2-7	CS Concepts - Recommendations to include: Intro to CS, Intro to CS, Algorithms, Cryptography.
Class 8	Recap & final assessments (same as the first week)

Table 3.3 presents a recommended timetable in the case where a teacher is able to commit 8 hours to the course. A short introduction to the course by using the "unplugged" lessons in the first module, Introduction and Concepts, commences this timetable. This could be used if access to PCs is limited but a teacher wants to introduce students to Computer Science as a subject. Unlike in Table 3.2 we recommend

that the students take the problem solving test if they are able to commit 8 hours to the course.

Table 3.4: Sample timetable for a class with 12 hours availability

Class 1	Introduction and assessments (this would include a view of CS survey and the Bebras Problem solving test which takes 35 mins)
Class 2-7	CS Concepts - Recommendations to include: Intro to CS, Intro to CS, Algorithms, Cryptography.
Class 8-11	Scratch – gives enough time to do the four exercises. Stronger students may finish them and can be encouraged to be creative and make something for themselves.
Class 12	Recap & final assessments (same as the first week)

The timetable presented in Table 3.4 is based on the initial Pilot Study conducted during the project research, and is recommended where a teacher can commit 12 hours to the course. It allows teachers to introduce students to key CS topics as well as begin programming with Scratch. Although Introduction and Concepts lessons are recommended in this and other timetables, teachers are given the flexibility to choose any lessons they wish. For example, if the class seems to particularly enjoy *Cryptography* then they could focus on that for a number of lessons.

In Table 3.5, the first timetable that includes a project is presented. This timetable is recommended where a teacher can commit to 16 hours for the course. One of the great things about CS and programming more specifically is that it allows students to be independent problem solvers and think creatively and imaginatively. Students could make small games, animations, stories or model real-world situations using Scratch. This allows them to use the skills acquired whilst learning (Classes 7-11) and start a project from the ground up.

We recommend, if possible, that classes take about 20 hours for this course. This allows students to get a good grounding in CS and they have the opportunity to devote a good amount of time to learning programming. Table 3.6 and 3.7 present two different schedules based on this time-frame. In Table 3.6, students get introduced to both Scratch and Python. This could especially be beneficial if students have previously used Scratch as they can remind themselves of key programming concepts (such as

Table 3.5: Sample timetable for a class with 16 hours availability

Class 1	Introduction and assessments (this would include a view of CS survey and the Bebras Problem solving test which takes 35 mins)
Class 2-7	CS Concepts - Recommendations to include: Intro to CS, Intro to CS, Algorithms, Cryptography.
Class 7-11	Scratch –this gives enough time to do the four exercises. Stronger students will finish them and can be encouraged to be creative and make something for themselves.
Class 12	A CS Concepts lesson – choose one
Class 13-15	Scratch project
Class 16	Recap & final assessments (same as the first week)

Table 3.6: Sample timetable for a class with 20 hours availability focussing on Concepts, Scratch & Python

Class 1	Introduction and assessments (this would include a view of CS survey and the Bebras Problem solving test which takes 35 mins)
Class 2-7	CS Concepts - Recommendations to include: Intro to CS, Intro to CS, Algorithms, Cryptography.
Class 7-11	Scratch –this gives enough time to do the four exercises. Stronger students will finish them and can be encouraged to be creative and make something for themselves.
Class 12	A CS Concepts lesson – choose one
Class 13-19	Python – this won't finish the Python course but will be a good introduction and stronger students might get most of the lessons done.
Class 20	Recap & final assessments (same as the first week).

loops) before moving onto Python and stretching their understanding with a textual programming language. Table 3.7 focuses on Scratch alone and would allow students to get a good grasp of the language and develop their own project with plenty of time to allow them to create something of substance. You could also substitute Python for Scratch on this timetable and focus on Python for the duration. Alternatively, you could mix the Concepts & Introduction with Scratch/Python and do 3-4 hours of each in a row. Some students will struggle with focusing on programming for weeks at a time and so often in these timetables we recommend taking breaks every now and then and doing an "unplugged" lesson before moving back into programming.

Table 3.7: Sample timetable for a class with 20 hours availability focussing on Concepts & Scratch

Class 1	Introduction and assessments (this would include a view of CS survey and the Bebras Problem solving test which takes 35 mins)
Class 2-10	CS Concepts - Recommendation: Intro to CS, Intro to CS, Algorithms, Cryptography as well as others.
Class 11-19	Scratch – This should be enough time to cover Scratch and let them get strong and creative at it with some sort of project.
Class 20	Recap & final assessments (same as the first week).

The final timetables presents a longer course timetable which takes up to 40 hours, and this could be extended if desired. Table 3.8 allows students to be introduced to key concepts through the unplugged lessons, to programming through both Scratch and Python as well as having a go at Web Development. 40 hours should give students the opportunity to do a Scratch and Python project as well as design their own website through the Web Development module. The order and structure of the timetable could be rearranged, with more time dedicated to different content as the teacher sees fit. As previously mentioned we encourage teachers to mix computer-based work with the "unplugged" lessons.

Table 3.8: Sample timetable for a class with 40 hours availability, covers Concepts, Scratch & Python, Web Development

Class 1	Introduction and assessments (this would include a view of CS survey and the Bebras Problem solving test which takes 35 mins)
Class 2-6	CS Concepts - Recommendation to include: Intro to CS, Intro to CS, Algorithms, Cryptography and one other.
Class 7-11	Scratch – gives enough time to do the four exercises. Stronger students will finish them and can be encouraged to be creative and make something for themselves.
Class 12	A CS Concepts lesson – choose one.
Class 13-16	Scratch project (see the Scratch project file for different schedules for the Scratch project)
Class 17	A CS Concepts lesson
Class 18-25	Python – this should be enough time to get through most lessons, especially if students work at home.
Class 26	A A CS Concepts lesson
Class 27-31	Python project (see the Scratch project file for different schedules for the Scratch project)
Class 33-39	Web Development
Class 40	Recap & final assessments (same as the first week)

3.4 TOPICS

From the literature review it was shown that there is a large range of tools and topics that can be used to teach Computational Thinking. Many of these are obvious starting points for a CS course such as programming and algorithms. It was determined from both the literature review, past experience in teaching as well as advice from experts in the field that a number of different modules be developed. The initial plan was for five separate but interconnected modules which would introduce students to a wide variety of CS areas. In the end only four were developed, however as discussed in Chapter 8 the hope is that this is just the first step in the development of this course.

In the following sections each of these modules is discussed in detail. Not all developed content is novel or unique to the course. Many other groups have developed great tools and lessons which teach the same topics and so these were used with the creator's permission and under creative commons license. Other activities or explanations are used widely in CS and so again cannot be attributed solely to this course.

3.4.1 COMPUTER SCIENCE CONCEPTS

This was the first module designed and is recommended as the starting point for teaching the course. From the literature review it was found that many CS topics can be taught without a PC, at least to an introductory level. Groups like CS Unplugged and Queen Mary University London design and develop engaging and hands-on activities which allow students to learn these CS topics often without realising they're doing CS. This can be an advantage when stereotypes about CS are commonplace. The "unplugged" or "computer-less" nature of these activities can also be appealing as some schools still suffer from a lack of equipment to allow whole classes to do lessons on programming or other computer-based topics. This allows teachers to be more flexible when planning out their schedules around booking computer rooms and other such practicalities.

Similar activities are also often used in the first year CS course in Maynooth University as an introduction to the labs. These allow students to engage with the topics without the "fear" of computers being in place and allow cooperation and team-building opportunities, as the activities are often done in pairs or small groups. The annual summer camp in the Computer Science department at Maynooth University employs similar methods when running sessions on Logic, Algorithms and Cryptography. Many of these activities were used or re-imagined designing this course.

The specific topics chosen were based on the results of the literature review, the most and central topics, and the experience of using them in the different settings described above. As stated in Chapter 8 the hope is to continually expand and develop these topics as well as introduce more as time goes by. A brief description of the topics and the activities are presented in this section. For more details, see Appendix C for the full lesson plans.

Introduction to Computer Science

Most students will have an idea of what CS is. This lesson is designed to correct what is usually a biased and stereotyped view of who can do CS, what it involves and why it matters. Activities include discussions on what CS is and what it is not, what problems CS could be helpful for and a task which allows students to develop a method to allow someone with Locked-In Syndrome to communicate.

Introduction to Computational Thinking

A lot of students will have never heard the term Computational Thinking before. As one of the main focusses of the course it was decided that a lesson to explain the fundamentals of this term were needed. As discussed in Chapter 1, this is easier said than done as many different definitions of CT exist. However, for the purposes of this lesson we took our guidance from the code.org curriculum who define it as Abstraction, Decomposition, Pattern Matching and Algorithm. This definition also fits into what was found in the previous research as described in Section 2.4. These skills are used in most if not all lessons as students are asked to solve problems and tasks, this lesson though explains the rationale behind CT as a problem-solving process. Activities include *code.org's* mathematical problem in which students use the four skills to make a seemingly difficult maths sum straightforward. Others are Mad Libs which are a fun way to explain the process of Abstraction and the Spit-Not-So game from Queen Mary University which challenges students to think outside the box for a solution to a game.

Algorithms

Algorithms are a key part of CS and are also often heavily linked with CT. Algorithms are often explained and practiced before students are taught programming. This can be done in several ways and the course provides two separate lessons which can be used either independently or one after the other. The first introduces the idea through everyday algorithms such as making a cup of tea or ordering a pizza. Students are introduced to the ideas of ambiguity and stepwise refinement through these. Ambiguity can be highlighted through getting students to write algorithms to play "rock, paper, scissors". This also allows students to begin testing and debugging solutions.

The second lesson involves two different versions of navigating a maze/obstacle course. The first version is undertaken on a square grid where the teacher knows the safe path/paths through the maze. Two teams race to cross the grid and again learn the art of testing their solutions. The second is an obstacle course which the students must write a detailed list of instructions which allows a "robot" to navigate through it. This can be used to explain how computers aren't smart and will do exactly as instructed, which means we must take care when programming them.

Cryptography

Cryptography is a topic that students will be more familiar with than they realise. Cryptography is regularly in the news and they will know the impacts that a lack of security and encryption can have online. Through four separate but related lessons students are introduced to classic ciphers, attacks on these ciphers as well as public key cryptography. Specific examples include the Caesar shift cipher, letter frequency attacks and a CS Unplugged activity which introduces students to one-way functions.

Searching and Sorting

One of the side focusses of these lessons is how data can be used, interpreted and manipulated. As a large part of programming is doing just that, it makes sense to include those topics in this section. Searching and sorting algorithms are vital tools to all computer scientists and a significant amount of time is given to these in undergraduate courses. This lesson introduces the idea of how and why we might sort data. Activities include a Bebras problem on sorting, allowing students to develop their own algorithms, the CS unplugged activity Battleships and showing different ways the same data can be sorted. An additional whole-class activity can be to "race" different sorting algorithms against each other. This can lead to a brief introduction about algorithm efficiency.

Finite State Machines

Finite State Machines (FSM) are, in essence, the make-up of all computers. They are simple systems that take in input and give output, they allow us to model what a computer is. This topic is expanded on more in other lessons and so the focus of this lesson shifts to regular expressions and constructing simple FSM. This allows students to understand how pattern matching works in programming languages as well as an introduction to inputs and outputs of different operations.

Data & Binary

The binary number system is hugely important to computers; it's how they operate. This lesson gets students to think first about the differences between data and information and gets them to think about how computers store data. Both topics are further discussed in other lessons and so the focus of this lesson is the binary number system. Through *Bakuro*[27] problems and converting numbers from binary to decimal and back students will come to understand how different number systems can work.

Programming Concepts

Although programming is covered in two separate modules described later, it was felt that an unplugged introduction to the basic concepts might be of some use. Students can often be quite overwhelmed when learning to programme and so concepts such as variables, loops and conditionals can be introduced without the need to learn keywords or syntax. The topic is split into two lessons. The first of these introduces variables and user input. Variables are taught through the common box analogy as described in a Teaching Computing London demonstration along with their variable dry run activity. Input is discussed as an important element of all programs with examples of this and other essential parts of programs discussed. Loops are introduced through examples such as running tracks and through designing Lego^{TM} mazes. Conditionals are taught through a decision tree exercise.

All these topics are then brought together in an activity where students design their own game. The idea is to use common game devices and their own imagination to create a game which uses Variables, Input, Loops and Conditionals. The rules are devised and items such as playing cards, dice, counters etc. are used to allow students to play their own games.

3.4.2 scratch

The Scratch lessons were designed to not only introduce students to programming concepts such as loops and variables, but also introduce them incrementally to all the functionality in Scratch 2.0. The tutorials are self-led but can also be demonstrated by teachers if this suits their class. The first tutorial is designed as a basic introduction

27 `https://teachinglondoncomputing.org/bakuro/`

and should be completed first. The following three tutorials can be done in any order. The order they are presented below is our recommendation. The lessons were designed to build on the skills learned in the previous ones and were aimed to progress in difficulty and complexity level. Cat and Mouse is the easiest with Guessing Game being the hardest and most complex. These subsequent tutorials have a few tiered tutorials. Depending on the confidence and level of the students, teachers can give students a step-by-step guide, or a more scaled back tutorial with requirements and hints. These can all be found in AppendixD.

Cat and Mouse

The aim of this exercise is to develop a two-player game where one player uses the mouse and the other uses the arrow keys. The sprites (the name for actors in Scratch) will move around the screen either following the mouse or at the command of the arrow keys and the aim is for one to chase the other, a cat and a mouse. Programming concepts introduced include user input, loops and event handling. Scratch tools used include keyboard and mouse input, importing pictures, event handling such as the broadcast and receive blocks and the repeat until block. A fully worked version of the game can be found here: `https://scratch.mit.edu/projects/153850167/`.

Polygon Drawer

The aim of this exercise is to draw different shapes based on user input. This exercise is repeated in the Python section and so is a good way to show the link between programming languages. Programming concepts introduced include nested loops, user input and variables. Scratch tools used include keyboard input, random numbers and mathematical operators, variables and the pen blocks. A fully worked version of the project can be found here: `https://scratch.mit.edu/projects/153784869/`.

Pong

The aim of this exercise is to create a working version of the arcade game Pong[28]. Programming concepts introduced include conditionals, infinite loops and defining functions. Scratch tools used include "if then", drawing the stage and sprites, "forever" and making our own block. A fully worked version of the game can be found here: `https://scratch.mit.edu/projects/153784869/`.

28 `https://en.wikipedia.org/wiki/Pong`

Guessing game

The aim of this exercise is to create a guessing game where the user tries to guess a randomly selected number by the computer. This links in well to the Searching & Sorting lesson and can be done through Python as well. This introduces students to using multiple conditionals and Boolean operators. Scratch tools used include multiple broadcasts and receives, "if then, else" and "or" and "and" operators. A fully worked version of the project can be found here: `https://scratch.mit.edu/projects/142567858/`.

The project

The best way for students to learn how to program in any language is to give it a go themselves. After following through the tutorials, students should hopefully have a grasp of the basic functions of Scratch and an idea of what can be made using it. Now the hope is that using this basic knowledge they can create something from scratch, using Scratch. The content of the project is completely open, they can make a game, a story, an animation, a quiz or whatever they want. This project, and other projects in the course, teach students about software design and the creation process. To this end the project will follow a structure in terms of planning, design, implementation and conclusion. Some suggestions are optional, and teachers can use them if it suits their class, others are more vital and so are recommended for all classes. Below is an outline of the project structure:

1. *Research and creative stimulation:*

   Direct students to the Scratch community and have them look through and search for ideas. A few sample projects will be provided on the CS2Go website. They can search specifically for their thoughts (for example Platform game, Short Movie) or just look around. If they find anything that particularly has a nice feature they want to use, they should make a note of it.

2. *Sketch/write down the idea informally:*

   Once students have some ideas, they need to write down their ideas or draw out their plans. These don't have to be masterpieces, or even close to the end product, but it's helpful for concretising the idea. Teachers should collect these ideas and look through them; some students will be extremely ambitious, and the teacher might want to tell them to focus on a part to begin with. However, they should be encouraged to keep working outside of the class e.g. assign homework. Other students will have picked something that might be too simple.

This part all depends on how long teachers are giving students to do the project and how much is expected of them to do at home.

3. *Produce an algorithm:*

   This is a more detailed version of the previous step. Once students' ideas have been finalised they should write down what Sprites they'll need, what each Sprite will need to do, what they need the stage to do, etc. They don't have to write down the exact blocks, just the general gist of what's going to happen.

4. *Creation:*

   Now that the students have gone through the design process they get to make their creations. The Internet and wider Scratch community will be hugely helpful for students who want to make/use a specific function, so encourage students to search on the Scratch wiki and the wider web. Teachers should encourage cooperation, as with all parts of this course, if they're stuck ask a friend who might know. Teachers should make sure students make a note of changes to their original designs, problems faced and the solution to those problems. This will help especially if it is decided to do either of Step 5 or 6.

5. *Presentation (optional):*

   A good way to test students' understanding of Scratch and the deeper programming concepts is to get them to talk through what they've made. This could be done one-on-one with a teacher or as whole-class presentations, which could be a good experience for students.

6. *Reflection (optional):*

   Another good way to test students' understanding and to make them aware of their own learning is to get them to write a reflective piece of writing. In this students can discuss what they learned, problems they encountered and how they fixed them, what they would do if they started over etc.

Sample schedules

We will now present three sample schedules for the projects. These schedules are guidelines on how teachers might like to run the projects. Teachers are free to change these as needed. It is important they give the students enough time to both come up with an idea as well as then implementing it, otherwise it can be quite frustrating for them. One thing to note is that these schedules do not include potential homework, although students should be encouraged to work on their projects at home. The algorithm and reflection piece especially could be good for homework but can equally be done in class. The length of time teachers plan to give them, as well as how much work is expected at home, should be a factor into the quality of projects expected. This is important when giving feedback on the ideas, teachers don't want students to go too big or too small. They should be encouraged to experiment and try out different ideas. This type of learning will improve not only their programming skills but also their CT and problem-solving skills.

The longest of the three schedules involves all of the above mentioned sections of the project. It assumes either one 1-hour class or 2 35/40-minute classes and no homework. This should allow students to develop and design a complete enough project that they can use all of the knowledge gained from the tutorials. The schedule is as follows:

- Week 1: Introduce project, show ideas, research and creative stimulation
- Week 2: Finish off research, sketch/write down the idea informally
- Week 3: Give feedback on the idea, produce an algorithm, begin programming project
- Week 4: Potentially give feedback on the algorithm, Programming project
- Week 5: Programming project
- Week 6: Programming project
- Week 7: Complete programming project, begin work on presentation/reflection piece
- Week 8: Work on presentation/reflection piece, submit reflection piece
- Optional: Week 9: Give presentation

The middle of the three schedules includes a subset of the described sections for the project. The setup and assessment/reflection is shortened compared to the longer project. It assumes either one 1-hour class or 2 35/40-minute classes and no homework. It is the recommended for most classes. The schedule is as follows:

- Week 1: Introduce project, show ideas, research and creative stimulation, sketch/write down the idea informally
- Week 2: Give feedback on idea, produce an algorithm, begin programming project
- Week 3: Programming project
- Week 4: Programming project

- Week 5: Complete programming project, begin work on presentation/reflection piece
- Optional: Week 6: Submit reflection piece, Give presentation

The shortest project allows students to make a more basic project which will allow them to personalise the knowledge that they have gained from the tutorials. The schedule is as follows:

- Week 1: Introduce project, show ideas, research and creative stimulation, sketch/write down the idea informally
- Week 2: Give feedback on the idea and algorithm, begin programming project
- Week 3: Programming project
- Week 4: Programming project
- Optional: Week 5: Complete project, work on reflection piece, submit reflection piece

3.4.3 PYTHON

One of the main reasons, besides those described in Section 3.2.2.2, that Python was chosen to be included in CS2Go was the PACT team's experience using Python in the past with schools. It was well received but it was felt an upgrade was needed to the content, namely the presentation and layout. More colourful, engaging and easy to read slide decks and exercise sheets were created. With a different target audience, TY students, some of the lessons also needed to be streamlined to suit secondary school class times and homework expectations. This was done with support from a postgraduate student who was restructuring a CS001 course, which is for mature students thinking of returning to full-time education. The course was built around Python Turtle, a graphical language built on Python which is based on the famous Logo programming language. We felt that this would be more engaging, especially for TY students, as the output would be very visual, rather than the usual numbers and sentences which are the standard output for novice programmers. The module was broken up into lessons and labs. The idea was that teachers would guide students through the lessons using the provided slides and code examples, alongside live coding if they were comfortable doing that. Once the idea, concepts or structures were introduced (for example loops) the students would then complete several exercises which built upon the previously covered content and added this new concept/idea into the problems.

These lessons are listed now in the suggested order of delivery e.g. teachers should begin with the Introduction lesson followed by Variables, Expressions and Statements, this leads into students completing Lab 1. The content is incremental as any programming language is; without the foundations more complex concepts will be hard to understand and implement. Whilst teaching one group of students Lesson 5, Turtle Graphics, was used as the introductory lesson. This was due to a lack of time and the desire to engage the students from the start with the graphical nature of Python Turtle so there is scope for that lesson to be the starting point. The final three lessons (6-8) require the knowledge from lessons 1-4 to be useful for students. Lessons 6-8 could also be taught in a different order if it was felt necessary. The lessons and other resoures, including slides, can be found in Appendix E.

Introduction

This lesson gives students a very high-level and brief overview of what Python is, how the computer runs the code, the different ways of running the code and helps them to write their first program, "Hello World". This lesson would also include an introduction to whatever platform teachers are using to run the Python code. The recommended platform is `repl.it` (see Section 3.2.2.2), however any text editor or IDE (Integrated Development Environment) that teachers are comfortable with can be used.

Variables, Expressions and Statements

This lesson introduces the students to the idea of a variable and assignment in Python. It then introduces printing variables and comments and variable types. Variable types are dynamic in Python and so the concept doesn't need to be covered in detail but can be expanded on if the teacher wishes.

Lab 1

After completing both the Introduction and Variables lessons, the students will work through the Lab 1 assignment sheet. This includes five questions which includes printing variables, computing the average of two numbers and calculating the area of a circle.

Strings

This lesson recaps the content covered in the previous two lessons before looking more in depth at strings. String operators, printing strings and a selection of string functions are shown to the students.

Keyboard Input

This lesson introduces students to Python's built-in input function, which allows keyboard input from the user. Students will be shown how to save this input as a variable and print it. The lesson also shows how to convert a string into an integer, this isn't explained fully but could be if the teacher felt it was helpful or interesting to their students.

Lab 2

This assignment sheet has six questions which include printing variables, using string functions and using user input.

Turtle Graphics

This lesson introduces students to the Turtle Graphics library designed for Python. They will learn about importing modules, creating an object, various Turtle Graphics functions and calling those functions on an object. Most of these topics aren't covered in detail, just their usage and a brief overview of what is happening will allow students to move on. However, if teachers wish to go more in depth then that is possible.

Lab 3

This assignment sheet contains seven questions related to Turtle Graphics. Questions include drawing shapes, changing the screen and pen colour and making screen objects.

Loops

This lesson introduces students to the fundamental programming concept of Loops. It introduces students to the Python syntax of a "for" loop as well as giving examples of why loops are useful. The main examples involve drawing X-sided Polygons. Students can also be introduced to the idea of an infinite loop; this also allows students to be introduced to "while" loops.

Lab 4

This assignment sheet contains six questions related to Loops. These questions all involve drawing shapes or different images using Turtle Graphics.

Selection Statements

This lesson introduces students to the concept of Selection (if, else, else if) statements. It introduces students to the Python syntax as well as explaining why if statements are useful and how they can be used. It also introduces students to comparison (=, <, > etc.) and logical (and, or, not) operators.

Lab 5

This assignment sheet contains six questions on Selection statements. Questions include drawing different shapes depending on a user inputted number, checking to see if someone is allowed in a party and seeing if a user's name is long or short.

Functions

This lesson introduces students to writing their own functions. They should by now be very familiar with functions (Turtle Graphics, Strings etc.) and so this lesson focuses mainly on defining their own. It introduces the syntax (def name()) and gives an example with drawing shapes.

Lab 6

This assignment sheet contains five questions where students must write their own functions. Questions includes drawing shapes, calculating the average of three numbers and figuring out the number of days in a month.

Python Project

The Python project is similar in layout and structure to the Scratch project. This will allow students who do both to see how the design process can be applicable to programming no matter what language or tool is used. After going through the lessons and exercises, students should hopefully have a grasp of the basic functions of Python Turtle and an idea of what sort of programs they can make. Using this basic knowledge, they can create something from scratch using Python/Python Turtle. The content of the project is completely open, they can make a game, a story, an animation, a quiz etc.

This project will teach the students about software design and the creation process. To this end the project will follow a structure in terms of planning, design, implementation and conclusion. Some ideas will be optional, and teachers can use them if they think it suits their class, others are more vital and so are recommended. Below is an outline of the project structure:

1. *Research and creative stimulation:*

   Students should first be encouraged to search the internet and the site `https://docs.python.org/2/library/turtle.html` and search for ideas. Several example projects will be linked on the CS2Go web portal. Students can search specifically for their thoughts (for example a game, animated drawing) or look for code that does specific functions. If they find anything that particularly has a nice feature they want to use, make sure they make a note of it.

2. *Sketch/write down the idea informally:*

   Now that students have some ideas, allow them to write down their ideas or draw out their plans. These don't have to be masterpieces, or even close to what it ends up being, but it's helpful for concretising the idea. Teachers should collect these ideas and look through them. Some students will be extremely ambitious, and teachers might want to tell them to focus on a part to begin with. However, they should be encourage to keep working after the class is finished e.g. assign homework. Others will have picked something that might be too simple. This part all depends on how long a teacher gives them to do the project and how much you expect them to do at home.

3. *Produce an algorithm:*

   This is a more detailed version of the previous step. Once their idea has been finalised students should write down what programming constructs they'll need (such as loops and functions), what each will need to do, what they need the background and turtle to look like etc. They don't have to write down the exact code, just the general gist of what's going to happen.

4. *Creation:*

   Now that the students have gone through the design process, they get to make their creations. The Internet will be hugely helpful for students who want to make/use a specific function, so encourage them to search through the Python turtle documentation (`https://docs.python.org/2/library/turtle.html`) and the wider web. Teachers should encourage cooperation as with all parts of this course. If they're stuck ask a friend who might know. Teachers should make sure students make a note of changes to their original designs, problems faced and the solution to those problems. This will help especially if it is decided to do either of Step 5 or 6.

5. *Presentation (optional):*

   A good way to test students understanding of Python and the deeper programming concepts is to get them to talk through what they've made. This could be done one-on-one with the teacher or as whole-class presentations, which could be a good experience for them.

6. *Reflection (optional):*

   Another good way to test their understanding and to make students aware of their own learning is to get them to write a reflective piece of writing. In this they could discuss what they learned, problems they encountered and how they fixed them, what they would do if they started over etc.

We will present possible schedules for the delivery of the projects. These schedules are guidelines on how teachers might like to run the projects. Teachers are free to change these as needed. It is important they give the students enough time to both come up with an idea as well as then implementing it, otherwise it can be quite frustrating for them. One thing to note is that these schedules do not include potential homework, although students should be encouraged to work on their projects at home. The algorithm and reflection piece especially could be good for homework but can equally be done in class. The length of time teachers plan to give them, as well as how much work is expected at home, should be a factor into the quality of projects expected. This is important when giving feedback on the ideas, we don't want students to go to big or too small, however do let them experiment and try, that's the beauty of programming.

The longest of the three suggested schedules involves all of the above mentioned sections of the project. It assumes either one 1-hour class or 2 35/40-minute classes and no homework. This should allow students to develop and design a complete enough project that they can use all of the knowledge gained from the tutorials. The schedule is as follows:

- Week 1: Introduce project, show ideas, research and creative stimulation
- Week 2: Finish off research, sketch/write down the idea informally
- Week 3: Give feedback on the idea, produce an algorithm, begin programming project
- Week 4: Potentially give feedback on the algorithm, Programming project

- Week 5: Programming project
- Week 6: Programming project
- Week 7: Complete programming project, begin work on presentation/reflection piece
- Week 8: Work on presentation/reflection piece, submit reflection piece
- Optional: Week 9: Give presentation

The middle of the three schedules includes a subset of the described sections for the project. The setup and assessment/reflection is shortened compared to the longer project. It assumes either one 1-hour class or 2 35/40-minute classes and no homework. It is the recommended for most classes. The schedule is as follows:

- Week 1: Introduce project, show ideas, research and creative stimulation, sketch/write down the idea informally
- Week 2: Give feedback on idea, produce an algorithm, begin programming project
- Week 3: Programming project
- Week 4: Programming project
- Week 5: Complete programming project, begin work on presentation/reflection piece
- Optional: Week 6: Submit reflection piece, Give presentation

The shortest project allows students to make a more basic project which will allow them to personalise the knowledge that they have gained from the tutorials. The schedule is as follows:

- Week 1: Introduce project, show ideas, research and creative stimulation, sketch/write down the idea informally
- Week 2: Give feedback on the idea and algorithm, begin programming project
- Week 3: Programming project
- Week 4: Programming project
- Optional: Week 5: Complete project, work on reflection piece, submit reflection piece

3.4.4 WEB DEVELOPMENT

Web Development is a vast topic which can include designing web systems, learning about design processes or even digital citizenship and how data protection works. In this module we currently focus on students designing their own websites. Through this students can get the satisfaction of seeing their designs published on the web and see the results of their code instantly. This section of the course is the least developed and the main area where future expansion and enhancement could be done. One reason for this is the wealth of existing tutorials and user-friendly guides to website development. Examples include Tutorials Point[29], W3 schools[30] and codecademy[31]. As with other activities, there is no sense in remaking or trying to imitate an already existing, well-established and successful format. W3 schools is a fantastic repository of help, guides and information on all things related to website development. The hope is in the future to develop more unplugged lessons on topics such as the Internet, Digital Citizenship and Networks. The lessons described in this section can be found in full in Appendix F.

Making a basic website

This is the first of a few tutorials designed to allow students to learn the very basics of HTML and CSS. Students will be introduced to Mozilla Thimble and be shown how to use the many features of the platform. They will be introduced to the idea of tags as well as many examples of these. Students will be introduced to two types of styling, how to add images, how to embed a video and how to publish their sites live.

Adding a list

This is a brief tutorial on how to make and style a list using HTML and CSS. The example used is that of a colourful diary.

Adding a menu

This lesson gives some theory behind website addresses and introduces students to HTTP. It then shows students how to add links using HTML and then how to put these into a list to create a menu. It also shows students how to style these lists to look more like a "standard" menu.

29 `https://www.tutorialspoint.com`
30 `https://www.w3schools.com/`
31 `https://www.codecademy.com/`

Databases

This is an unplugged lesson which introduces students to the concept of a database. This is done through making a class database with information on things like name, hair colour, age, favourite book, etc. Students will then be introduced to a uniquely designed Database-query language which is based on SQL (Structured Query Language). Students can then query their classes database and learn how to write statements and display results.

3.5 CONCLUSION

Overall, 80 hours of content have been developed for CS2Go. The hope is that this is just the beginning and that further content will be developed in the future. Specifically this will include: expansion of the Web Development module including more unplugged lessons and web development tutorials, the creation of an App Development module, further Scratch tutorials and more advanced Python programming lessons. The content created will also be altered and expanded based on further research, input from teachers and students as well as any other developments in teaching methods and content.

4 ASSESSMENT

Assessment is one of the key factors when designing and developing courses within any level of education. For the assessment of this course, several surveys and feedback forms were created along with a problem-solving test. This chapter will describe the development and content of these various assessments. They will all be referred to in future chapters when discussing the feedback and results from the various studies developed.

4.1 GENERAL ASSESSMENT FORMS

One of the main goals of this project was to develop lessons that would be interesting to students, teach them about CS and be easily usable for teachers with little to no CS background. It is clear that the assessment of whether this was achieved or not should not be done just by the developers or others who work in the areas of CS or Education. The main audience of CS2Go is secondary school teachers and students. To see whether the developed lessons were beneficial, engaging and easily usable a series of feedback forms were designed. These were used to collect relevant data from those who taught or were taught the lessons during the duration of this project. In this section, these forms and surveys are described.

4.1.1 PERSONAL SURVEY

In the first week of the course, students complete a personal survey to obtain demographic data and other personal data such as age, gender, and previous programming/Computer Science experience. As we developed the course, we were interested to see how students from different backgrounds and demographics might engage with the lessons. To do this, we needed to collect the information on a range of topics anonymously. To do this, a personal survey was developed and teachers assigned students random identifiers (such as DGS01) which students then used to complete all of the feedback described in this section. The full personal survey can be found in Appendix G.

4.1.2 LESSON FEEDBACK

One of the main goals in this early stage of course development was to see whether students enjoyed the lessons and whether they felt they learned something from the class. Upon completion of each lesson or class, students filled out an anonymous feedback form. All of the forms had questions which asked about whether they had enjoyed the lesson/class, what they had enjoyed/not enjoyed and what they had learned. In addition, individual lessons had unique multiple-choice questions based on the topic covered. These differed for each form and some had no multiple-choice questions. These questions can be seen in Chapter 6 when the results of the year-long school study are discussed. A sample survey of the generic questions can be found in Appendix H.

An anonymous end-of course survey with similar types of questions to those shown in Appendix H was also prepared and administered after the pilot study as described in Chapter 5. This survey wasn't used during the year-long study as it wasn't needed as the individual class feedback being the focus.

4.1.3 EXAM QUESTIONS

During my teaching in the main year of study (2017-18), as described in Chapter 6, the Mathematics teacher of the school requested some questions to supplement the students' Christmas exams. These are presented below with sample answers that were supplied to the teacher. Although these were developed for a specific purpose, it shows the potential for a more standard exam based on the content of CS2Go.

Q1. In your own words, describe what Computer Science is (2-3 lines).

Sample Answers:

- *Computer Science is concerned with the study of everything to do with computers and our relationship with them.*
- *Solving real world problems using the power of computing technology.*
- *Anything like programming, coding, website design, app design etc. you could award half marks as it's not wrong but only a part of the whole. The above definition is what they were given in class and is available to them in the slides.*

Q2. What are the four parts that make up Computational Thinking? If you can't think of them describe them.

Sample Answer: 1) Abstraction - Pulling out specific differences to make one solution work for multiple problems 2) Decomposition - Answer: breaking a problem up into smaller parts 3) Pattern matching - looking for similarities between problems/solutions 4) Algorithm – Set/sequence/list of instructions/commands/directions of how to do something/how to solve a problem.

Q3. What is an algorithm?

Sample Answers: Set/sequence/list of instructions/commands/directions of how to do something/how to solve a problem.

Q4. Write out an algorithm for how to make a cup of tea, make sure to include all relevant steps.

Sample Answer: Many correct answers for this. One is below, so long as you could make a cup of tea from the instructions without too many assumptions that's fine.

- *Find the kettle*
- *Fill the kettle up to the fill line*
- *Place it on the holder*
- *Flick the switch to start boiling the water*
- *Whilst the water is boiling, find the mug cupboard*
- *Choose a mug that you like and is big enough for your drink*
- *Place the mug on the side*
- *Now find the cupboard that contains the tea Choose your favourite type of tea, it could be Barry's, Herbal, Earl Grey etc.*
- *Place the teabag into the mug*
- *When the kettle has boiled, add water 3/4 of the way up the mug.*
- *Get the milk and sugar from the fridge and cupboard respectively.*
- *Add the milk and then add the sugar, stir in using a spoon.*
- *Remove the teabag when it is as strong as you wan i to be.*

Q5. The following text has been encrypted using a Caesar Shift Cipher, decrypt it.
Hint: Note the individual letter
Ymnx nx f xjhwjy rjxxflj, bjqq itsj ktw knsinsl ny!
Sample Answer: This is a secret message, well done for finding it! (the key is 5)

4.1.4 TEACHER FEEDBACK

Along with the student feedback it was vital to get information from teachers about how usable the lessons were, whether they needed more help or resources and whether they felt the students learned and engaged with the topic. This information could then be used to improve the course and develop more content. Teachers were given digital forms to fill out after teaching lessons to their class. An example of a form can be found in Appendix I as well as at `https://goo.gl/forms/HkdRSDYIEc9Xkxxj2`.

4.2 VIEW OF COMPUTER SCIENCE SURVEY

One of the outcomes we hoped would occur after students were taught with lessons developed for CS2Go was a change in students' opinions of CS. Many students wouldn't have been taught much, if any, CS prior to Transition Year and so it was a perfect opportunity to survey a large number of students' opinions. To do this, a survey was designed based on a survey developed by Taub, Armoni, and Ben-Ari [89]. It was administered before and after the course and was designed to better understand and evaluate students' views of what Computer Science is, what it involves, and who a computer scientist is. The same survey was used for both of the schools study as discussed in Chapter 5 and Chapter 6 and can be found in Appendix J. The same questions were used with the first-year undergraduate study discussed in Chapter 7 with the exception of Q1; this was removed as all of the students were currently studying CS.

4.3 COMPUTATIONAL THINKING ASSESSMENT

4.3.1 GOALS OF THE ASSESSMENT

One of the areas that had to be analysed to judge the success and impact of CS2Go was whether it improved students CT skills. To make this possible there was a clear need to find or develop a Computational Thinking assessment. It had to fit a number of requirements, namely:

- Be applicable to the target age range (15-17 years old)
- Allow for differentiation between strong and weaker students (i.e. have harder and easier questions)

- The test needed to allow students to complete the questions without any prior knowledge
- Be completed within a 40-minute class time
- Allow for a pre- and post-test of similar difficulty and content
- Test students' Computational Thinking skills

4.3.2 CT ASSESSMENT

Assessment of CT is in it's infancy and as such, there aren't many methods for educators to test it. As it is seen more and more as a central skill for students to have, the lack of assessment options can cause difficulties for educators. However, some work has been done in the area.

Of note is an effort to develop a Computational Thinking test called the Computational Thinking Test (CTt) and another called Dr. Scratch. Dr Scratch analyses Scratch projects to deliver a CT score based on a number of different metrics [67]. This is a great tool and we recommend it as a tool to analyse Scratch projects developed in one module of CS2Go. As it works exclusively with Scratch, this didn't suit our purposes to study students "general" CT skills pre-course and post-course. The CTt has been developed as a series of multiple-choice questions that are presented online in either a "maze" or "canvas" interface. There a number of factors which define the questions [32]. The group have analysed these two metrics (CTt and Dr Scratch) alongside the Bebras problems [82]. They found that the CTt was partially convergent with the other two assessments. They claim this is to be expected as the three assess CT but from different perspectives. One strength they believe that the CTt has is that it can be done in "pure pre-test conditions". This can allow early detection of problems but also doesn't allow for contextualised assessment. This is a strength of the Bebras problems, which have "real-life" questions but they also claim the "psychometric properties of some of the problems are skill far off being demonstrated".

With this being said, we felt that, from assessing various forms of assessment for CT that exist, both through the systematic literature review described in Chapter 2 and through interactions with other researchers and educators that the Bebras problems would provide a good basis for a CT assessment.

4.3.3 BEBRAS PROBLEMS

Bebras is an international competition which aims to promote CS and CT among school students at all ages. Participants are usually supervised by teachers and the challenge is performed at schools using computers or mobile devices.

As part of their work in schools, the PACT group are involved in the Irish version of this test and have designed and used Bebras problems in order to provide teachers with resources to introduce students to CT. They are designed to be 3-minute-long questions and require no prior knowledge of programming or CS topics. All the problems are linked to topics in computing such as Cryptography, Trees etc. and this allows them to be used to introduce students to these topics without students even realising they are learning them.

The fact that the Bebras problems are designed to test CT skills means they are well suited to assess students CT skills before and after the course. Gouws, Bradshaw, and Wentworth [33] previously used the South African version in a similar manner and it was this that inspired the development of our own CT assessment. Other studies have also been carried out on the Bebras problems to investigate both their effectiveness and to compare them to other Computational Thinking tests [24, 42, 43, 91]. These studies have shown that Bebras problems can be an effective tool in promoting CT and problem-solving skills [24]. Work has also begun in comparing them to other worldwide tests such as PISA [42] as well as analysing the underlying psychometric structure of the problems [43]. The results are promising but are in the early stages and more work is required in this area.

4.3.4 METHODOLOGY

The current format of the Bebras challenge doesn't suit as a comparative test as the questions change each year. The challenge is often conducted on PCs and we wanted to allow teachers to do it through both pen and paper or online if desired. It was decided that 13 questions would be used in each test, with students allowed 35 minutes to complete them. This considers both the 3-minute design of the question as well as the fact that some of the questions are designed for a younger age group than the target demographic. It was hoped that each test would be as close as possible to each other in terms of difficulty level as well as question topic and type. To do this, many questions from Bebras challenges across the world were examined and critiqued.

The questions used in the UK challenges were deemed most appropriate and the contents of the test were sources from the 2015 and 2016 challenges. For the target age group (15-17-year olds) the UK challenge involves 18 multiple-choice questions over 40 minutes. As explained above this was adjusted slightly for our purposes to be shorter but also allowed for some non-multiple-choice questions as well. The first criteria for the tests was to ensure that they were as close in terms of difficulty level as possible. The UK Bebras challenge is broken into six age groups as presented in Table 4.1.

Table 4.1: Bebras UK Sections

Name of Group	Year Group (England & Wales)	Approx. age group
Kits	2 &3	6-8
Castors	4 & 5	8-10
Juniors	6 & 7	10-12
Intermediate	8 & 9	12-14
Seniors	10 & 11	14-16
Elites	12 & 13	16-18

Each age group is then further divided into three Sections, namely, Section A, Section B and Section C. Questions in Section A are considered the easiest with Section C problems being the more complex. Questions that are submitted for the Bebras problem are reviewed by a panel of experts in Computing education who are involved in the Bebras challenge. Questions that are accepted for either the qualification rounds, or the final challenge are often used in multiple age groups and across the three Sections.

To ensure that each created test was as similar in difficulty as possible, these ratings were used to select questions for each test, ensuring that corresponding problems were used in at least one common section and age group. The chosen corresponding problems, which can be found in full in Appendix K and L, for the tests along with the sections they have in common can be seen in Table 4.2.

The second criteria for the assessments was to have similar topics and styles for the questions where possible, and to have these topics relating to areas covered in the course. This was not as much a priority as the difficulty, so questions were considered

Table 4.2: Matching sections of the two created tests

Test 1	Sections in common	Test 2
Bracelet	Kits B, Castors A	Bebras Painting
Animation	Castors B, Juniors A	Bottles
Animal Competition	Castors B, Intermediate A	Party guests
Cross Country	Intermediate A	Tube System
Stack computer	Senior B, Elite A	Pirate Hunters
Throw the dice	Juniors C	Magic potion
Drawing stars	Intermediate B	Concurrent directions
Beaver lunch	Senior B	Theatre
You won't find it	Intermediate C, Elite A	Secret messages
Bowl Factory	Intermediate C, Elite B	Triangles
Fireworks	Senior C	Scanner code
Kangaroo	Elite C	The Game
Spies	Elite C	B-enigma

even if this wasn't possible. Table 4.3 presents the topics covered by each question for each test.

Prior to either of the tests being used, they were tested by a small group to ensure that the questions were clear, made sense, that our timing (35 minutes) was reasonable, and that both sets of questions appeared similar in terms of difficulty. The group found that the second test was perhaps slightly harder, but that for 35 minutes it was doable and that the questions were clear in general.

To further assess if the two tests were similar in difficulty and to validate their effectiveness the questions were sent out to teachers, students and academic staff with instructions of how to rate the difficulty of the questions. The hope was that this sample of different demographic and career groups would show not only that the two tests are similar in difficulty but allow us to weigh either specific questions or one of the tests accordingly if there was a discrepancy. Table 4.4, Table 4.5 and Table 4.6 presents the qualifications and areas of work of the participants who reviewed these tests. There was a mixture of genders and ages but this data was not collected.

Table 4.3: Topics of the questions for each test

Test 1	Topic	Test 2	Topic
Bracelet	Pattern Matching	Bebras Painting	Algorithms
Animation	Attributes & Variables	Bottles	Sorting
Animal Competition	Data ordering	Party guests	Graphs
Stack computer	Stacks	Pirate Hunters	Graphs
Throw the dice	If-then-else	Magic potion	Logic & binary
Drawing stars	Objects	Concurrent directions	Parallel instructions
Beaver lunch	Trees	Theatre	Sequences
You won't find it	Ciphering	Secret messages	Ciphering
Bowl Factory	Sorting	Triangles	Iterative, pattern matching
Fireworks	Encoding	Scanner code	Pixels
Spies	Gossip Problem	B-enigma	Encrypting

We asked people to rank the questions on two scales. Twenty people completed it for Test 1, with 18 of those also completing it for Test 2. The first scale was rating the questions in each tests from easiest to hardest, this gave each question a ranking from 1 to 13. It was felt a second scale was needed as some questions might be classified as being the "easiest" two, but there could be a big gap in difficulty between them. The same could be true of any two questions within the ratings scale. Since each test had 13 questions it was decided that a scale from 1-10 wouldn't allow respondents to be clear and would in fact limit the ranking. A scale of 1-20 was decided upon, with 1 being easiest and 20 being hardest. Participants weren't given further instruction unless it was requested, they were free to rank the questions as they saw fit. The form provided to these reviewers can be found in Appendix M.

Table 4.4: Qualification profile of our participants

Highest Qualification	No. of participants
PhD	5
Masters	1
Bachelors Degree	10
Leaving Certificate	3
Unreported	1

Table 4.5: Job profile of our participants

Job Title	No. of participants
Lecturer	5
Primary school teacher	2
Secondary school teacher	1
Tutor/Postgraduate Student	5
Youth worker	2
Nurse/Veterinary Nurse	2
Undergraduate Student	2
Unreported	1

4.3.5 RESULTS

Analysis of Bebras Test 1

When ranking the questions from hardest to easiest (1-13) Table 4.7 presents obtained scores, with the questions being ranked from smallest (easiest) to largest (hardest).

When asked to rate the questions on a scale from 1-20, Table 4.8 presents the scores for each question with the questions presented from smallest (easiest) to largest (hardest).

Table 4.6: Area of work of our participants

Area of work	No. of participants
Computer Science	9
Irish	1
Mathematics	1
Electronic Engineering	1
Youth work	2
Medicine	2
Teaching	3
Unreported	1

Table 4.7: Test 1 Question Rankings out of 13

Rank	Question	Ranking out of 13
1	Bracelet	1.75
2	Animation	4.8
3	Cross Country	4.95
4	Throw the Dice	5.7
5	Drawing Stars	6.3
6	Beaver Lunch	6.65
7	You Won't Find It	7.15
8	Kangaroo	7.45
9	Animal Competition	7.6
10	Fireworks	7.95
11	Stack Computer	9.6
12	Bowl Factory	9.85
13	Spies	11.25

Table 4.8: Test 1 Question Rankings out of 20

Rank	Question	Score out of 20
1	Bracelet	3.2
2	Animation	5.6
3	Cross Country	6.15
4	Beaver Lunch	7.6
5	Drawing Stars	7.6
6	Throw the Dice	7.7
7	You Won't Find It	8.75
8	Animal Competition	8.95
9	Kangaroo	9.05
10	Fireworks	9.45
11	Bowl Factory	12.6
12	Stack Computer	13.5
13	Spies	15.1
	Average	8.87

Interestingly, there is not much of a difference between these two rankings in terms of easiest to hardest ranked questions. This is to be expected, and the two rankings are presented in Table 4.9 to show this comparison.

It should be noted that *Beaver Lunch*, *Drawing Stars* and *Throw the Dice* were rated as almost exactly the same level of difficulty, with scores of 7.6, 7.6 and 7.7 respectively (see Table 4.8) out of 20. It should also be noted that there is a large jump in difficulty from 10th to 11th position (*Fireworks* to *Bowl Factory*). *Kangaroo* and *Fireworks* are rated 9.05 and 9.45 respectively, but the scores then jump up to 12.6, 13.5 and 15.1 for *Bowl Factory*, *Stack Computer* and *Spies*. A similar gap can be seen when going from the first three questions to the 4th question. *Bracelet*, *Animation* and *Cross Country* are rated as 3.2, 5.6 and 6.15 respectively, which in of itself covers a broad range. The score then jumps up to 7.6 for *Beaver Lunch* and *Drawing Stars*. This can lead us to roughly break the test into three distinctive levels of difficulty, with questions 1-3 being considered

Table 4.9: Test 1 Question Comparison in terms of a ranking from 1-13 and a difficulty rating from 1-20

Question 1-13	Rank	Question 1-20
Bracelet	1	Bracelet
Animation	2	Animation
Cross Country	3	Cross Country
Throw the Dice	4	Beaver Lunch
Drawing Stars	5	Drawing Stars
Beaver Lunch	6	Throw the Dice
You Won't Find It	7	You Won't Find It
Kangaroo	8	Animal Competition
Animal Competition	9	Kangaroo
Fireworks	10	Fireworks
Stack Computer	11	Bowl Factory
Bowl Factory	12	Stack Computer
Spies	13	Spies

the easiest, questions 4-10 being considered intermediate and questions 11-13 being considered the hardest.

This lines up roughly with the age categories that the questions from each test were used in during the Bebras competition. Table 4.10 presents a comparison between these three orderings. For the original category and UK results we have used the percentage in the highest category they were entered in, which can be seen in the table.

If we use the rankings in each of these columns we can rank the questions across all three columns to give an overall ranking. For example, *Bracelet* was ranked 1 in Column 1 and 1 in Column 2, giving a score of 2. If scores are identical in any of the columns then they will be given the same score e.g. in column two *Beaver Lunch* and *Drawing Stars* have a score of 7.6 so they'll both be given a value of 5 (i.e. the highest ranked question of the two).

Table 4.10: Test 1 Questions Extensive Ranking

	Col 1	Col 2	Col 3
Rank	Ranking from 1-13	Scores from 1-20	Bebras Category (Highest)
1	Bracelet (1.75)	Bracelet (3.2)	Bracelet (Inter A)
2	Animation (4.8)	Animation (5.6)	Animation (Inter A)
3	Cross Country (4.95)	Cross Country (6.15)	Animal Competition (Inter A)
4	Throw the Dice (5.7)	Beaver Lunch (7.6)	Cross Country (Inter A)
5	Drawing Stars (6.3)	Drawing Stars (7.6)	Beavers Lunch (Senior B)
6	Beaver Lunch (6.65)	Throw the Dice (7.7)	Throw the Dice (Senior B)
7	You Won't Find It (7.15)	You Won't Find It (8.75)	Fireworks (Senior C)
8	Kangaroo (7.45)	Animal Competition (8.95)	You Won't Find It (Elite A)
9	Animal Competition (7.6)	Kangaroo (9.05)	Stack Computer (Elite A)
10	Fireworks (7.95)	Fireworks (9.45)	Drawing Stars (Elite A)
11	Stack Computer (9.6)	Bowl Factory (12.6)	Bowl Factory (Elite B)
12	Bowl Factory (9.85)	Stack Computer (13.5)	Kangaroo (Elite C)
13	Spies (11.25)	Spies (15.1)	Spies (Elite C)/

In column 3 of Table 4.11 the score will be given of the highest question, so for example *You Won't Find It*, *Stack Computer* and *Drawing Stars* were all used in Elite A, so they will all be given a value of 10 as *Drawing Stars* is the highest placed in the list. Doing this for each question allows us to then rank them from 1 to 13, with 1 being the easiest question (the lowest total across all four columns) and 13 being the hardest (the largest total across all four columns). This ranking is shown in Table 4.11.

Table 4.11: Test 1 Overall Ranking of Questions

Rank	Question	Total Score	Breakdown*
1	Bracelet	6	1+1+4
2	Animation	8	2+2+4
3	Cross Country	10	3+3+4
4	Throw the Dice	14	4+6+6
5	Beaver Lunch	17	6+5+6
6	Drawing Stars	20	5+5+10
7	Animal Competition	21	9+8+4
8	You Won't Find It	24	7+7+10
9	Fireworks	27	10+10+7
10	Kangaroo	30	8+9+13
11	Stack Computer	33	11+12+10
	Bowl Factory	33	12+11+10
13	Spies	39	13+13+13

*Rank of (col1 + col2 + col3)

Analysis of Bebras Test 2

When ranking the questions from hardest to easiest (1-13) Table 4.12 presents obtained scores, with the questions being presented in order from smallest (easiest) to largest (hardest). When asked to rate the questions on a scale from 1-20, Table 4.13 presents the scores for each question. The questions are presented from smallest (easiest) to largest (hardest). Unlike in Test 1 there appears to be more of a difference between these two rankings, as shown in Table 4.14.

It is interesting to note that with this test it appears there are a few more discrepancies between the two rankings. From Questions 3 to 11 there are several questions "out of place" between the two rankings, some just one rank (like *Theatre* and *Bottles* in ranks five and six) or in the case of *Concurrent Directions* and *Party Guest*, differ

Table 4.12: Test 2 Question Rankings out of 13

Rank	Question	Rank out of 13
1	Bebras Painting	2.22
2	Tube System	4.33
3	Magic Potion	5.39
4	Party Guest	5.78
5	Bottles	6.22
6	Theatre	6.5
7	Concurrent Directions	6.78
8	Secret Messages	7
9	Scanner Code	8.89
10	B-Enigma	8.94
11	Triangles	9
12	The Game	9.89
13	Pirate Hunters	9.94

by three or four ranks. The reason for this is that these questions were all rated very similarly by most people. In terms of the difficulty score (from 1-20) there is only one point separating *Concurrent Directions* (7.72) in third position, and *Secret Messages* (8.78) in eighth position. This is what leads to the slight mismatch in those positions. Similarly, there is only one point separating *Triangles* (10.89) in ninth place with *The Game* (11.78) in 12th place. Also, of interest is the fact that *Bottles* and *Party Guest* (both 8.44) and *The Game* and *B-Enigma* (both 11.78) were rated with the same level of difficulty.

These figures allow us to split the questions broadly into three groups as was the case with Test 1. In this case, question one and two are considered the easiest, questions three through to eight are considered the intermediate questions, with questions nine through to 13 being considered the hardest questions.

These rankings line up roughly with the age categories that questions were used in during the Bebras competition. Table 4.15 presents a comparison between these three

Table 4.13: Test 2 Question Rankings out of 20

Rank	Question	Score out of 20
1	Bebras Painting	3.72
2	Tube System	5.94
3	Concurrent Directions	7.72
4	Magic Potion	8
5	Theatre	8.06
6	Bottles	8.44
7	Party Guest	8.44
8	Secret Messages	8.78
9	Triangles	10.89
10	Scanner Code	11.56
11	B-Enigma	11.78
12	The Game	11.78
13	Pirate Hunters	13.44
	Average	9.12

orderings. For the original category and UK results we have used the percentage in the highest category they were entered in, which can be seen in the table.

If we use the rankings in each of these columns we can rank the questions across all three columns to give an overall ranking. For example, *Bebras Painting* was ranking 1 in column 1 and 1 in column 2, giving a score of 2. If scores are identical in any of the columns then they will be given the same score e.g. in column two *B-engima* and *The Game* have the same score, so they'll both be given a value of 12 (i.e. the highest ranked question of the two).

In column 3 of Table 4.16 the score will be given of the highest question, so for example *Theatre*, *Scanner Code* and *Triangles* were all used in Elite B, so they will all be given a value of 11 as *Triangles* is the highest placed in the list. Doing this for each question we can then rank them from 1 to 13, with 1 being the easiest question (the lowest total across all four columns) and 13 being the hardest (the largest total across all four columns). This ranking is shown in Table 4.16.

Table 4.14: Test 2 Question Comparison in terms of a ranking from 1-13 and a difficulty rating from 1-20

Question 1-13	Rank	Question 1-20
Bebras Painting	1	Bebras Painting
Tube System	2	Tube System
Magic Potion	3	Concurrent Directions
Party Guest	4	Magic Potion
Bottles	5	Theatre
Theatre	6	Bottles
Concurrent Directions	7	Party Guest
Secret Messages	8	Secret Messages
Scanner Code	9	Triangles
B-Enigma	10	Scanner Code
Triangles	11	B-Enigma
The Game	12	The Game
Pirate Hunters	13	Pirate Hunters

4.3.6 FINDINGS

Using the average of the difficulty of the two tests as given in Table 4.8 and Table 4.13 we can see that both tests are of a similar level. Test 1 questions were rated on average at a perceived difficulty of 8.87 and Test 2 questions were rated on average at a perceived difficulty of 9.12. This leads us to be able to conclude that the two tests are of similar difficulty.

These tests were run over the course of the 2017-18 academic year and they were run in a number of schools as well as on a first year undergraduate CS course. It was delivered in schools as part of the wider CS2Go roll-out and it was decided that running it with the undergraduate students would be helpful as there is a larger, more consistent sample. It was also felt that undergraduate students could benefit from the problem solving aspect of the assessment.

Table 4.15: Test 2 Questions Extensive Ranking

	Column 1	Column 2	Column 3
Rank	Our Analysis (Ranking from 1-13)	Our Analysis (scores from 1-20)	Bebras Category (Highest)
1	Bebras Painting (2.22)	Bebras Painting (3.72)	Bebras Painting (Castors A)
2	Tube System (4.33)	Tube System (5.94)	Bottles (Junior A)
3	Magic Potion (5.39)	Concurrent Directions (7.72)	Party Guests (Inter A)
4	Party Guest (5.78)	Magic Potion (8)	Tube System (Inter A)
5	Bottles (6.22)	Theatre (8.06)	Concurrent Directions (Senior A)
6	Theatre (6.5)	Bottles (8.44)	Pirate Hunters (Elite A)
7	Concurrent Directions (6.78)	Party Guest (8.44)	Magic Potion (Elite A)
8	Secret Messages (7)	Secret Messages (8.78)	Secret Messages (Elite A)
9	Scanner Code (8.89)	Triangles (10.89)	Theatre (Elite B)
10	B-enigma (8.94)	Scanner Code (11.56)	Scanner Code (Elite B)
11	Triangles (9)	B-enigma (11.78)	Triangles (Elite B)
12	The Game (9.89)	The Game (11.78)	The Game (Elite C)
13	Pirate Hunters (9.94)	Pirate Hunters (13.44)	B-enigma (Elite C)

With both cohorts it was hoped to see if the test could be completed and structured in the 35 minute time-period allotted. It was also hoped that it could be seen from results that the test targets the students Computational Thinking skills. This is hard to really define but we used students previous mathematics and programming experience as metrics to compare groups. Mathematical ability has been shown to be a

Table 4.16: Test 2 Questions Overall Ranking

Rank	Question	Total Score	Breakdown*
1	Bebras Painting	3	1+1+1
2	Tube System	8	2+2+4
3	Bottles	14	5+7+2
4	Party Guest	15	4+7+4
	Magic Potion	15	3+4+8
	Concurrent Directions	15	7+3+5
7	Theatre	22	6+5+11
8	Secret Messages	24	8+8+8
9	Scanner Code	30	9+10+11
10	Triangles	31	11+9+11
11	Pirate Hunters	34	13+13+8
12	B-enigma	35	10+12+13
13	The Game	37	12+12+13

*Rank of (col1 + col2 + col3)

predictor of success in programming [77] and the research suggests that programming is a specific way of testing CT skills [62].

4.3.7 CONCLUSIONS

Based on the analysis from our panel we can conclude that the two tests are of approximately equal difficulty. We believe that based on previous studies, the way Bebras problems are developed and input from our research group and other educators that the Bebras problems do test students CT skills. This is further backed up by our findings from the undergraduate students discussed in Chapter 7. During this study we found that those who had previously programmed and who studied Higher Level mathematics achieved higher results in Test 1.

One interesting development that we plan to pursue would be to develop "equivalent" Bebras problems. Each Bebras exercise is usually based around a specific CS-related concept or problem. To not only make the test equivalent in difficulty but also topic, we would have to develop Bebras exercises which have the same underlying concept or idea but with a different story or real-world application. This is no easy task but if a method could be developed this would not only help our test but also allow the Bebras challenge itself to develop similar questions year after year.

An area of interest in our research group is methods of predicting success in programming courses. Being able to implement interventions to help students seen as potential struggling students is vitally important and beneficial to all educators. If this test could be shown to predict success in either programming or general academic success it could be a helpful tool for educators. We plan to use the data obtained from the undergraduate students and their final grades to begin to see if this is possible.

4.4 CS2GO WEB PORTAL

As discussed previously the development of the Computational Thinking test using Bebras problems was completed, in part, to allow teachers to conduct the test through both paper and online means. The online method for all data collection on this project were Google Forms. Google forms are a great tool but they are limited in their functionality for what is required for the CS2Go project. With this in mind, as well as the distribution of the project materials to schools, educators and students, it was decided that a dedicated web system would be developed. This section will give a brief overview of the goals, design and outcomes of this development project.

4.4.1 DEVELOPMENT

The goal of the web system was to develop a Virtual Learning Environment. The content of CS2Go was displayed and used on a third-party platform, Schoology. eLearning is a powerful tool to engage, educate and inspire students and faculty. There was an obvious need for an in-house curriculum engine. Developing an eLearning application provides many benefits, mainly due to it being independently controlled by the Department of Computer Science, this also removes the need to rely on a third-party platform. Due to these concerns, the overall functionality, look, feel and usability can be tweaked and maintained in-house.

Firstly, a UML use case diagram, which is an UML behavioural diagram was generated. Use case diagrams aid in identifying system functionalities, and categorising system requirements for an application.

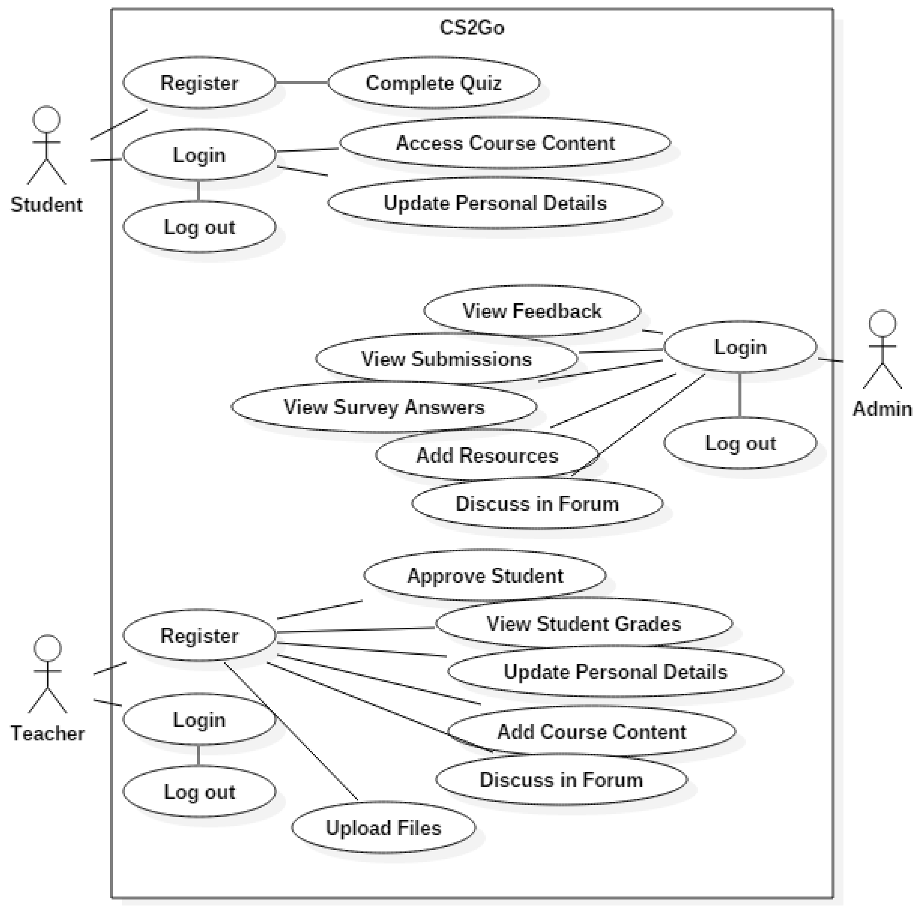


Figure 4.1: Use case diagram

In order to implement the functionalities in Figure 4.1, the following user and system requirements had to be addressed:

- The frontend should be built using HTML5, CSS3 and JavaScript. The use of outside frameworks and libraries such as jQuery and Bootstrap are permitted
- The system should allow all the content to be uploaded and downloaded from it. This includes file types such as PDF's, PowerPoint etc.
- Test and survey data should include questions and multiple-choice answers. The results should be stored in a database
- Test and survey data should be visually represented in a user friendly and informative format, via charts or a tabular layout
- The system should be developed using a python microframework called Flask, due to Maynooth University migrating all their web services to python, Flask
- Due to the possibility of front-end information being visible, user data should be stored in the back-end. This will improve security concerns around the accessibility of data
- The database layer should be a MongoDB database. Additionally, due to Maynooth University moving towards a non-relational database. Using MongoDB will allow for more flexibility and handle the ever-increasing demand for data without any issues at all
- The code base must be future proof; well designed, testable and efficient by using software development practises
- The system should be easy (for someone with a CS background) to add lessons and content to it

At the conclusion of this project, all of the above system requirements were met as well as further functionality.

4.4.2 TESTING

Browsers

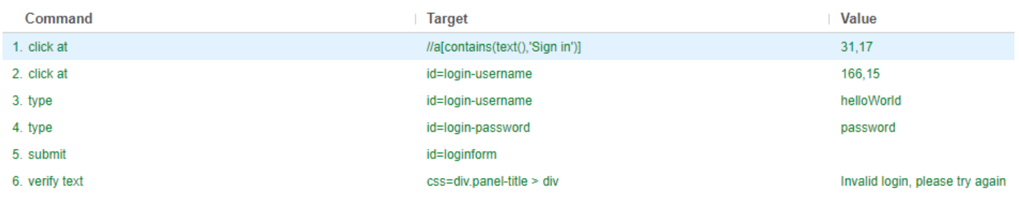
It was evident that the system should work across platforms as most people nowadays using a wide range of devices. The system worked as designed and as intended across the four major browsers (Google Chrome, Microsoft Edge, Mozilla Firefox and Safari).

Functionality Testing

Functionality testing was broken into multiple sections.

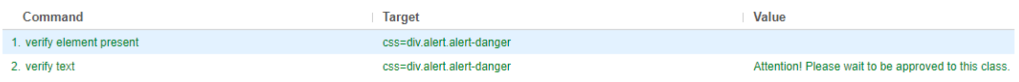
(a): Selenium Testing

Selenium is a web testing plugin which uses simple scripts to run tests directly within a browser. It provides a convenient interface for developing automated tests. The selenium plugin includes a recording feature, which records a typical users actions as they are performed, these actions can include commands such as open, click, type and many others. Verification commands are also possible which allows the testing suite to specify expected values or behaviours. This recording is then converted into reusable test scripts that can be later executed or manually alternated. Passing test cases turn green and failing test cases turn red as the commands are run in real time. Test cases were run at the end of every iteration which tested and evaluated the outcome of a given objective, if these failed the errors were fixed before moving onto the next iteration. The test cases run can be seen in Figures 4.2 and 4.3.



Command	Target	Value
1. click at	//a[contains(text(),'Sign in')]	31,17
2. click at	id=login-username	166,15
3. type	id=login-username	helloWorld
4. type	id=login-password	password
5. submit	id=loginform	
6. verify text	css=div.panel-title > div	Invalid login, please try again

Figure 4.2: Test Cases



Command	Target	Value
1. verify element present	css=div.alert.alert-danger	
2. verify text	css=div.alert.alert-danger	Attention! Please wait to be approved to this class.

Figure 4.3: Test Cases

(b): User Testing

A beta version of the web application was deployed on Heroku's cloud platform which enabled any user to sign up and view the application. A survey was created via Google Forms to gather feedback from those who tested and interacted with the web system. In this survey, the participants were asked their opinion on various questions such as 'What do you like about the UI?', 'How would you rate the UI?', 'What do you feed could be improved within the UI?', the same questions were asked based on their UX as well as any comments or suggestions regarding the application. To validate the outcome of the survey results little to no information was given regarding how to interact with the system. If a user was uncertain in relation to any given element, clearly the design aspect was not completed correctly. This was not the case. The results indicated that the majority of the participant found that the UI and UX satisfiable as shown in Figure 4.4.

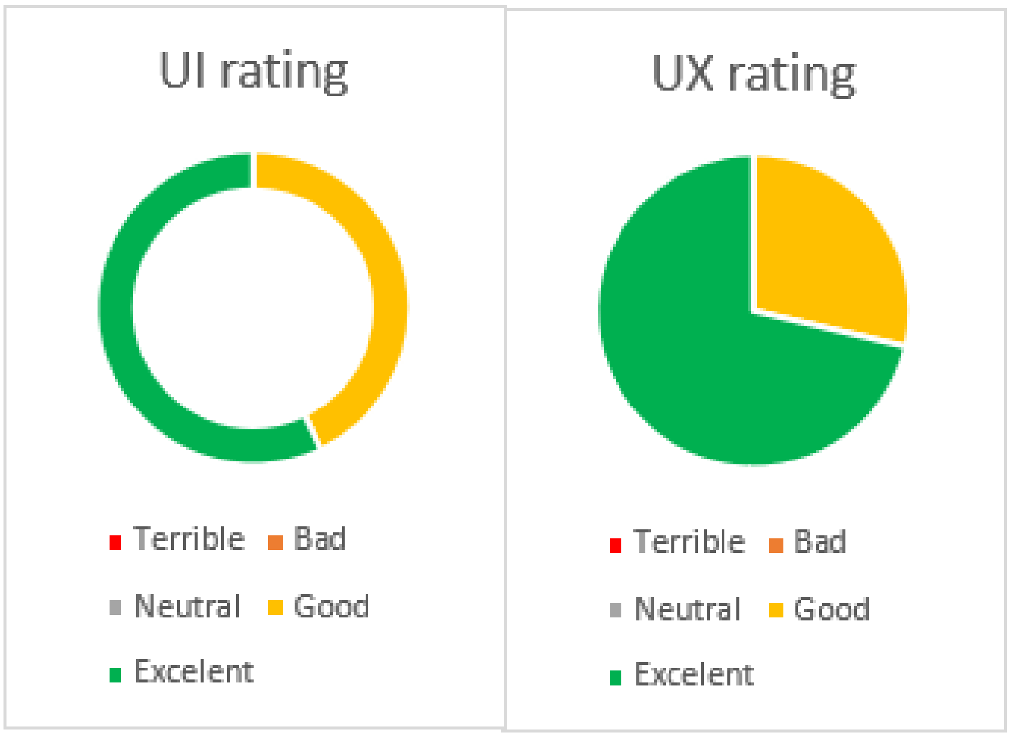
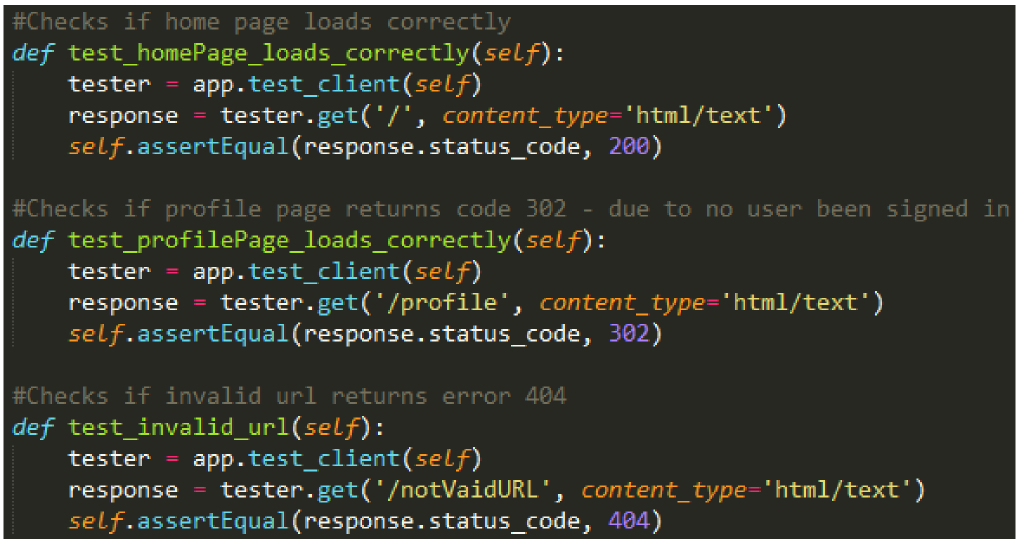


Figure 4.4: User Testing Results

(c): Unit Testing Unit testing is a form of software testing where individual units of the software is tested. The purpose is to validate that each unit of the software performs as designed. In Flask, unit tests are created using the unittest library by creating a class in the test file that inherits from the unittest's test case. Debugging the file uses the unittest library and automatically finds all test that are written and runs them and returns the output. Unit testing, alongside the testing of the systems functionality, reinforced that CS2Go functioned as intended. Some of these unit tests are shown in Figure 4.5.



```
#Checks if home page loads correctly
def test_homePage_loads_correctly(self):
    tester = app.test_client(self)
    response = tester.get('/', content_type='html/text')
    self.assertEqual(response.status_code, 200)

#Checks if profile page returns code 302 - due to no user been signed in
def test_profilePage_loads_correctly(self):
    tester = app.test_client(self)
    response = tester.get('/profile', content_type='html/text')
    self.assertEqual(response.status_code, 302)

#Checks if invalid url returns error 404
def test_invalid_url(self):
    tester = app.test_client(self)
    response = tester.get('/notVaidURL', content_type='html/text')
    self.assertEqual(response.status_code, 404)
```

Figure 4.5: Unit Tests

4.4.3 CONCLUSION AND FUTURE WORK

The aim to develop a virtual learning environment was achieved. Students and teachers can enrol in the course, access the material and run the assessments through the system. The website will be officially launched and tested with a larger pool of students in the autumn term of 2018-19. Improvements will be made throughout the year through both the research team in Maynooth and from feedback from users.

5 PILOT STUDY

5.1 INTRODUCTION

The idea to develop the CS2Go course and assessment presented in Chapter 3 and Chapter 4 arose from a need identified by our research group, having worked with schools around Ireland. We observed that teachers were keenly interested in delivering Computer Science lessons and this led to more schools and teachers joining the programme. It has been our intention from the outset to expand the content on offer and to investigate what other topics and methods could be used [66]. It was felt that there was an opportunity and a desire to create a more complete and intensive course for Transition Year, with a view to developing it into a Junior Certificate short course. The following aims, previously listed in Chapter 1, were developed for the course.

- To introduce students to Computer Science, what it is, how it can affect their lives and how they can be involved
- To improve students' view of Computer Science. This includes changing views on the gender imbalance in CS and stereotyped "nerd" view
- To improve students' Computational Thinking and problem solving skills. We hypothesise that this will make them aware of a problem solving process and how it can be beneficial in many subjects and areas of life
- To teach students CS concepts such as Algorithms, Cryptography, Sorting/Searching, etc. with a focus not just on the concepts themselves but on real-world applications
- To give students an introduction to programming so that they can write their own basic programs i.e. variables, input, operators, loops and conditional statements

Students who have participated in PACT courses in the past have commented that the modules had been both enjoyable and a good way to develop programming and other skills such as team work. However, they also stated a desire for more practical applications and we have been working to ensure that the topics and methods used in this course reflect their feedback [66].

5.2 PILOT LESSONS

Due to the nature of the study, which involved working directly with young people and collecting data from them, ethical approval was sought and received from the Maynooth University Research Ethics Committee; these forms can be found in Appendix N. This included the development of an Information Sheet and Consent form for use during this project. These can be found in Appendix P and Q respectively. During the spring term of 2017 (March - May) a pilot study was run which took sections of the developed course and delivered them over a 10-week timeframe. Over the course of the ten weeks, two lessons were delivered per week (one 40 minute class and one 35 minute class) to one Transition Year (TY) class - with students aged from 15 to 17 years of age. The study was conducted in a mixed-gender, fee-paying school (chosen as the first author was an alumnus) in a town in the east of Ireland, with an enrolment of 300 students. The school was informed of the study requirements and expectations, namely a class of at least 20 TY students with access to computers and sufficient classroom space. The Mathematics department of the school selected 22 students whom it felt would benefit from the course. Most were described by their teachers as being stronger than average in mathematics.

The aims of the pilot study were to examine the extent to which the course was enjoyable for the target age group and its content appropriate. The study also examined student outcomes including whether students had learned something from the course and whether there were any improvements in problem-solving skills or changes in attitudes to CS. After completing each lesson students filled out a feedback form described in 4.1.2 and found in Appendix H. This sought information on their understanding of the topics covered and on the extent to which they had enjoyed the lesson and learned something from it.

Over the course of the ten week period 20 classes were delivered to the students. In general, the classes consisted of non-computer based ("unplugged") lessons and Scratch programming lessons. An overview of the main topics covered in each lesson is presented in Table 5.1.

5.3 RESULTS

One of the aims of this small-scale pilot study was to determine whether the CS2Go material was enjoyable and appropriate for TY students. Care was exercised in interpreting the results as the number of participating students is small, and students were

Table 5.1: Schedule of Lessons Taught during the Pilot Study

Week	Lesson 1	Lesson 2
1	Surveys, Introduction to course	Problem solving test
2	Introduction to CS	Introduction to CT
3	Algorithms 1	Algorithms 2
4	Searching & Sorting	Searching & Sorting
5	Cryptography	Cryptography
6	Programming concepts	Scratch
7	Scratch	Scratch
8	Scratch	Scratch
9	Graphs	Problem-solving test
10	Surveys, logic puzzles/games	Logic puzzles/games

purposefully selected to take part. Although 22 students took the course, not all of them attended all the lessons or completed all the evaluation activities.

5.3.1 PERSONAL SURVEY OUTCOMES

The personal survey, which is described in Section 4.1.1 and can be seen in Appendix G, collects demographic and other personal data such as age, gender, previous programming/Computer Science experience. Table 5.2 presents a summary of the collected data for this study. As can be seen, only 21 of the 22 students completed this survey. The gender breakdown of the respondents was 12 males and 9 females. Nineteen of these students had taken higher level mathematics in the Junior Certificate Examination and two had taken the subject at ordinary level. Fewer than half of the students reported having some previous programming experience while a smaller proportion had used web technologies. The average self-assessed programming level reported was 2.44 out of 5.

Table 5.2: Demographic Data and Backgrounds of Participating Students

Demographic & other personal information	No. of students
Gender of Students	Male - 12 Female - 9
Student Age	15 years old - 7 16 years old – 11 17 years old – 3
What level of maths did you take in the Junior Certificate Examination?	Higher Level - 19 Ordinary Level – 2
Previous programming experience	Yes - 9, No - 12
Rating of programming level	Mean = 2. 44 Likert Scale (1-5) 1 = very poor, and 5 = very good
How often do you program	Mean = 1.89 Likert Scale (1-5) 1 = not at all, and 5 = daily
Programming languages previously used	Python, C#, Scratch, Javascript, C, C++
Any website/app development experience	Yes - 6, No - 15
If yes, what did you use?	HTML, Javascript, XCode, App Inventor
Have you heard of:	Codecademy - 4, Khan Academy - 1, Call to Code - 1, None of the given options or similar - 16

5.3.2 LESSON FEEDBACK OUTCOMES

The results of the analysis to the question, "did you enjoy the class?", in the lesson feedback form, are provided in Table 5.3 along with a gender breakdown. A Likert Scale rating of 1-5 was used for responses, where 1 = "I did not enjoy the class at all", and 5 = "I really enjoyed this class". Overall mean ratings are similar across lessons (ranging from 4.1 to 4.6). The range in values is somewhat larger for male students (3.5 - 4.6) than for female students (4.0 - 4.7). Female students had the highest enjoyment ratings for the *Introduction to Computer Science* lesson while male students had the highest enjoyment ratings for the *Algorithms* lesson.

Table 5.3: Lesson Feedback – Average "Enjoyment of Class" Ratings by Gender

Topic name	Total	Males	Females
Introduction to Computer Science	4.6 (8)	4.5 (2)	4.7(6)
Introduction to CT	4.1 (9)	3.5 (2)	4.3(7)
Algorithms	4.5 (16)	4.6 (10)	4.3 (6)
Searching and Sorting	4.4 (16)	4.3 (7)	4.6 (9)
Cryptography	4.2 (12)	4.4 (8)	4.0 (4)
Programming concepts	4.5(13)	4.4 (9)	4.5 (4)
Graphs	4.4 (13)	4.4 (9)	4.3 (4)

Mean ratings (5 = I really enjoyed this class, 1 = I did not enjoy this class at all)

The *Introduction to CT* lesson had the lowest enjoyment rating for male students whereas female students had the lowest enjoyment rating for the lesson in *Cryptography*. There are however, no statistically significant gender differences.

5.3.3 END-OF-COURSE SURVEY

On the final day of the course, a feedback form was distributed and completed by 17 students who were present on the day. The questions and responses are presented and discussed next.

- **Did you enjoy the course?** Analysis of responses to this question indicate a class average rating of 3.8 (out of 5). Males, with a score of 4.0, had a more enjoyable experience of the course than females, who had a score of 3.5.
- **What did you like?** Students enjoyed learning about Computer Science for the first time and learning new things about mathematics and computers. Group work was also mentioned as something students liked along with the games. Referring to the *Algorithms* lessons, students stated that they "*enjoyed...working with my friends*" and "*liked how practical the class was*". The unique concepts and variety of classes were also mentioned as highlights.
- **What didn't you like about the course?** Not much information was given in response to this question. One student reported not liking programming with Scratch. From observation and conversations with students, we believe that hands-on activities were more popular than Scratch programming. Two students stated that the course seemed very long, although it should be noted that one had very low attendance. It was also stated that the course was complicated and that there was not enough help.
- **Favourite class/activity or topic covered?** The lessons on Problem solving, Scratch, principles of Computer Science, Algorithms, doing the test, and Linear sorting were identified as favourite activities by different students.
- **What was your least favourite class/activity or topic covered?** Scratch, including understanding the tutorials, some of the puzzles, Algorithms, and different ways to filter code were mentioned as least favourite activities by different students.
- **Do you feel like you learned something during the course?** Apart from one negative response, all students responded positively to this question.
- **What did you learn?** In response to this question, students mentioned new computer skills, how to solve different types of problems, new Computer Science terms, how to programme using Scratch, how to make games, what Computational Thinking is and how it works, and different types of sorting.
- **If you could change something about the course what would it be?** Of the small number who responded to this question, the suggestions included making the tutorials easier, providing more help, setting aside programming, making the classes longer, adding more practical material, and identifying more everyday uses.

Overall, students reported learning a wide array of skills. These included core Computer Science concepts such as programming, sorting and problem solving. In addition, they gained an understanding of what CS is, what CT is, and the differences between them.

Analysing the feedback, it was observed that the problem-solving aspects of the course were well received with several students commenting that they enjoyed the different puzzle types used. Puzzles and problem-solving activities are central to the course and allow students to improve their CT skills - a key requirement for a computer scientist. Students also enjoyed the variety of concepts introduced in the course and the different teaching styles that were used. Using this feedback, further adaptations were made to include more tutorials and support documentation to help students who are struggling.

5.3.4 PROBLEM-SOLVING TESTS

The first problem-solving test (Test 1) was administered to 22 students at the beginning of the course. Of these, two students completed Test 1 after attending one class/lesson and one other student completed it after attending two classes/lessons. The second problem-solving test (Test 2) was completed by 19 students in the final week of the course. Results of Test 1 and Test 2 are presented in Table 5.4. To provide anonymity for the students they were each assigned a letter to use when filling out the feedback forms and assessments. This letter system is used in the tables below.

The average scores are 7.18 out of 10 for Test 1 (n = 22) and 6.84 out of 10 for Test 2 (n = 19). Differences between the scores are not statistically significant (t-score = 0.51, p = 0.61). The average scores of students who took both tests (n = 19) are 7.47 (Test 1) and 6.8 (Test 2) - again the differences are not statistically significant (t-score = 1.00, p = 0.32). A finding of note is that in both tests those students who had previous programming experience (n = 9 in both) performed better than those who did not have that experience (n = 12 for Test 1, n = 10 for Test 2). Those with programming experience averaged a score of 8.3 for Test 1 and 8.0 for Test 2, while those without experience averaged 6.8 for Test 1 and 5.8 for Test 2. For both groups, differences in average scores between the two tests are statistically significant (T-Score = 2.13, P-Value = 0.05 and T-Score = 3.45, P-Value = 0.00 respectively). This could indicate that programming is a vital tool in improving students' problem-solving and CT skills.

Table 5.4: Outcomes of the Computational Thinking Assessments

Student	Test 1	Test 2	Student	Test 1	Test 2
A	8	10	L	8	7
B	7	-	M	8*	5
C	8	7	N	5	6
D	10	11	O	10	7
E	9	7	P	10	7
F	6	5	Q	9	7
G	7	6	R	1	-
H	8	-	S	5	3
I	7	6	T	5**	7
J	9	9	U	3*	4
K	8*	8	V	7	8
			Mean	7.2	6.8

Maximum score = 13 (1 per problem); * test taken after one class; ** test taken after two classes.

5.3.5 VIEW OF COMPUTER SCIENCE SURVEY

Of the 22 students enrolled to take part in this course, 18 filled out the pre-course survey, while 15 completed the post-course survey. The data obtained from these are displayed in Table 5.5. The numerical values range from 1-5, with 1 being "Strongly Disagree" with the statement and 5 being "Strongly Agree" with the statement. For the Yes/No questions, only Yes/No answers are given; there was also an option for Maybe on the first three questions.

The data in Table 5.5 shows that the course does not appear to have had much impact on students' interest in CS as a subject at third level. One interesting point to note is that more students said that CS is not interesting and that it is challenging in the post-course survey compared to the pre-course survey. This finding could be seen as disappointing, but we believe that students who are exposed to CS at an earlier age

Table 5.5: Results of the View of Computer Science Survey from the pilot study students

Question	Before (n = 18)	After (n = 15)
Have you considered studying CS in university?	Y – 3, N - 7	Y – 3, N - 7
Is CS interesting to you?	Y -10, N - 1	Y – 7, N - 3
Is CS challenging?	Y – 7, N - 2	Y – 8, N - 2
Using the internet is central to CS	3.44	3.27
Using Word, Excel etc. is central to CS	3.22	3.07
Installing software (e.g. Windows, iTunes) is central to CS	3.67	3.2
Programming is central to CS	4.56	4.13
Being able to solve different problems is central to CS	4.67	4.6
CS is an area related to maths	4.06	3.6
A computer scientist should be good at working with others	3.61	4
Boys/men are more likely to study CS then girls/women	3.28	2.6
Work in CS can be done without a computer	2.56	3.13
Have you heard the term CT?	Y – 5 N - 13	Y – 9 N - 6

Note: Y = Yes; N = No.

will be able to make more informed decisions in the future. Currently students do not have exposure to CS at second level and any exposure, be it positive or negative for students, can contribute to informed decisions about choice of third-level courses. CS has one of the highest drop-out and non-continuation rates across all third-level subjects [77] and so if students find out that they don't like studying CS or are not suited to it then this is valuable.

5.4 PRACTICAL LESSONS LEARNED FROM THE PILOT STUDY

One of the main aims of the pilot study was to learn what obstacles there could be to teachers using the content and whether the lessons would translate well from the lesson plans to the classrooms. The feedback previously presented along with the experience of teaching it influenced changes and adaptions to various lessons. In this section we will look at some of those practical lessons learned from the teaching experience.

5.4.1 ATTENDANCE

Due to the fact that TY students were involved in the study, a problem arose in relation to inconsistent attendance. TY is a programme in which students are encouraged to try out new subjects, engage in new activities and pursue extra-curricular activities. This makes it a perfect fit for trialling a Computer Science course. However, extra-curricular and other activities can disrupt class attendance which in turn can impact in a negative way on programming lessons. Programming is a skill that is learned incrementally and so students who miss lessons will need to catch up as the topics and tools taught in each lesson are vital to the next phase of learning.

5.4.2 FACILITIES

Two rooms were used for the duration of this study. The first was an open-spaced room with movable chairs and tables. This was a perfect location for the concept lessons which involved individual pen and paper activities as well as group activities. The classroom structure was also beneficial for active learning and student-led activities; it was set up with tables allowing groups of four students to sit around and work either together or in two pairs on most activities. From our experience this kind of group work is both enjoyable and promotes learning, though students took a while to adapt to the sequence of teaching employed in the course. In general, this involved a brief introduction followed by group, pair or individual work. Several comments were made in feedback forms to the effect that students enjoyed the interactive nature and group work.

The second room used for the study was a computer room with roughly 20 computers. These were older machines which meant several of them crashed during the lessons and lost internet connection on occasion. Also, there was no screen and no wall for projecting onto, so we could not show students how to use Scratch, create an account, or navigate to tutorials. These flaws were mentioned by students who would have appreciated a better introduction to Scratch. Considering our experience of this course, or any similar course that includes programming, it is important that good-quality facilities are made available. Primarily this means students having one-to-one access to a personal computer with fast internet access and the memory and processing capability required to run multiple software programs. Projectors or TV monitors which allow teachers to show students examples and "live-code" are also necessary to make the teaching experience easier for both students and educators.

5.4.3 CLASS TIME

Instruction time was limited to one 40-minute class and one 35-minute class per week, separated by a lunch break. This meant that many students did not have an opportunity to complete as much of the Scratch course as they would have liked. Another issue was that with the break, and students returning late, the second lesson was often reduced to 30 minutes. This, combined with the need to fill out a feedback form at the end of each lesson/class, reduced instruction time considerably. This, in turn, meant that a few exercises were not tested. All things considered, however, the experience was beneficial as we have changed our approach to lesson plans for teachers in the final course that was developed. All lessons in the completed course were designed with enough content to last for one hour for schools that have hour-long classes, but will allow schools with shorter classes to remove certain (optional) activities whilst including the more core parts of the course.

5.5 CONCLUSIONS

5.5.1 OVERALL RESPONSE TO THE COURSE

While the overall feedback from students was positive, we were aware that considerable work was required to prepare for the delivery of Computer Science courses in schools and the integration of CT into other subjects. As a result of the study, we altered various lessons and fine tuned modules such as Scratch to ensure that the materials developed are sufficiently challenging, interesting for students, and age-appropriate. This ensure that when the main study, described in Chapter 6, was conducted the content was improved. This will be a continual process as lessons are developed and taught and feedback is gained from students and teachers.

5.5.2 PERFORMANCE ON THE PROBLEM SOLVING TEST

As noted in Section 5.3.4, students on average scored slightly lower on the second problem solving test compared to the first. Although not statistically significant this was of course a disappointing result. We believe that several factors might have led to this; one is the sporadic attendance of students as discussed in Section 5.4.1. Another reason could have been due to the length of time that the study was delivered in. It has been shown that introducing CT to students can take time and so the short amount of time given may not have been sufficient to significantly impact their CT skills. It should also be noted that although the course includes lots of activities which are designed to improve problem solving skills the students are not "trained" on Bebras problems, so they may not necessarily improve in this test environment with the short time period given. The need for a larger scale study had already been noted and this is described in detail in Chapter 6.

Part III

TESTING THE COURSE AND RESULTS

6 SCHOOLS STUDY

6.1 introduction

With the course content largely developed and a Pilot Study having been successfully undertaken in spring of 2017 (as described in Chapter 5), the next phase was to enlist other teachers and classes to use the content. This would allow for more feedback and data to analyse the course's impact and usefulness in Irish schools.

6.1.1 participant information

The hope for this study was to have around 100 students taught components of the CS2Go course and to receive feedback from them, their teachers and have the students complete the major assessments described in Chapter 4. In some regards this goal was reached, but in general it became clear that it would not be entirely possible in this iteration of the project for several reasons which are discussed in Section 6.4.1.

Overall, students in 7 schools participated in some part of the study e.g. filling out surveys or taking the CT test. It is known from conversations with other educators and teachers that the content has been used more widely. These students come from a variety of different settings including mixed sex, all-boy and all-girl schools, large public schools, smaller private schools and schools in urban areas along with some in more rural settings. The previous CS experience of the various teachers also varied with some having qualifications in CS and vast experience teaching, whilst others had limited knowledge. Some taught several lessons and completed feedback, others just did some of the major parts of the assessment and one or two lessons, others were control groups and others took part in classes without filling out the major assessment parts described in Chapter 4.

It is known from anecdotal conversations that more teachers and classes used the course and the content throughout the year (using Schoology). Over 80 educators were given access to the content throughout the year. The hope is that in future this will be more controlled and structured but for now we present the data gathered.

One thing to note is that for the *View of CS* survey and Computational Thinking Assessment, students took these tests at a variety of times and environments. Students also may have used classes developed in this course recently, or not for months and may or may not have done a large amount of other CS related content. All the results then cannot be attributed to the course, whether negative or positive, but instead show a representative of the whole years growth. Exceptions to this include the comparison of the control and study groups as well as the two classes in the same year group.

6.2 CLASS FEEDBACK

In this section we will look at the general feedback received from both students and teachers on the developed lessons. This will include a general enjoyment level determined by a Likert scale (1-5), multiple-choice questions as well as quotes from students on what they liked and felt they learned during the lesson. Due to the number of responses and their length, a small subsection of quotes will be presented with the hopes of showing a balanced yet representative view of each class.

6.2.1 COMPUTER SCIENCE CONCEPTS

This was the first module designed and is recommended as the starting point for teaching the course. The "unplugged" or "computer-less" nature of these activities can also be appealing as some schools suffer from a lack of equipment to allow whole classes to do lessons on programming or other computer-based topics. This allows teachers to be more flexible when planning out their schedules around booking computer rooms and other such practicalities. These lessons are described in more detail in Section 3.4.1.

As previously discussed in Chapter 4, most lessons had an associated feedback form for both teachers and students. With this module there are some questions common to all feedback forms, and some unique to each form. The common questions include the question "Did you enjoy this class?"; students are then asked to rate this on a Likert scale from one to five, with one being they really did not enjoy the class and five being that they really enjoyed the class. A number of these lessons also included multiple-choice questions on the topics covered. These are designed to quickly sample if students retain any information shown to them during the lesson. These questions are unique to each topic and so differ from form to form, the results of these are presented in all cases. The following section presents the results grouped by the class they were associated with.

Introduction to Computer Science

Table 6.1 presents the overall high level feedback in relation to the class entitled "Introduction to Computer Science". It can be seen that this lesson was well received by students with an average rating of 3.51 out of 5. Almost 70% of students said they felt like they learned something during this class.

Table 6.1: Introduction to CS class - High-level Feedback

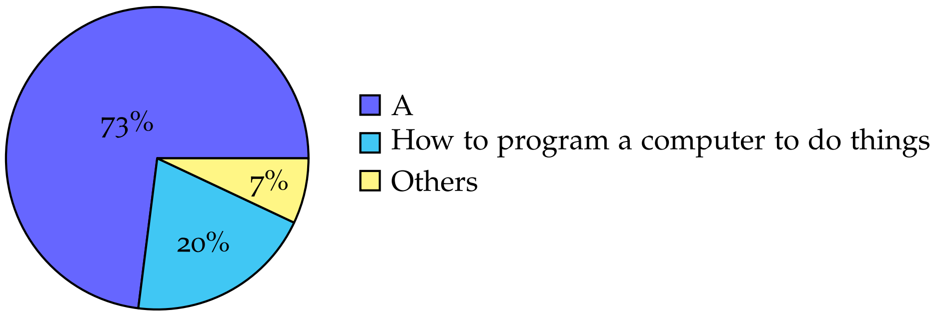
No. of Students	Did you enjoy this class? Average rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
111	3.51	76 (68%)	5 (5%)

Fig. 6.1 presents the findings in relation to the question *"What is Computer Science?"* For this question students were given 5 possible answers as well as an option to add their own answer. Seventy three percent responded that Computer Science is *"How to solve real-world problems using the power of computers"* with 20% responding that it was "How to program a computer".

Figure 6.2 shows how the students responded to the question *"Which of these Problems could a Computer Scientist help Solve?"*. It should be noted that a student could select as many of these options as they wanted. Just over 80% (89 out of 111) selected "Yes" to a computer scientist being able to "help people communicate easier around the world". We believe this is significantly higher than the other responses as the lesson involves an activity to help a person communicate who has locked-in syndrome. The other answers were chosen to allow all of them to possibly be correct. For example, you could argue that CS can help figure out why Ireland lost to Wales in a rugby match using statistical analysis and you could also use CS to discover which pizza is the best in town, perhaps through a review/ranking website or algorithm.

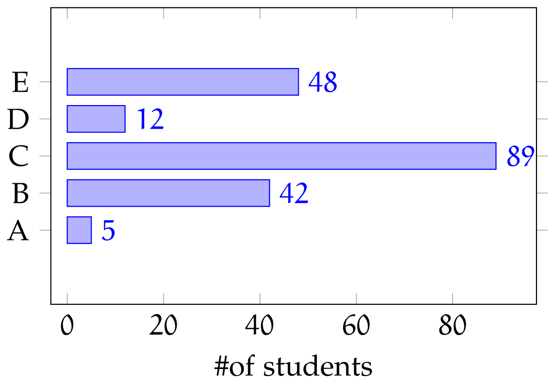
It is clear that with the other responses students still don't see the far-reaching possibilities of CS with no other option being selected more than 44% of the time (48/111 responses for the option "The quickest route between a set of cities").

Figure 6.1: What is Computer Science?



A - How to solve real world problems using the power of computers

Figure 6.2: What problems could a Computer Scientist solve?



A - Why Ireland lost to Wales in a rugby match

B - The best way a company can increase profits without firing someone

C - How to help people communicate easier around the world

D - The best pizza in town

E - The quickest route between a set of cities

We will now present a sample of feedback quotes from students on what they enjoyed about this class.

- "I liked how it was different from the computer room but I didn't understand much of what was happening" (Female, 15)
- "It makes me think about the method to solve problem!" (Female, 16)
- "I liked the problem solving and learning what computer science really is" (Female, 16)

- "I enjoyed learning about computer science as I didn't really know about it and how effective it is in todays world made me want to start coding" (Male, 16)

It should also be noted here that there was some negative comments about the class being "boring" and "uninteresting". It can be seen in Table 6.1 that 68% of students felt they learned something from this class. We will now present some examples of what students felt they learnt in this lesson.

- "I had a vague idea what computer science was but now its clearer" (Male, 15)
- "Computer science is more about problem solving than working with computers" (Male, 16)
- "I learned that coding is about problem solving in the world of today and what coding is all about" Male, 16
- "That I do not need to be great at maths to try coding" Female, 16

Teacher feedback

Three teachers provided feedback on this lesson, all from mixed sex schools. All teachers felt that their students learned something from the class, with two of the three saying they felt they enjoyed it; the third was not sure if the students enjoyed it. Feedback in general was positive with teachers finding that students enjoyed the class, especially the *Locked-In Syndrome* activity as well as working together. One thing that was noted by a teacher was that students' lack of understanding of what CS is hindered the initial discussion. This shows that a class covering these concepts is vital to explain to students what CS is and is not.

Teachers found the lesson plan and materials sufficient, but some recommended more pictures on the slide shows and one commented how they needed to "brush up on terminology" and that they "could return to this lesson more confident if I teach it a couple more times". All teachers responded that they would teach this lesson again.

Introduction to Computational Thinking

The *Introduction to Computational Thinking* class is made up of a few tasks but is more theoretical in nature than the *Introduction to CS* class. The class was again well received by students, as can be seen in Table 6.2, with an average rating of 3.54 out of 5 for student enjoyment levels of the class. Additionally 74% of students said that they learned something.

Following are some quotes from students about what they did and did not like about the class:

- "I liked the mad lip activity I thought it was funny and interesting" (Female, 16)

Table 6.2: Introduction to CT class - High-level Feedback

No. of Students	Did you enjoy this class? Average rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
69	3.54	51 (74%)	6 (9%)

- "It was fun and engaging and I learned a lot" (Female, 15)
- "The games. The solution to the number problem was interesting. The stories were funny" (Female, 15)
- "I liked the way you can solve big problems in small parts" (Female, 15)
- "When we have to work together" (Female, 15)

In addition to this feedback, specific questions relating to the content covered in this lesson were asked. The *Mad Lib* exercise about abstraction was well received and 72% correctly identified the definition for abstraction as shown in Figure 6.3. It can be seen that 85% of students correctly identified the definition for decomposition, as shown in Figures 6.4, and 82% of students correctly identified the definition of Computational Thinking, as seen in Figure 6.5. These are all positive findings and suggest that this lesson works well and meets the expected learning outcomes.

Some students found the concepts covered in this lesson hard or difficult as can be seen in the following feedback:

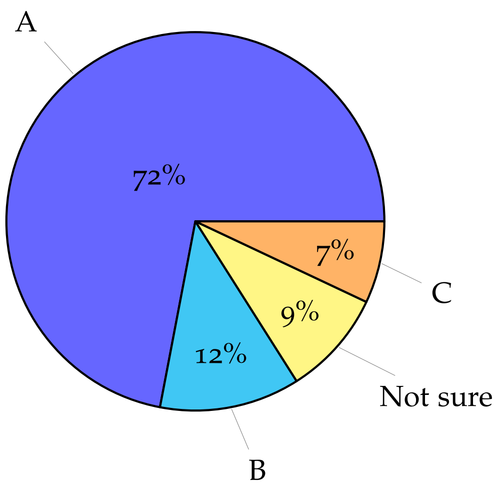
- "It was an okay class, I found the maths problems quite hard too." (Male, 15)
- "I liked the game we played at the end but I was confused about how that had anything to do with computers and computer science." (Female, 15)

Other students found the class boring:

- "didn't really interest me" (Female, 15)
- "was kind of boring. didn't really enjoy it that much." (Male, 15)

Seventy four percent of students said they learned something in this class, which is encouraging as introducing students to CT is one of the key aims of this course. Following are some quotes of what students felt they learned during this lesson:

Figure 6.3: What is abstraction?

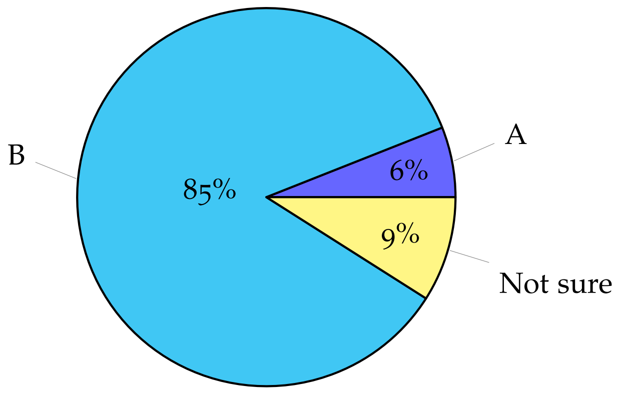


A - Pulling out specific differences to make one solution work for multiple problems

B - Breaking down a problem into smaller parts

C - A list of steps that you can follow to finish a task

Figure 6.4: What is decomposition?

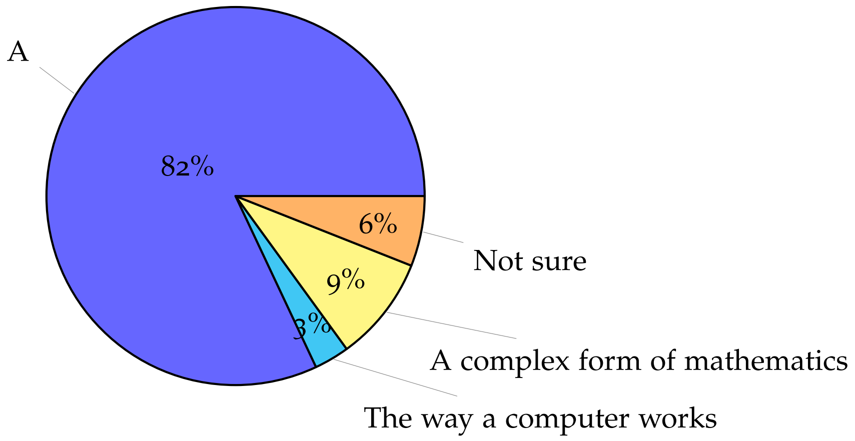


A - Pulling out specific differences to make one solution work for multiple problems

B - Breaking down a problem into smaller parts

- "I learned more about computer science and what is involved, decomposition and abstraction" (Male, 15)
- "What computational thinking is" (Male, 16)

Figure 6.5: What is Computational Thinking?



A - A technique to solve problems

- "I learned the steps in computational thinking, looking for patterns, making solutions for multiple problems, steps to follow a track" (Male, 16)
- "How to tackle different problems step by step" (Male, 15)

Teacher feedback

Two teachers provided feedback on this lesson, both taught the lesson in a mixed sex school. Both of these teachers felt that students enjoyed the class and learned from it. Teachers commented that most students enjoyed the *Mad Lib* activity as well as the *Spit-Not-So* game. The teachers felt that the method for adding up the numbers from 1 to 200 was also engaging for the students. Teachers felt that the main learning outcome for their students from this class was about how decomposing problems can make them easier to solve.

As with other lessons teachers felt that more time could be spent on the lessons, with one saying that being more informed themselves of the terminology could help with the delivery of the lessons. One teacher asked for more definitions of terms such as nouns, verbs etc. to make the *Mad Lib* activity easier. Both teachers said that they would teach this lesson again.

Algorithms 1 & 2

It should be noted that some teachers combined this class with a lesson on *Cryptography*, which can be seen from their responses to the feedback forms. However, as they filled out their feedback on the *Algorithms* form the results have been included in this section. Qualitative answers that refer to the *Cryptography* classes have been left out of this section.

Some teachers also combined this class with a lesson on *Searching and Sorting*, which can be seen from their responses to the feedback forms. However, as they filled out their feedback on the *Algorithms* form the results (both qualitative and quantitative) have been included in this section. Qualitative answers that refer to the *Searching and Sorting* classes have been left out of this section.

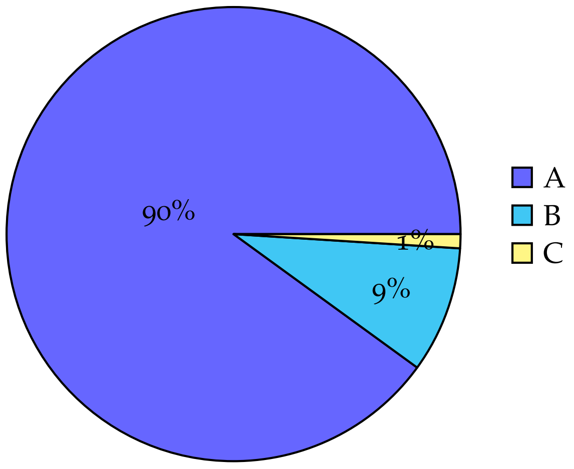
The *Algorithm* classes were well-received in general, although not as well as the previously discussed lessons. The classes received a 3.37 rating for *Algorithms 1* and a 3.22 rating for *Algorithm 2*. This can be seen in Table 6.3. The two classes are very different in their design but in general they are aiming to give students a very concrete understanding of what an algorithm is. Students should also see the importance of algorithms and this will be highlighted further during both programming modules. Students also felt they learned less than in the previously discussed lessons, with 61% thinking they learned something in *Algorithms 1* and only 43% thinking that they learned something in *Algorithms 2*. However, even though those figures aren't encouraging, this could be that many students weren't sure whether they learned something. This is clear when you see that only 13% for Algorithms 1 and 21% for Algorithms 2 saying they didn't learn something. However, as shown in Figure 6.6, Figure 6.7 and Figure 6.8 over 70% of students correctly identified some of the key definitions from the lessons. These definitions are for key algorithmic concepts which were "What is an algorithm?", "What is Step-wise refinement?" and "What is ambiguity?". This is encouraging and shows that the vast majority of students could answer those questions correctly, this may be down to the fact that they had heard these definitions before.

Table 6.3: Algorithms 1 & 2 - High-level Feedback

No. of Students	Did you enjoy this class? Average rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
76	3.37	46 (61%)	10 (13%)
68	3.22	29 (43%)	14 (21%)

These two classes were well received in general, with some feedback comments from students for *Algorithms 1* now presented:

Figure 6.6: What is an algorithm?

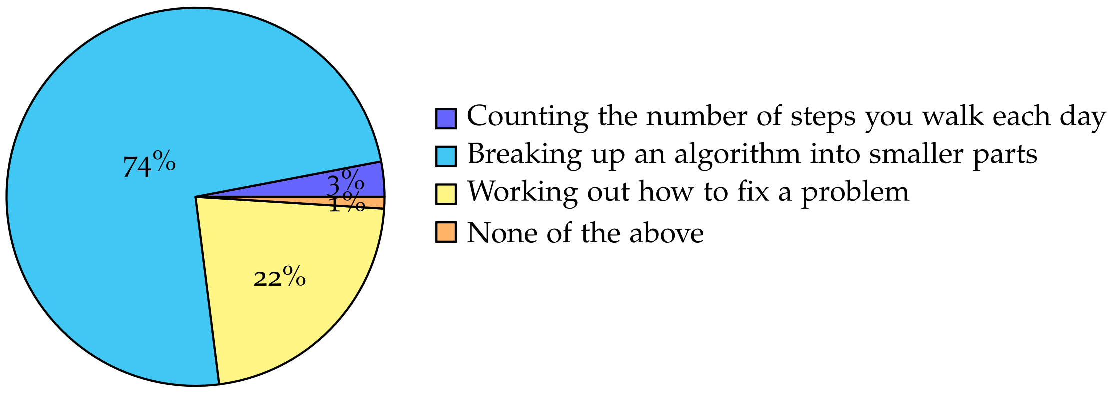


A - A set of instructions of how to solve a problem or how to do something

B - A way in which a computer understands human instructions

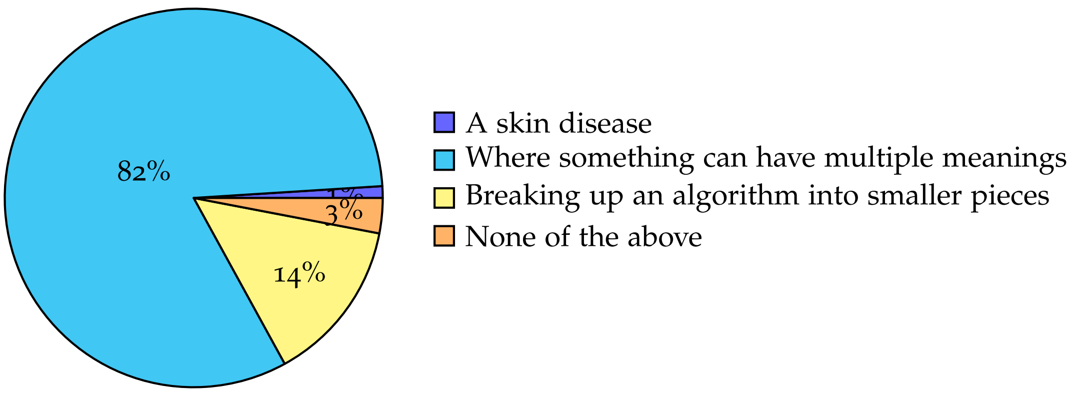
C - A rare type of alligator found in the Amazon rainforest

Figure 6.7: What is Step-wise Refinement?



- "I learned a lot of new things" (Female, 15)
- "I like hands on work" Male, 16
- "Completing tasks in groups with classmates" (Female, 16)
- "I enjoyed learning about Algorithms and how to solve them" (Female, 16)
- "I enjoyed the lightbots" (Male, 15)
- "I though it was fun to make steps of how to make a cup of tea" (Male, 16)
- "Found it slightly boring as I don't have much of an interest in computers" (Female, 15)

Figure 6.8: What is ambiguity?



- "Some parts were quiet difficult and boring" (Female, 16)
- "It was boring" (Male, 15)

Over 60% of students felt they learned something in *Algorithms 1*, with some of their quotes on what they felt they learned being:

- "How to use step-wise refinement" (Male, 15)
- "I learned what ambiguity is" (Male, 15)
- "How to break a task down into smaller steps." (Female, 16)
- "How to develop my problem solving skills and I liked the algorithms." (Female, 15)
- "I learnt that we can instruct a computer or program to carry out any operation we want" (Female, 15)

Algorithms 2 was also well received with a rating of 3.22 out of 5. Some of the comments students made about what they enjoyed or didn't enjoy are presented here:

- "I enjoyed making the Lego sculptures" (Female, 16)
- "I enjoyed getting to create my own algorithm." (Female, 15)
- "Worked in groups" (Male, 16)
- "i learned about how hard it is to make algorithms" (Female, 15)
- "I didn't think it was very interesting or beneficial to me." (Female, 15)
- "I found it very repetitive" (Female, 15)
- "the Lego was a bit different but seemed a bit irrelevant from what we were learning." (Female, 15)

For the *Algorithms 2* class, students in general felt like they learned less or were not sure if they learnt anything from the lessons. This could be due to students either already knowing the concepts if they had previously completed the lesson *Algorithms 1*, or the fact that the second lesson on algorithms does not include as much explanation of the usage and importance of algorithms. The following are some of the feedback quotes from students on what they felt they learnt during the classes:

- "How computers understand us and how we can use computers to help us" (Female, 16)
- "I learned that you must be specific when writing algorithms" (Female, 16)
- "to make an easy to follow and specific algorithm" (Female, 15)
- "I learned about what an algorithm is and how it works" (Female, 15)
- "I learned how to write more detailed instructions." (Male, 15)

For those who didn't feel they learned anything, or weren't sure, here are some of their reasons why they felt this way:

- "I already knew most of the things being taught" (Female, 15)
- "Already knew about topic" (Male, 16)

Teacher feedback

For both lessons, two teachers provided feedback. One taught the lesson in a mixed school, the other in an all boy's school. From their feedback of *Algorithms 1*, both teachers felt that their students learned something (namely what an algorithm is) from the lesson, although one was unsure if their students enjoyed it. One teacher asked if more information could be provided on who uses algorithms, with both not using every activity in the lesson plan. One teacher said they would use it again, with the other cutting it short as not all students engaged.

Regarding *Algorithms 2*, both teachers felt that their students enjoyed and learned something from the lesson, with the obstacle course and Lego structures being singled out as highlights. Learning outcomes mentioned include working in teams and how important listing instructions clearly can be. One comment made by both teachers was on limiting the number of Lego blocks for that activity as students get distracted making complex structures. It should be noted that the lesson plan gives guidance that 6-10 blocks should be used in this activity per group. Both teachers felt that they would use the lesson again in future.

Cryptography

The topic of Cryptography is split in CS2Go into four separate lessons. It can be seen from Table 6.4 that *Cryptography 1* was one of the least liked of all the classes in the first module with a rating of 2.47 out of 5, and only 47% of students feeling like they learned something. Firstly, students commented how difficult it was to read the text on the slides; especially important when reading off encoded messages. After speaking to the teacher this was due to the projector. Teachers are encouraged to download the slides and edit them as needed, including the text colour and size, so hopefully this won't be an issue in the future. Secondly, teachers commented on how long it took to decode some of the ciphers. Teachers will be encouraged to give hints if necessary in future. Others who did like the class said that they "liked the idea of how cryptography applied to real life" and others liked using the Caesar shift cipher.

Table 6.4: Cryptography 1, 2, 3 & 4 - High-level Feedback

Lesson	No. of Students	Did you enjoy this class? Avg rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
Crypto 1	17	2.47	8 (47%)	5 (29%)
Crypto 2	24	3.96	19 (79%)	4 (17%)
Crypto 3	15	3.8	11 (73%)	4 (27%)
Crypto 4	19	3.95	16 (84%)	3 (16%)

The low enjoyment levels, and perhaps the other issues with slides, led to a low percentage of students saying they learned things. However, as shown in Figure 6.9 and Figure 6.10, 90% of students correctly identified the correct definition of cryptography and 88% correctly identified the Caesar shift cipher as a well-known cipher. It can be seen in Figure 6.11 that only 37.5% of students could correctly describe the method for a substitution cipher - this could be down to some students not getting to this section of the lesson during the time period. Students who felt they learned something included things like "Cryptography definitions", learning how to "brute force a code" and "the Caesar shift cipher".

As shown in Table 6.4 *Cryptography 2*, *Cryptography 3* and *Cryptography 4* are some of the most well-liked classes in the whole course, with ratings of 3.96, 3.8 and 3.95 out of

Figure 6.9: What is Cryptography? (asked across Cryptography 1 & 2)

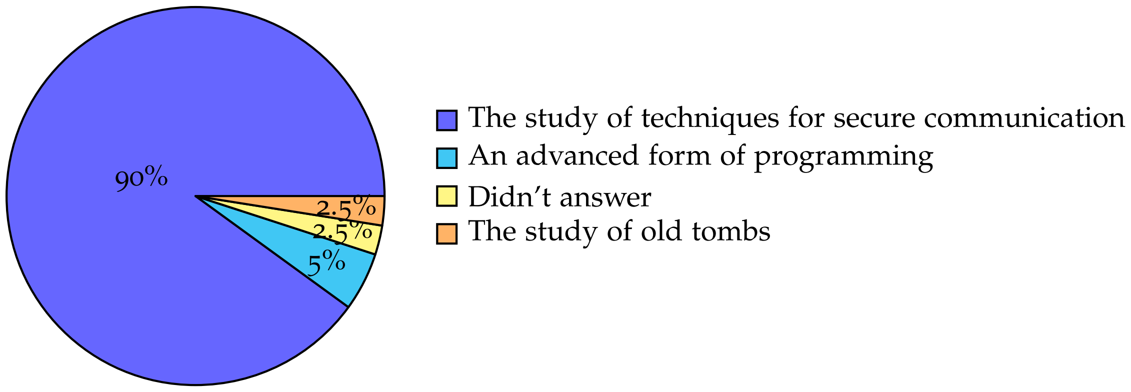
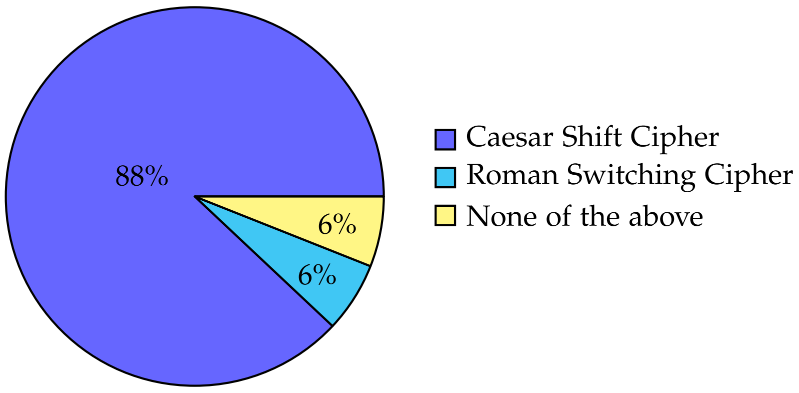


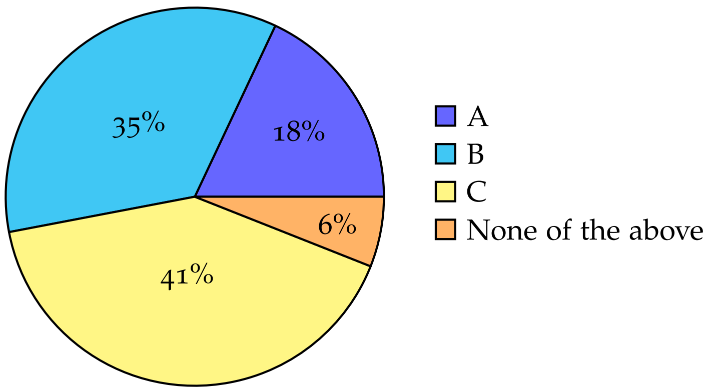
Figure 6.10: Which of these is a well-known cipher? (asked only in Cryptography 1)



5 respectively. Students also felt like they learned something from these classes with at least 73% agreeing with the question in each lesson. Following are some quotes from students on what they liked or disliked about the classes, all three lessons are included together as it was the same cohort of students who took these lessons.

- "I enjoyed solving the puzzles" (Female, 15)
- "I liked decoding using the frequency attack" (Female, 16)
- "I enjoyed the group work decryption" (Male, 16)
- "I liked the maps and learning how to decode them" (Female, 15)
- "Because I really learned something" (Female, 16)
- "I enjoyed the class as it was fun decrypting the different codes/sentences" (Female, 15)
- "I thought it was fun but the 1st part was too long" (Male, 15)

Figure 6.11: How would you use a substitution cipher to encode the following: "This is a secret message"? (asked only in Cryptography 1)



A - Take each letter and turn them into a number i.e. A->1 B-2 etc.

B - Take each letter and randomly assign it to another one i.e. A->T B->F etc.

C - Take each letter and move them all a certain number up i.e. moving them 3 would make A->D B->E etc.

This last quote refers to an activity in *Cryptography 3* which includes a letter-frequency attack. The length of this was an issue that several students pointed out, this activity has been adapted to make it smoother and less time-consuming. Some things students felt they learned included "the basics of public key encryption", "public and private maps", "different types of codes and how to solve them", "there are many ways to encrypt messages" and "the frequency attack and to use it". The teacher noted that the lesson contained more content than could be covered in an hour, but they would use the lesson again.

Teacher feedback

One teacher gave feedback on *Cryptography 1*, and they taught the lesson in a mixed school. They found that their students enjoyed the class, although the substitution cipher became boring quickly for the students. They also felt that their students learned the importance of computer security which students didn't know much about.

Searching & Sorting

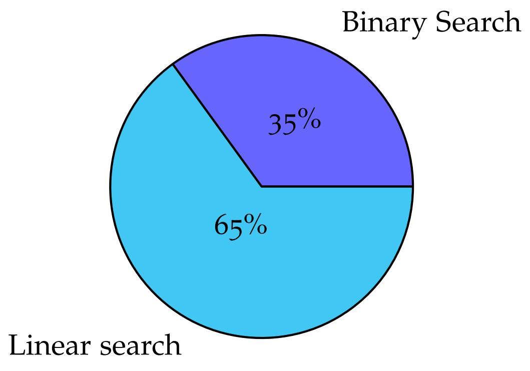
It should be noted that some teachers combined this class with a lesson on Algorithms which can be seen from their responses to the feedback forms. However, as they filled out their feedback on the Algorithms form the quantitative results described in Table 6.5 don't include those values, instead they are presented in Table 6.3. Qualitative responses that refer to the Searching and Sorting lesson activities may have been included in this section below. As shown in Table 6.5 this class was well received with students giving it a rating of 3.43 out of 5 in terms of their enjoyment levels.

Table 6.5: Searching and Sorting - High-level Feedback

No. of Students	Did you enjoy this class? Average rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
23	3.43	20 (87%)	0 (0%)

Students also felt that they learned a lot in this class, with 87% saying they felt they learned something and no students saying they didn't learn anything. This is, however, not well reflected in the responses to various multiple-choice questions presented in Figure 6.12, Figure 6.13 and Figure 6.14. This could be down to the fact that all of these ideas (Binary search, sorting algorithms etc.) are new to the students and one lesson isn't sufficient to learn them.

Figure 6.12: Which of these is the best way to search in an ordered data set?



Following are some quotes from students on what they enjoyed or didn't enjoy about the class:

- "I enjoyed the fact that I discovered different ways to look at a set of numbers or words, numerically, etc" (Female, 15)
- "The game we played and the group questions" (Male, 15)
- "The game we played and the group questions" (Female, 15)
- "It was interesting/the problem with the arrows" (Female, 15)

Figure 6.13: Which of these is NOT important when sorting data?

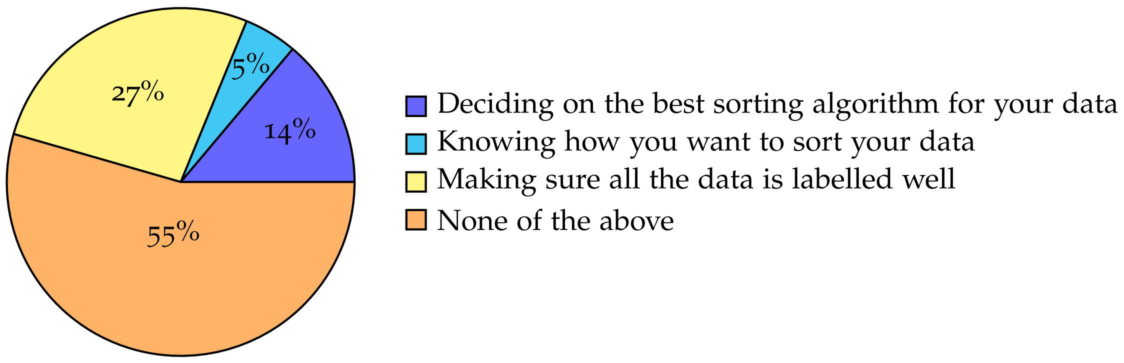
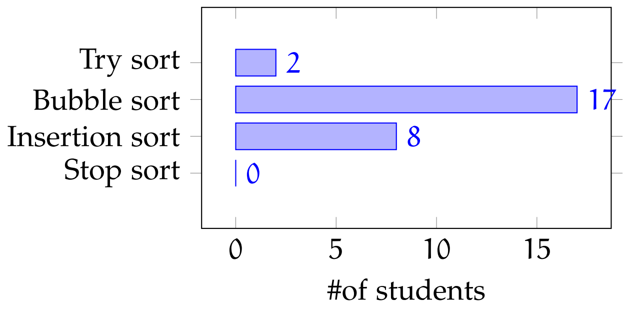


Figure 6.14: Which of these is a sorting algorithm we covered in class? (select all the apply)



- "Guessing the archery question" (Female, 15)
- "Was difficult" (Male, 15)

As shown in Table 6.5 a very high percentage of the students (87%) felt that they learned something from this class, with no students answering "no" to that question, which is very encouraging. We will now present some quotes of what students felt they learned:

- "Creative thinking to sort the question" (Female, 16)
- "How to sort things quicker and more efficiently" (Male, 15)
- "Linear and binary searches" (Male, 15)

Teacher feedback

No teachers taught this class other than the first author, so no feedback was collected.

Other lessons

The following lessons, presented in Table 6.6 were added at various points throughout the year, this is the main reason for the lack of use and the lack of feedback on them. For convenience these lessons have been grouped together, although the data presented is mainly about one lesson, Programming Concepts.

Table 6.6: Other Lessons - High-level Feedback

Lesson	No. of Students	Did you enjoy this class? Avg rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
Programming Concepts*	24	3.75	15 (63%)	4 (17%)
Graphs	0	N/A	N/A	N/A
Finite State Machines	2	1	0 (0%)	0 (0%)
Data & Binary	0	N/A	N/A	N/A

*includes both Variables & User Input as well as Loops & Conditionals – these were originally one lesson

As shown in Table 6.6 *Programming Concepts* was a well-received class and over 60% of students felt like they learned something from the class. Below is some feedback from students on what they did and didn't like about the classes as well as what they learned.

- "Making a game" (Male, 15)
- "I liked the games and decision trees" (Female, 15)
- "Conditional Thinking" (Male, 15)
- "I was bored" (Male, 15)
- "I learned that games are made just like a computer program using variables, inputs, conditionals and loops" (Female, 15)

- "What a decision tree is" (Female, 15)
- "I learnt about variable, input, loops and conditional" (Female, 16)

Teacher feedback

The only lesson with sufficient input to be looked at (*Programming Concepts*) was taught by the first author so no feedback was gathered on any of these classes.

6.2.2 SCRATCH

The Scratch lessons were designed to not only introduce students to programming concepts such as loops and variables, but also introduce them incrementally to all the functionality in Scratch 2.0. The lessons, in their current form, are tutorials which are self-led but can also be demonstrated by teachers if this suits their class. The first tutorial is designed as a basic introduction and should be completed first. The following three can be done in any order. These subsequent lessons have a few tiered tutorials; depending on the confidence and level of the students, teachers can give students a step-by-step guide, or a more scaled back tutorial with requirements and hints.

The Scratch feedback forms differed from those given for other modules of the course. Students were asked questions such as how long they took to complete the exercise, how much help they needed, did they find the tutorial helpful as well as whether they liked the exercise or found it hard. A sample of the form can be found here: `https://bit.ly/2MEprbP`.

Cat and Mouse

Cat and Mouse is recommended as the first tutorial that students should do in Scratch. It gives a basic introduction to the Scratch interface and gives students the tools to make basics games. These skills can then be advanced in future tutorials or in their own projects and work. The tutorial sheet goes through the project step-by-step with a few opportunities for students to discover functionality for themselves, but in general it is very guided. The tutorial sheet and teachers guide can be found in Appendix D.

From the data presented in Table 6.7 it can be seen that students generally enjoyed this lesson and found it easier than the subsequent tutorials. As it is meant to be an introduction that is good to see, alongside the fact that 65% of students completed it in an hour or less, with over 90% completing it in under two hours as presented in Figure 6.15. Following are some quotes about what students enjoyed about this lesson:

Table 6.7: Cat and Mouse - High-level Feedback

No. of students	Did you enjoy this exercise? Avg rating (1-5)	Did you find the exercise difficult (5) or easy (1)?	Was the tutorial easy to follow? (1-5)
54	3.24	2.52	2.72

- "It was quite fun to try something new like coding and making your own game" (Male, 15)
- "I liked how I was let figure out the way to do things myself" (Female, 15)
- "I got to be creative and make my own game" (Male, 15)
- "Learning how Scratch works" (Male, 15)
- "I found it interesting" (Male, 16)

Some comments on difficulties or issues students found:

- "I found it kind of hard until my classmates helped me" (Female, 16)
- "It was boring because I have used Scratch previously and I find it easy" (Male, 15)
- "It was hard on the slow computer" (Female, 15)

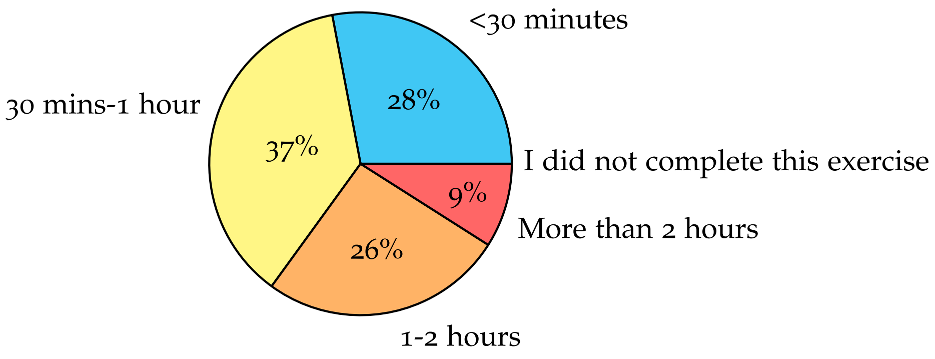
When asked *"Did you need lots of extra help, was it too easy/boring?"* most students responded with variations of no, though there was a mixture of whether people found it easy or hard, as reflected by the 2.52 rating presented in Table 6.7.

Some suggestions were made on ways to make the tutorial or game better/easier to understand including "more pictures of what the code should look like", "video tutorials" and "making the instructions clearer".

Teacher feedback

Two teachers gave feedback on this Scratch lesson, they both taught the lesson in a mixed class. Both teachers found it took about 5-6 sessions (5-6 hours) to complete the activity, which is more than was anticipated. Both teachers felt that students generally enjoyed it, although some who had used Scratch before found it more boring. Of note were the creative aspects of it as well as the sense of achievement when completing a working game. Both teachers felt their students learned something, with it being their first experience in some cases of programming and Scratch. Both teachers said

Figure 6.15: Roughly how long did it take you to complete this exercise?



they would use the class again with one commenting that it was a "nice introduction to Scratch".

Pong

Pong is a classic arcade game and this tutorial allows students to make a version of this in Scratch. It expands on the knowledge gained from Cat and Mouse and allows students to learn more about Scratch's functionality. There are two versions of the tutorial for Pong: one goes through the project step-by-step, the other, designed for stronger students, gives a general overview of what is needed for each section of the game. The tutorial sheets and teachers guide can be found in Appendix D.

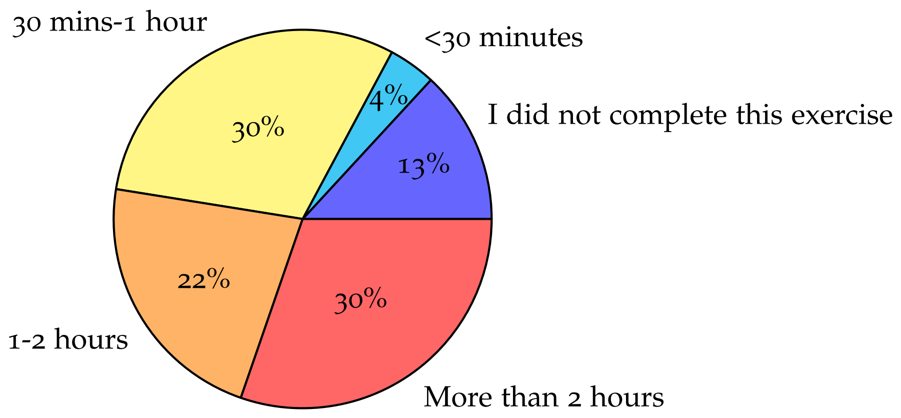
It can be seen from Table 6.8 as well as from Figure 6.16 that students found this tutorial more difficult than *Cat & Mouse*. This is also reflected in both how much they liked the lesson (2.96 out of 5) and how easy the tutorial was to use (3.09), shown in Table 6.8. This is further backed up by the fact that 30% of the students took more than two hours to complete the tutorial and 13% did not finish it as shown in Figure 6.16.

Table 6.8: Pong - High-level Feedback

No. of students	Did you enjoy this exercise? Avg rating (1-5)	Did you find the exercise difficult (5) or easy (1)?	Was the tutorial easy to follow? (1-5)
23	2.96	3.48	3.09

When it comes to what students did/didn't enjoy there is a mixed response. Some students enjoyed it saying things such as they enjoyed "trying to problem solve" or that it "allows me to gain a head start for future experience like this". Others found it "hard and confusing" and others commented on the game "glitching". Students also needed more help than the previous exercise, with many finding Scratch "boring". Others however found it "fun" and "easy". Comments about improvements again contained suggestions of video tutorials and to describe the scripts in more detail.

Figure 6.16: Roughly how long did it take you to complete this exercise?



Teacher feedback

One teacher gave feedback on this lesson, they taught in an all-boys school. They spent five 40-minute classes on the lesson which, like *Cat & Mouse*, was longer than expected. The feedback they felt from the class was mixed, with some enjoying the challenge, others enjoying being left to use Scratch as they liked whilst others found it frustrating. Even with this mix though they felt their students learned problem solving and logical thinking from the exercise. One comment was that students struggled to read the descriptions, a recommendation being that the teacher would have access to the whole code and explanations and just explain bits as they go along. This works fine in general and could be a good way to run the class, however the tutorial is designed to be self-led and so more detail is required to compete the exercise independently – this differential learning was highlighted by the teacher as a big plus to the lesson. We encourage teachers to adapt all the resources to their context and students. One thing mentioned by both students and teachers could be introducing video tutorials; this is something we will consider in the future.

Polygon Drawer

The Polygon Drawer tutorial allows students to learn more about loops in programming by drawing different patterns of polygon shapes. There are two versions of the tutorial for Polygon Drawer: one goes through the project step-by-step, the other, designed for stronger students, gives a general overview of what is needed for each section of the game. The design of the tutorial allows students to incrementally make the project more complex. The tutorial sheets and teachers guide can be found in Appendix D.

As shown in Table 6.9 this lesson was given an average rating of 3 out of 5 in terms of students enjoyment levels. Across the three Scratch lessons described so far they all received a very similar rating with a range from 2.96 to 3.24 out of 5. This lesson appears to be the easiest for students with the lowest rating of 2.42. This is backed up not just by the low rating but also the fact that 76% of students finished it in under an hour as shown in Figure 6.17. This could be linked to the fact that students have already completed (at least) two Scratch lessons. Students found the tutorial mildly easy to follow, it may be that more help is required or perhaps slides for teachers to walk their students through the exercise.

Table 6.9: Polygon Drawer - High-level Feedback

No. of students	Did you enjoy this exercise? Avg rating (1-5)	Did you find the exercise difficult (5) or easy (1)?	Was the tutorial easy to follow? (1-5)
38	3	2.42	2.5

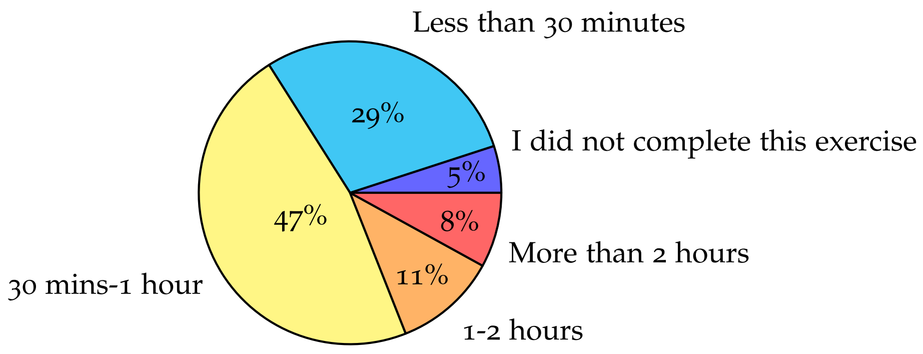
Comments of what students did and did not like varied greatly as with the previous lessons. Some of the quotes received from students include:

- "I enjoyed making all the different shapes but thought it was a bit simple" (Female, 15)
- "It was actually fun making this game. Much better than the Cat & Mouse game" (Male, 15)
- "It was boring" (Male, 15)
- "Experimenting with the different ways to program" (Male, 15)
- "It was cool to see what shapes I could make, I was surprised that it was cool with all the shapes" (Male, 16)
- "It was confusing" (Female, 15)

From reviewing their answers it is clear that less extra help was required for this tutorial compared to for the *Pong* lesson, which again is supported by the lower 2.5 (out of 5) rating for how easy the tutorial was to understand/follow.

Suggestions for improvements from the students include more pictures, clearer wording and colours. All tutorial sheets are available in colour, so this may have just been down to the teacher or printer availability. Interestingly one student suggested to "give less advice". The design of the Polygon drawer is an incrementally built program and there are two tutorial sheets available. One gives a step-by-step guide whilst the other gives mainly hints and tips and allows students to engage in discovery, self-led learning. The hope is to push stronger students (or those with previous experience) whilst scaffolding learning for those who are struggling or familiarising themselves with Scratch.

Figure 6.17: Roughly how long did it take you to complete this exercise?



Teacher feedback

No teacher feedback was received on this lesson.

Guessing game

No students undertook the *Guessing game* lesson this year. Subsequently there is no student or teacher feedback for this lesson.

6.2.3 PYTHON

The Python course was built around Python Turtle, a graphical language built in Python which is based on the famous Logo programming language. We felt that this would be more engaging, especially for TY students, as the output would be very visual, rather than the usual numbers and sentences which are the standard output for novice programmers. The module was broken up into lessons and labs. The idea was that teachers would guide students through the lessons using the provided slides and code examples, alongside live coding if they were comfortable doing that. Once the idea, concepts or structures were introduced (for example, loops) the students would then complete several exercises which built upon the previously covered content and added this new concept/idea into the problems.

Python feedback forms differed from those in other modules of the course. Students were asked whether they felt they learned something, enjoyed the class as well as how many questions they completed on the exercise/lab sheet. An example form can be found here: `https://bit.ly/2ws71QA`.

Only one class group provided feedback on the Python lessons, although it is known that others used it during the year and found it a helpful resource. Due to time constraints the class that gave feedback started with Lab 3, which is based on the Python Turtle class as described in Section 3.4.3.8. From both the data presented in Table 6.10 and from classroom observations the lessons were well received, with students giving the lesson a rating of 3.9 out of 5 in terms of their enjoyment levels.

Table 6.10: Python - High-level Feedback

Class	No. of students	Did you enjoy this exercise? Average rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
Lab 1	1	5	1 (100%)	0 (0%)
Lab 2	0	N/A	N/A	N/A
Lab 3	20	3.9	16 (80%)	1 (5%)
Lab 4	0	N/A	N/A	N/A
Lab 5	0	N/A	N/A	N/A
Lab 6	0	N/A	N/A	N/A

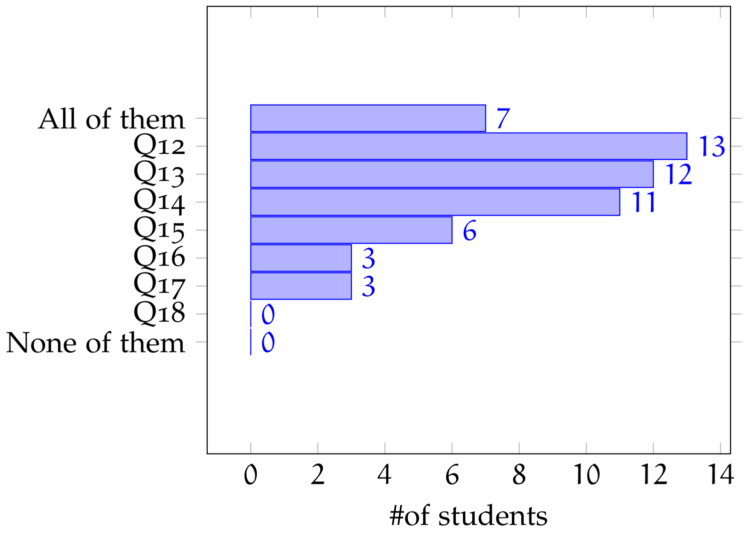
Lab 1

There was only one student who completed this lab sheet. She had previously completed Lab 3 and moved back to start on Lab 1 as directed by the teacher. She enjoyed "learning how to use python and how to code" and that she learned "how to use simple python".

Lab 3

One of the advantages of the designed lab sheets and lessons is that students can work independently and at their own pace. In Figure 6.18 we can see that the number of students who completed each lesson steadily decreases. This trend is to be expected as students work at their own pace and deal with the problems on their own with assistance from their teacher and peers.

Figure 6.18: What questions did you complete?



As can be seen in Table 6.10 students generally enjoyed this class, and we now present some quotes from the students who provided feedback:

- "I love coding in general and I liked learning a more complicated coding language other than scratch." – (Male, 15)
- "It's fun!!!!!!!" – (Female, 16)
- "I really liked being able to code and use variables." – (Female, 16)
- "the python coding and using turtle" – (Female, 15)
- "difficult" – (Female, 15)
- "it was alright just boring"- (Male, 15)

Students also felt they learned something during the class, which is to be expected as most had never programmed before. Below are some quotes from students on what they felt they learned.

- "how to use simple python" – (Female, 15)

- "I learn how to do some simple computer program" – (Female, 16)
- "The basics of Python" – (Male, 15)
- "I learned how to create a screen and change the background colours" – (Female, 15)

6.2.4 WEB DEVELOPMENT

The web development feedback form was like those given during the *Introduction & Concepts* module. The feedback form used can be found here: `https://bit.ly/2LEnczz`. Only one class of 26 students used the *Making a basic website* lesson plan. From Table 6.11 we can see that it wasn't well received and only 42% of students felt they learned something. This is a disappointing result, but we strongly believe that this module is worthwhile, and more work is required to make the tutorials and lessons more engaging for students. Similar tutorials to this lesson have been used at Maynooth's annual summer camp and have always been a popular lesson.

Table 6.11: Web Development - High-level Feedback

Class	No. of students	Did you enjoy this class? Avg rating (1-5)	No. of students who said they learned something	No. of students who said they didn't learn something
Making a basic website	26	2.31	11 (42%)	7 (27%)
Databases	0	N/A	N/A	N/A

Some comments made by students include that they found it "hard" and "challenging" as well as "hard to understand". Other students felt there "wasn't as much to do as I first thought" and that "it wasn't working most of the time". On the plus side some commented that they "found it challenging, which made it interesting" and they "liked to coding part".

The majority of students who felt they learned something gave a variant of "how to make a website" when asked what they learned, along with some stating that "learning more about HTML than they already knew". Those who didn't feel like they learnt anything put this down to "not understanding", "I already knew what to do" and that "the code wasn't working". These final comments will lead to further testing of this any code given to teachers, although the method and code presented in the Web Development lesson was working as of this publication on Mozilla Thimble, the recommended tool.

6.3 overall course assessment

6.3.1 views of computer science survey

As previously described in Chapter 4, students completed a survey that investigated their views of CS. Due to the nature of this study, it is hard to take any real results from this except for one class taught directly by the author. We will first present all the respondents' data and then look at this specific subset in more detail.

Table 6.12: All Students - View of CS Feedback - Q1-3 & 13 - Survey 1

Survey 1 (n=237)			
Question	Yes	No	Maybe
Q1. Would you/have you considered studying CS in college/university?	42 (17.7%)	127 (53.6%)	68 (28.7%)
Q2. Is CS interesting to you?	65 (27.4%)	86 (36.3%)	86 (36.3%)
Q3. Is CS challenging?	62 (26.1%)	22 (9.3%)	153 (64.6%)
Q13. Have you heard the term CT?	34 (14.3%)	203 (85.7%)	N/A

Numbers listed are the total number of responses for each option

In Tables 6.12 and 6.13 the results are presented for Q1, Q2, Q3 and Q13. These questions have been separated from the rest for ease of reading. These questions had Yes, No or Maybe responses, except for Q13 which was just Yes or No. The percentage of students who have heard the term CT rises from 14.3% in survey 1 to 50.7% in survey 2. As one of the goals of CS2Go is to introduce students to CT this is encouraging, although we would hope to have that percentage increase even further. The other significant shift is Q3, with more students finding CS challenging in survey 2 than they did in survey 1 (43.1% compared to 26.1%). We believe this is down to students not knowing what CS entails, which can be seen by the high percentage of students who said Maybe in survey 1 (64.6%).

Table 6.13: All Students - View of CS Feedback - Q1-3 & 13 - Survey 2

Survey 1 (n=237)			
Question	Yes	No	Maybe
Q1. Would you/have you considered studying CS in college/university?	13 (18%)	42 (58.3%)	17 (23.6%)
Q2. Is CS interesting to you?	19 (26.4%)	27 (37.5%)	26 (36.1%)
Q3. Is CS challenging?	31 (43.1%)	9 (12.5%)	32 (44.4%)
Q13. Have you heard the term CT?	37 (50.7%)	35 (49.3%)	N/A

Numbers listed are the total number of responses for each option

In Table 6.14 the results of Q4-Q12 from all students who completed the view of CS survey either at the beginning or the end (or both) of the study are presented. All of these results can only suggest trends, even those that are statistically significant, as the sample sizes differ so greatly (237 to 72) and, as discussed in Section 6.4.1, some students took the surveys after not using much of the CS2Go content.

There are a number of changes observed in the answers from survey 1 to survey 2 which are interesting. Students view of Q4 and Q5 (Using the internet/Using Word, Excel etc. is central to CS) reduced from 3.536/5 to 3.306/5 (p-value = 0.091). This shift towards more strongly disagreeing with these statements make sense as students were probably less familiar with what CS is before they began the lessons. The same trend was seen for Q6 although not to a significant level.

Another shift which was to be expected was in Q12 which asks students whether "Work in CS can be done without a computer". Students responses shifted from an average of 2.215/5 to 2.819/5 (p-value = 0.003). This is to be expected as many of the lessons in CS2Go are unplugged in nature and so aim to show CS isn't only about using a computer, and that the concepts can be taught in interactive lessons.

One shift that is interesting and more unexpected is in Q11, which asks whether "Boys/men are more likely to study CS than girls/women". Students views shifted to more strongly agreeing with this statement, from 2.574/5 to 2.986/5 (p-value = 0.05). This is discouraging as one of the hopes of CS2Go is to remove stereotypes about CS being a male-only subject. However, it could just be that those who didn't complete survey 2 disagreed more strongly to begin with in survey 1, a larger and more consistent sample is needed to see this impact.

Table 6.14: All Students - View of CS Feedback Q4 - Q12

Question	Avg Survey 1	Avg Survey 2	T-Test
Q4. Using the internet is central to CS	3.536	3.306	T-Score = 1.695 P-Value = 0.091
Q5. Using Word, Excel etc. is central to CS	3.127	2.847	T-Score = 1.649 P-Value = 0.101
Q6. Installing software (e.g. Windows, iTunes) is central to CS	3.253	3.097	T-Score = 0.817 P-Value = 0.415
Q7. Programming is central to CS	3.911	3.778	T-Score = 0.714 P-Value = 0.476
Q8. Being able to solve different problems is central to CS	3.895	4.083	T-Score = 1.068 P-Value = 0.287
Q9. CS is an area related to maths	3.473	3.528	T-Score = 0.322 P-Value = 0.748
Q10. A computer scientist should be good at working with others	3.109	3.181	T-Score = 0.391 P-Value = 0.696
Q11. Boys/men are more likely to study CS than girls/women	2.574	2.986	T-Score = 1.976 P-Value = 0.05
Q12. Work in CS can be done without a computer	2.215	2.819	T-Score = 3.037 P-Value = 0.003

All questions were rated on a Likert scale from 1-5 with 1 being strongly disagree and 5 being strongly agree.
n = 237 for Survey 1 and n = 73 for Survey 2.

Case study on one class

Due to the issues previously discussed and discussed further in Section 6.4.1 the numbers of students filling out the Views of CS survey at the beginning of the course and the end of the course differed greatly. This leads to us not being able to compare the results from the two very accurately. Also, due to other issues discussed, many students didn't get exposure to a significant amount of the course, or else they used lessons but didn't take the Views survey until much later in the school year. This again leaves us unable to conclude much from the surveys regarding how the lessons impacted students' views. To combat this issue, we can look at one cohort of students which were taught by the author of this thesis from September 2017 to January 2018, with 10 hours in total being delivered. These students took the first survey in the first class and took the second survey after concluding the lessons. Table 6.15 and Table 6.16 present the results for the two surveys along with T-Tests showing the change from survey one to two.

The results from Q1, Q2, Q3 and Q13 shown in Table 6.15 offer mixed results. The clear increase in those who have heard the term Computational Thinking, from 20% to 67%, is encouraging. The five who hadn't heard the term after the lessons could have forgotten, or been absent, during the lessons the term was mentioned. We don't believe it is necessary to repeatedly mention CT in every lesson, the lesson focussed on the topic is designed to give students an overview of what the term mean and why it's a helpful skill. The other lessons are designed to teach CS topics whilst also improving students CT skills through problem-solving exercises, team activities and other challenges and so explicitly stating the skills they're learning during the lessons would be unnecessary.

The number of students answering *maybe* to whether CS is interesting (Q2) or challenging (Q3) has decreased after the lessons as shown in Table 6.15. For Q2 it has decreased from 33% to 27% and for Q3 it has gone from 80% down to 40%. One of the issues with CS courses in Irish universities and colleges is that many students have misconceived ideas of what CS is and is not. Introducing students to the topic earlier will allow them to make informed decisions about whether they enjoy the subject or not. This can be seen in the results of Q1 which again has less students responding *maybe* to this question. Encouragingly more students (3 in survey 1, 5 in survey 2) responded that they would or had considered studying CS further.

Due to the small sample size, many of these results aren't significant at the usual level (p = 0.05), however, some are of interest as they provoke further areas of study to be investigated. Students views for Q4 , Q5 and Q6 in Table 6.16 shifted towards

Table 6.15: Case Study Class View of CS Feedback - Q1-3 & 13

Question	No. of Yes	No. of No	No. of Maybe
Q1. Would you/have you considered studying CS in college/university?			
Survey 1	3 (20%)	7 (47%)	5 (33%)
Survey 2	5 (33%)	6 (40%)	4 (27%)
Q2. Is CS interesting to you?			
Survey 1	6 (40%)	2 (13%)	7 (47%)
Survey 2	6 (40%)	3 (20%)	6 (40%)
Q3. Is CS challenging?			
Survey 1	3 (20%)	0	12 (80%)
Survey 2	8 (53%)	1 (7%)	6 (40%)
Q13. Have you heard the term CT?			
Survey 1	3 (20%)	12 (80%)	N/A
Survey 2	10 (67%)	5 (33%)	N/A

Q1, Q2 and Q3 all had three responses: Yes, No, Maybe.
Q13 had two: Yes, No.
Both surveys has 15 respondents.

disagreeing with the statements. This makes sense as prior to the lessons many students would not have known what CS is, so terms like the Internet, Word/Excel and installing software could have been associated strongly with CS in student's minds. After being introduced to what CS is, different topics and some programming, students disagreed more strongly with those statements.

Students views shifted for Q8 which stated that solving problems is central to CS. Students agreed more strongly with this statement after the lessons (p-value = 0.136). This is encouraging as it is one of the main aims of the course, to promote problem solving and computational thinking for students. This is further backed up by the

Table 6.16: Case Study Class - View of CS Feedback Q4 - Q12

Question	Avg Survey 1	Avg Survey 2	T-Test

Q4. Using the internet is central to CS	3.6	3	T-Score = 1.598
			P-value = 0.121

Q5. Using Word, Excel etc. is central to CS	3.067	2.4	T-Score = 1.876
			P-value = 0.071

Q6. Installing software (e.g. Windows, iTunes) is central to CS	3.2	2.533	T-Score = 1.607
			P-value = 0.119

Q7. Programming is central to CS	4.067	3.733	T-Score = 0.915
			P-value = 0.368

Q8. Being able to solve different problems is central to CS	4.267	4.667	T-Score = 1.535
			P-value = 0.136

Q9. CS is an area related to maths	4.067	3.4	T-Score = 2.534
			P-value = 0.017

Q10. A computer scientist should be good at working with others	3.2	3.733	T-Score = 1.600
			P-value = 0.121

Q11. Boys/men are more likely to study CS than girls/women	2.733	2.867	T-Score = 0.328
			P-value = 0.745

Q12. Work in CS can be done without a computer	2.267	3.4	T-Score = 2.646
			P-value = 0.013

All questions were rated on a Likert scale from 1-5 with 1 being strongly disagree and 5 being strongly agree. n = 15 for all questions.

increase of those students who had heard Computational Thinking (Q13 Table 6.15) with this figure increasing from 20% up to 67%.

Most of the lessons (about 70%) that were taught to the students were from the *Computer Science Concepts* module, which is mainly unplugged, team-based exercises and discussions. This can be seen in the results from these surveys as students agreed more strongly with Q10 and Q12. Q10 refers to team-work being important for computer scientists, and students agreed more strongly that this is true after the lessons (p-value = 0.121). Q12 refers to work in CS being done without a PC, which students agreed more strongly that it is possible (p-value = 0.013) in the second survey. These two results combined can show that the lessons had an impact on students, and when linked with the results of Q8 discussed previously, give evidence that unplugged and similar style lessons can be very beneficial to students learning CS; it does not always have to be about programming on a computer.

A final result of note is in relation to the results of Q9, which refers to CS being an area related to maths. One of the major hang-ups for students studying CS at third level is the assumption that they must be good at mathematics to study CS. Whilst the two are related, and examples in the lessons we have developed clearly emphasise some links, it is not necessarily true that one must be a strong mathematical student to excel at CS. Removing these misconceptions before students get to college may lead to greater uptake in the subject at a time when there is a need and a desire for more CS graduates.

6.3.2 computational thinking assessment

The Computational Thinking Assessment, as described in Chapter 4 was also used during the year. In this section we will discuss the results from the students who took either assessment. We will refer to these as Test 1 and Test 2 throughout this section.

All respondents

Table 6.17 presents results from all respondents who took either of the Computational Thinking Assessments. We have split the results into two groups; in the middle column are all students who took each test. In the right-most column are students who took both tests. It can be seen that students in both groups did slightly better in the second test than the first test. For the entire population this was a rise from 5.806 to 6.627 out of 13. This is a difference of 0.821 and this is a significant difference (t-score = 2.473, p-value = 0.014). For those who took both tests there is no significant difference (t-score = 0.159, p-value = 0.873) with their scores rising from 6.527 to 6.6. Due to the large difference in sample size it is hard to glean any insights from these results.

Table 6.17: School results for the Computational Thinking Assessments

	Avg of those who took at least one test	Avg of those who took both tests
Test 1	5.806 (n = 187)	6.527 (n = 55)
Test 2	6.627 (n = 76)	6.6 (n = 55)

Control group vs study group data

One of the important parts of any study with students and their learning from lessons is to try and show that the lessons or intervention is actually the reason for changes or improvements in students performance is whatever task they undertake. For our study, we had a pool of students who took the various assessments for CS2Go and were taught the lessons by their teachers, for the purposes of this section we will call these students the "study group". Alongside this we arranged to have a "control group" of students who would complete the Computational Thinking assessment but not be taught from the CS2Go content. Two Transition Year classes from different schools took the Computational Thinking Assessment. One took the first assessment in September and the second in April time, the other was administered by us during our teaching block as described in Section 6.3.1.1. In this section we divide the results up based on those in the control group versus the study group.

In Table 6.18 we can see an increase in both groups from Test 1 to Test 2. As with the previous section there is not much we can see from this data as the sample sizes are so vastly different.

Table 6.18: Control group vs Study group results

	Average of study students	Average of control students
Test 1	5.696 (n = 148)	6.237 (n = 38)
Test 2	6.683 (n = 41)	6.897 (n = 29)

Looking at the students from both the study and control classes who took both tests in Table 6.19, we can see that there is in fact a dip in the study groups scores from Test 1 to Test 2, whereas there is an increase in the control group. The sample sizes are again small, and as previously discussed some of these students will have taken the test weeks or months after any lessons so we can't read into the results much. However, the 19 students in the control group did perform better in Test 2 than Test 1 as seen in Table 6.19.

Table 6.19: Control group vs Study group results (Those who took both tests)

	Average of study group	Average of control group
Test 1	6.806 (n = 36)	6 (n = 19)
Test 2	6.667 (n = 36)	6.474 (n = 19)

We hypothesise that this increase could show the general progression we would expect in a year. It can be generally assumed that a student's CT or problem-solving skills should improve as they go through a year of education and general life experience. CT is a cross-disciplinary skill and can be helpful and used in many areas of life, so one would hope to improve this skill over time from a variety of different sources. This needs to be verified further and with a substantially larger sample in a more controlled environment, but it is a starting point.

Using this information we can calculate an "expected gain" value for students through a year.

Expected gain (based on control group): 6.474 – 6 = 0.474

0.474/6 = 7.9% increase

We can see that based on the 19 control group members who took both surveys we have an expected gain of 7.9%.

This is in contrast to the study group who declined by 2% as shown below.

Gain in study group: 6.667 – 6.806 = -0.139

-0.139/6.806 = 2% decrease

Case study school - control group vs study group

As previously described in Section 6.3.1.1, one group of students taught by the author were taught the lessons for a sustained period, with assessment forms being administered prior to and after the teaching. The class taught comprised approximately half of the Transition Year group in the school. These students were administered the views of CS survey and the Computational Thinking Assessment at the same times as the class being taught using CS2Go lessons. This allows us to directly compare two groups receiving similar teaching throughout the year.

As can be seen from Table 6.20 the whole year group on average performed similarly across each test, with a slight decline from Test 1 (7.053) to Test 2 (7) (T-Score = 0.090, P-Value = 0.928). However, the interesting thing to note in this data is that the study class improved significantly from 7.25 to 8.294 (T-Score = 1.288, P-Value = 0.206). Students in the control group did worse in the second test when compared to the first test dropping from 6.833 to 5.667 (T-Score = 1.505, P-Value = 0.144). This could be an isolated occurrence caused by the drop in participants (33% less students took Test 2) and also the lack of motivation for students in the control class to really try at the questions the second time around. The first one could have been something interesting and different but by the second one they would have been aware they had nothing to gain from completing it.

Table 6.20: Same school - Control group v class

	Average of whole year group	Average of study class	Average of control class
Test 1	7.053 (n = 38)	7.25 (n = 20)	6.833 (n = 17)
Test 2	7 (n = 32)	8.294 (n = 17)	5.667 (n = 12)

The encouraging thing from this though is that a group of students who were introduced to CS, CT and programming through the CS2Go content showed an increase from Test 1 to Test 2. This is extremely encouraging and with further, wider-scale, studies we hope to confirm that CS2Go can be used to improve students CT skills.

Gain in study group: 8.294 – 7.25 = 1.044

1.044/7.25 = 14.4%

Using the previously used calculation we can see that the gain in the study class is 14.4%. This is significantly higher than the 7.9% calculated from the control group as described in Section 6.3.2.2. This is encouraging and could show that the CS2Go content did in fact improve students CT skills.

These findings are reflected when looking at the students from both the study and control classes who took both tests, presented in Table 6.21.

The study class improved on average, though not to a significant level (T-Score = 0.807, P-Value = 0.426), although more than our "expected gain".

Gain in study class: 7.933 – 7.2 = 0.733/7.2 = 10.18%

Table 6.21: Same school - Control group v class (those who took both tests)

	Average of study group	Average of control group
Test 1	7.2 (n = 15)	7 (n = 9)
Test 2	7.933 (n = 15)	6.333 (n = 9)

The control group dropped from Test 1 to Test 2, again not to a significant level (T-Score = 0.676, P-Value = 0.509). Of note is that the study class did significantly better in test 2 than their control group peers (T-Score = 1.789, P-Value = 0.087).

6.4 discussion

6.4.1 issues with schools teaching the course

One of the main downfalls of this study, which we have stated previously, is that classes were, for the most part, not done in any order or with any structure. The hope of CS2Go is that it will supplement the teachers' other resources and work to achieve the outcomes described in the curriculum they are using. The flexibility and adaptability of the lessons are a major positive and as discussed in the teacher feedback, many teachers cut short lessons or combined lessons if they felt that was needed. Other teachers also supplemented the classes with other material. One future hope is for the newly developed website to become a place where educators can share other resources and ideas for the lessons and topics covered by CS2Go. For a study of the courses effectiveness on CT skills and students views of CS though, a more robust and controlled sample is needed. As discussed in Section 6.5.1 the hope is to do this in the future.

6.4.2 class reaction

It can be seen in the results discussed in Section 6.2 that, in general, the classes were well received by students and a large percentage felt they learned something from the lessons. This in itself is an encouraging finding and with slight adaptations and adding further content, we believe CS2Go can be a really beneficial tool for schools and educators in Ireland and further afield. Most lessons received a rating above 3 out of 5, which when considering most students general apathy towards school, along with the variety of contexts covered by the schools, is very encouraging. With further training we believe teachers will be able to use these lessons to introduce students to CS in a fun and engaging way. With the introduction of CS to the Leaving Certificate course and the expansion of the Junior Cycle Coding Short course, the opportunity to equip teachers who are passionate and excited about CS is one we hope to continue to do.

6.4.3 APPEARS TO IMPACT STUDENTS VIEWS OF CS

From the one structured group of students' (see Section 6.3.1.1), it can be seen that student's views of what CS is and what is central to CS are impacted. This is a positive outcome as correcting and informing students of the misconceptions surrounding CS is one of the hopes of this course. Introducing students to CS earlier can only benefit them as the skills acquired will be helpful in many walks of life, but it also allows students to know whether CS is a subject they wish to pursue further. Computer Science third-level courses have a very high failure and drop-out rate in Ireland [77]. We believe this is due to students not understanding the core concepts of what CS is, and so when introduced to them in third level they are surprised and disengage. Showing these concepts earlier, in a less pressurised environment which Transition Year allows, could help students make informed decisions.

6.4.4 APPEARS TO IMPACT STUDENT'S CT SKILLS

As discussed in Section 6.3.2.3, promising results were shown in the small sample group in the Computational Thinking Assessment. Students' scores increased by the end of the course. This was only a small sample and the larger groups give a mixed view of whether this could be true of wider groups. One of the hopes of a more structured study (see Chapter 8) is to see whether or not CS2Go does improve students CT skills. It should be noted that in a separate study, described in Chapter 7 run with first year undergraduate students, we found that their scores did increase over the course of a year studying CS as part of their degree. This alongside the group of the small sample described here encourage us that CS in general, and by extension CS2Go, could have a positive impact on students' CT skills.

6.5 FUTURE WORK

6.5.1 STRUCTURED STUDY

One of the major issues with this study is that students weren't consistently taught classes from this course. Teachers are encouraged to pick and choose what suits them and their contexts, and this flexibility is offered by the structure of CS2Go. However, to gauge both its impact on student's learning, CT skills, views of CS and teachers view of the course, a more structured and rounded study is required. The hope is that in the next academic year we will run a more controlled study with a smaller number of schools. Several teachers will be contacted, and these will form a group which we will be able to keep track of and equip to teach the course over a set time. The teachers will be given a guided timetable to follow, which will allow some flexibility but will allow all students to be introduced to the basics of CS, programming. The plan is that we can then see how the lessons impact the various metrics we are assessing.

6.5.2 EXPAND ON COURSE

As with any course there are always opportunities to expand. We will continue to do this throughout the next year, improving the existing lesson plans as well as developing new content. One area we hope to focus is on the web development and app development sections of CS2Go. This could include using App Inventor as well as lessons on digital citizenship and online safety. There is also a lot of potential to expand both the Python and Scratch modules.

6.5.3 TEST THE COURSE WITH OTHER YEAR GROUPS

CS2Go is aimed at Transition Year students. However, there is no reason the lessons can't be used by younger and older students. We have already been contacted by several teachers who wish to use the content for the new Leaving Certificate course. This is great to hear that teachers see it as useful for the new course and an area we hope to expand on in the future. The Junior Certificate Short Course in Coding is also an area we hope to promote the use of CS2Go.

6.5.4 video tutorials and demonstrations

One major improvement suggested by both students and teachers is video tutorials of the Scratch exercises specifically. This is an area we would hope to engage with and develop resources to help both students and teachers. We also feel there could be an opportunity for practical examples of different activities to be videoed to allow teachers to see how they work in a classroom context.

7 FIRST YEAR UNDERGRADUATE STUDY

7.1 INTRODUCTION

One of the main goals of most introductory Computer Science courses in third level education is to allow students to develop critical thinking and problem-solving abilities that will be useful not just in future years of study, but throughout their life. Indeed, this is one of the hopes of almost all university courses, with problem solving, critical thinking and analytical skills being highly sought after skills in almost all sectors of work. Universities and colleges hope to produce graduates with not just subject-area knowledge but also these important cross-domain skills.

The first-year teaching group in the Department of Computer Science at Maynooth University have tried many different strategies to pass on both subject-specific knowledge as well as develop students who can solve difficult problems through both programming and computational thinking. As well as being involved in the education of third-level students, the Department of Computer Science at Maynooth University has, for many years, strived to pass their knowledge and experience to secondary school students and teachers. This has been done through many ways including summer camps, workshops, open days and college fairs. One way in particular is the PACT group, who have developed resources and materials for secondary school and primary school teachers. One of the main focusses of the group is on using Computational Thinking to teach cross curricular skills.

Alongside testing the CT Assessment, described in detail in Chapter 4, in schools during the 2017-18 school year it was decided that this test would also be suitable at third-level. The decision was made to run the CT Assessment and "Views of CS" survey on the first-year CS students at Maynooth University. The plan was to show students that problem-solving is a key part of CS and to see whether the course improved their CT skills, whether their opinions of CS changed over the course of the year and also whether the CT Assessment could be a predictor of success in the course. The large and consistent group also allowed for a better evaluation of the CT Assessment than has so far been possible in secondary schools, so the data could be used to better adapt and adjust the test to suit all groups.

7.2 FIRST YEAR COURSE IN MAYNOOTH

In Maynooth University the first year Computer Science course consists of two separate streams. These streams are described in the following sections.

7.2.1 STREAM 1

The first stream consists of the modules CS161 (Introduction to Computer Science 1) and CS162 (Introduction to Computer Science 2). These modules are taught in semester one and two respectively of first year. For anyone wanting to study Computer Science in Maynooth these are mandatory modules which all students must take, and they are pre-requisites for all second-year modules. The modules are 7.5 ECTs each and consist of 36 Lecture hours (3 hours per week), 36 laboratory hours (3 hours per week) and 48 independent study hours.

The overview and learning outcomes of these two modules, adapted from the university course guide, are: Students will be taught programming fundamentals (all through Java) including variables, types, expressions, simple I/O, conditional and iterative control structures, String and use of class API's for creating objects and calling methods. Students will learn about object-oriented programming, arrays, static methods, variable scope and sorting and searching algorithms. They will also learn about languages, regular expressions, finite automata, recursion, basic computer architecture, components of modern operating systems and number systems.

During these modules student will be given an understanding of algorithms and problem solving and be able to create and write their own algorithms as well as understand and evaluate others. They should then be able to take these algorithms and implement them using Java. These two modules are taught primarily using the Java programming language. Java is a popular first language and has many advantages when used as the first language taught. The theory parts of the course are usually taught using practical exercises such as making up Finite State Machines using JFlap, converting numbers from binary to decimal and writing out regular expressions. These are often followed by programming exercises which show how Java can be used to do these tasks. A key focus in the first semester module is engaging and testing student's problem-solving skills using a few different exercises. These are often done in pairs or small groups and are sometimes, but not always, linked to the topic of the programming lab that week. These include Bebras problems as previously described and other puzzles and logic problems.

7.2.2 STREAM 2

The second stream consists of the modules CS171 (Computer Systems 1) and CS172 (Computer Systems 2). These modules are taught in semesters one and two respectively. For anyone taking the denominated degrees in Computer Science and Software Engineering or Multimedia and Mobile Web Development these are compulsory modules. For omnibus Science and Arts students or those studying other subjects these modules are optional. The modules are 7.5 ECTs each, but the make-up is different for each, and they cover very different topics unlike CS161 and CS162 which are intrinsically linked. CS171 consists of 36 Lecture hours (3 hours per week), 36 laboratory hours (3 hours per week) and 48 independent study hours. The overview and learning outcomes, adapted from the university course guide, are below: Students will learn about a wide range of CS topics, examples include Artificial Intelligence, Signal Processing, Computer visions and Machine learning. Concepts are taught practically through use of the graphical programming language, Processing.

CS172 consists of 36 Lecture hours (3 hours per week), 24 laboratory hours (2 hours per week), 12 tutorial hours (1 hour per week) and 48 independent study hours. The overview and learning outcomes of these two modules, adapted from the university course guide, are: Students will be introduced to first order logic and proof strategies. Basic set operations and properties as well as defining and understanding inverse, image, bijections, total & partial functions etc. will be covered. Students will learn to reason and evaluate propositional and predicate logic statements as well as syllogism and natural theory proofs. These will be done using truth tables, syntax trees, contradiction and natural deductions. Students will also learn programming in a declarative language (Prolog) and learn about lists, recursion and search strategy.

7.3 UNDERGRADUATE STUDY - 2017-18

The academic year runs from late September through to May and consists of two semesters. Both semesters run for 12 weeks, the first from late September to December, with examinations taking place in January. The second semester runs from early February through to early May, with examinations taking place in May. Repeat examinations are offered in August for any student failing one of more modules.

During the third week of semester one, the first-year students were told about the study and asked to participate. This was optional and took place during their mandatory lab time; however students were not graded on the CT Assessment nor did their participation/non-participation have any impact on their module grade. Students were then asked to fill out two surveys. The first survey was a personal survey to provide demographic data, and the second survey allowed us to gather perceptions on their views of CS, described in Sections 4.1.1 and 4.2. They then took part in the first CT Assessment. As with the secondary school students they were allowed 35 minutes to complete the problems and could use pen & paper. Students could leave early if they completed the test. The results as well as the answers to the questions were then released the following week to all students.

In week 10 of the second semester the students were again asked during their lab time to take part in another CT Assessment. It was explained that this test had been uniquely developed for them and that the reason for running the test was to allow them to see if their CT skills had improved over the course of a semester. In week 10 of semester two the students took the final CT Assessment as well as completing the second View of Computer Science survey.

The first-year cohort consisted of more than 300 students at the beginning of the academic year. A total of 271 of these were present to fill out a survey collecting personal and demographic data. This pool comes from a wide-range of courses and backgrounds and below are some demographic data points of interest. As is the case with most CS undergraduate courses, the majority of those taking the course were males, with 75% of the class who we surveyed being male.

As this is an introductory module taken by first year students, most students taking the course had just finished secondary school after taking the state-wide exams, the Leaving Certificate. This can be observed in Figure 7.1 where 224 of the 271 respondents (82%) were aged 17-19 years old; this would be the normal age bracket of students taking the state exams. There was a total of 26 mature students, that is students aged 23 and older.

In Maynooth University there are several different ways for students to graduate with a qualification in Computer Science. Many courses also allow or require the first year CS modules (CS161 & CS162) to be taken. Below is a list of all of the courses represented from the data we collected, the course codes are assigned by Maynooth University. The number of each can be seen in Table 7.2.

- Theology and Arts - MH001
- Robotics and Intelligent Devices - MH306[32]
- Psychology through Science - MH209
- Physics with Astrophysics - MH204

32 Re-titled as Robotics from 2019

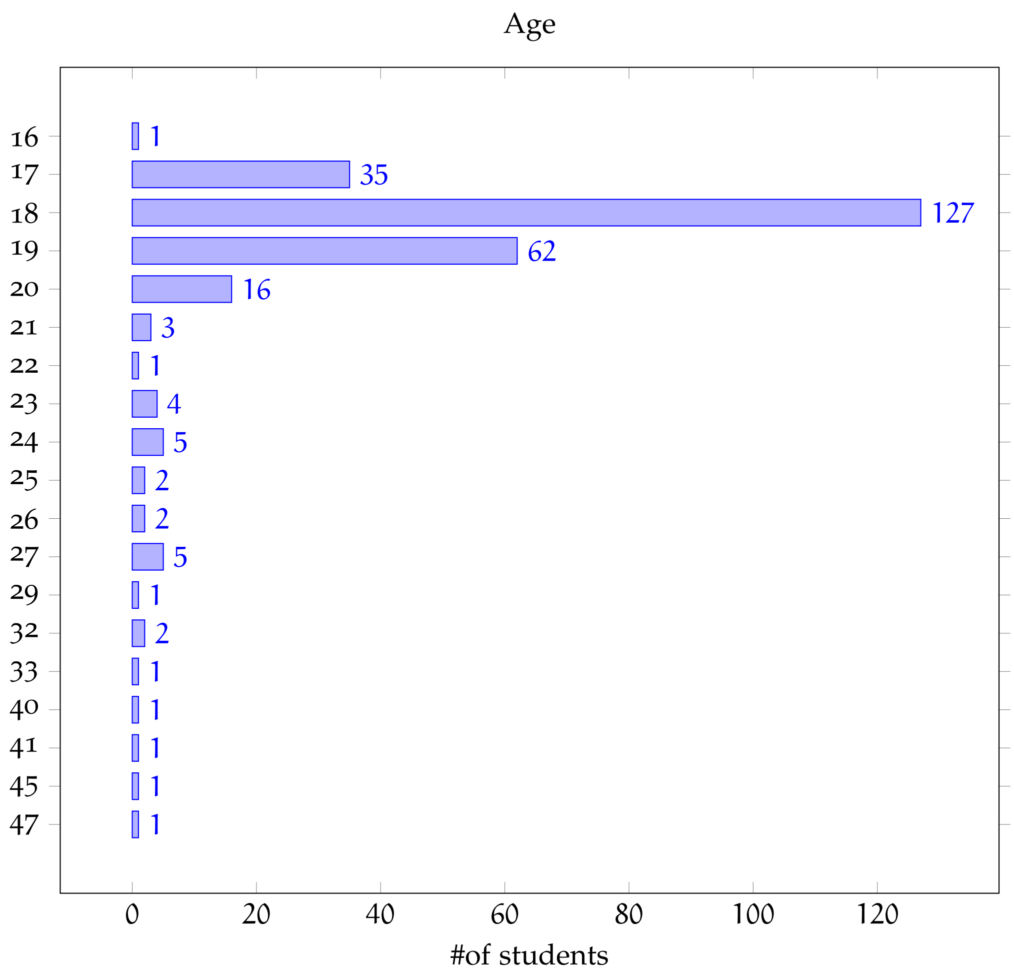


Figure 7.1: Age of students taking CS

- Pharmaceutical and Biomedical Chemistry - MH210
- Multimedia Mobile and Web Development (MMWD) - MH601
- Media Technology
- Media Studies - MH109
- Mathematics with Education - MH212
- Finance - MH401
- Exchange student
- Computer Science & Software Engineering (CSSE) - MH601/602[33]
- Biological and Biomedical Science - MH208
- Bachelor of Science - MH201

33 Will be known as CSSE from now on

- Bachelor of Arts - MH101

It is worth mentioning the four largest course participation amongst the students. Computer Science & Software Engineering (CSSE) (86 students) and Robotics & Intelligent Devices (32 students) are both denominated degrees provided by the Department of Computer Science. CSSE students must take both CS streams as described in Section 7.2. The two largest degree programmes in Maynooth University are the Bachelor of Science (MH201) and Bachelor of Arts (MH101) and Computer Science is an option for both programmes and 112 students across these courses were enrolled in CS.

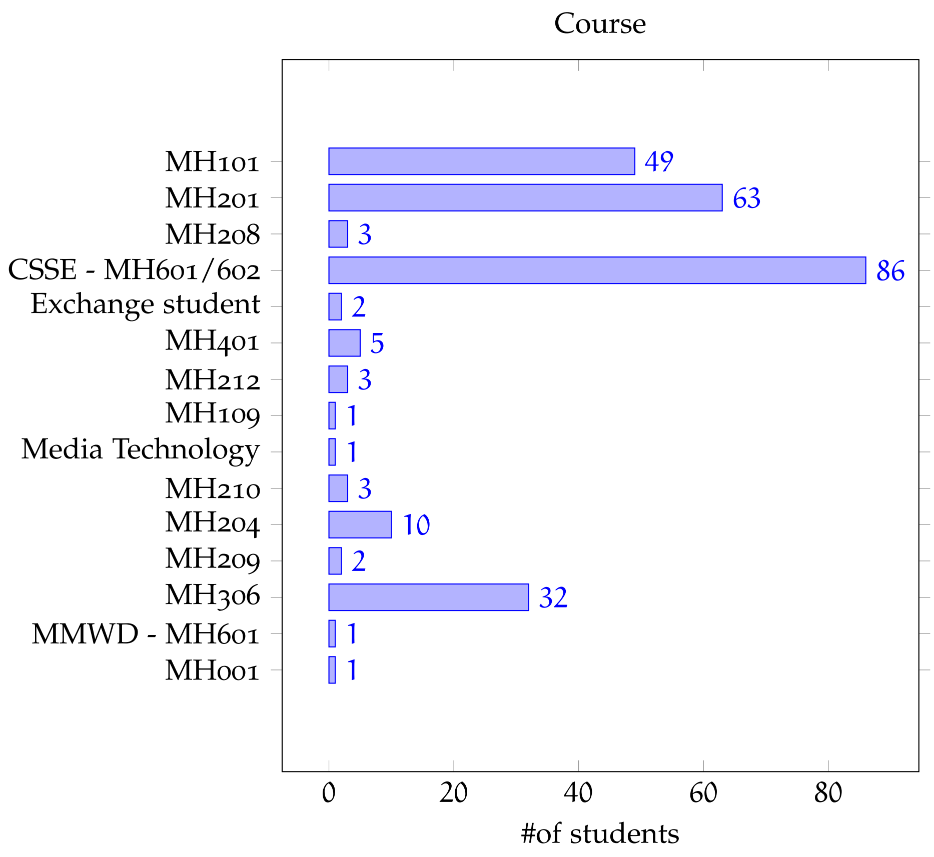


Figure 7.2: Courses taken by students in first year Computer Science

The first-year modules are designed to teach students the basics of programming and introduce them to fundamental CS concepts. The course assumes zero prior CS knowledge and aims to keep all students following along with the curriculum. From our data this is representative of the class, with 55% of the class never having programmed before highlighted in Figure 7.3.

Many of the 45% who had programmed before have very limited knowledge which can be seen more clearly from Figure 7.3. Over 50% of students stated they had never programmed before. It should also be noted that even those who have had previous experience may have little to no knowledge. In Figure 7.3 it can be seen that only 8% of students have been programming for over a year prior to the start of this course. Over three-quarters (78%) have only programmed once or twice at most.

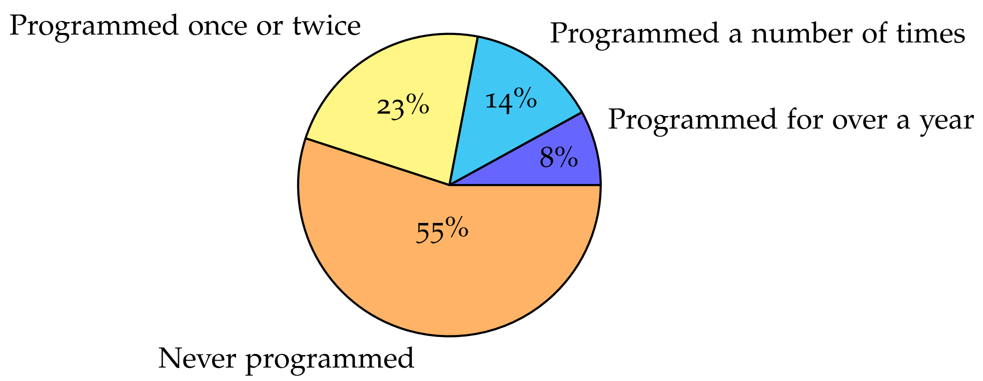


Figure 7.3: How much programming experience do you have?

Fig 7.4 shows the results from students self-reported responses to how often they program. Answers were on a Likert scale with 1 = Not at all, and 5 = Daily. This data, coupled with Figure 7.3 reinforces the assumption that most of the class (at least 70%) had virtually zero knowledge of programming, whilst less than 10% have any real experience. Some of these students will be relatively experienced and strong programmers can find the content very basic. The introduction of stream 2 allows those students who are interested in pursuing CS beyond first year or who have a desire to focus on it to further their skill set beyond programming. With the introduction of CS in the Leaving Certificate students studying CS will, potentially, be stronger programmers and have far more knowledge of CS than is currently the norm. This will be a problem across the third level educational institutions and will need to be dealt with in due course.

The Leaving Certificate Mathematics exam is generally taken at either Ordinary or Higher level. There is a Foundation level which can be made available to students if necessary. To be admitted into many courses in Maynooth University you need a O3 or O4 (60%<70% and 70%<80% in Ordinary Level respectively) grade as a minimum requirement[34]. This leads to a wide-range in mathematics abilities in most first year classes. This can lead to problems as CS and mathematics are related and strength in one can often help the other, whereas a weakness in mathematics can often lead to difficulties in CS [77]. Almost all students who take CS as a subject in first year must be taking first year mathematics which should allow students to be at a level which

34 `https://www.maynoothuniversity.ie/study-maynooth/undergraduate-studies/how-apply/school-leaver-applicants-leaving-certificate`

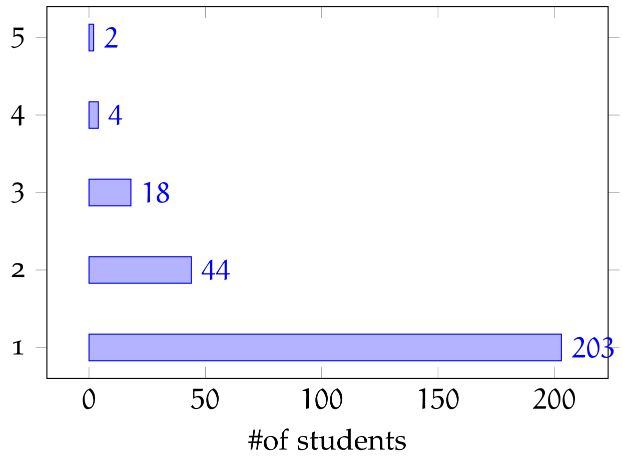


Figure 7.4: How often do you program (1-5)

allows them not to be hindered in CS by their mathematics ability. For any students pursuing CS further they must take mathematics modules in at least second year as well.

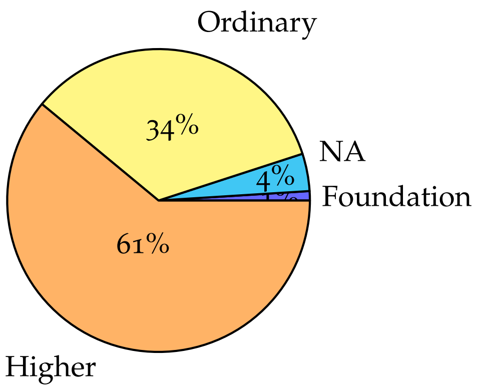


Figure 7.5: What level of mathematics did you take for the Leaving Certificate

As with programming, most students have never done any website or app development as shown in Figure 7.6. This question was mainly asked as both topics are becoming more popular in secondary schools and informal educations groups such as summer camps and CoderDojo's.

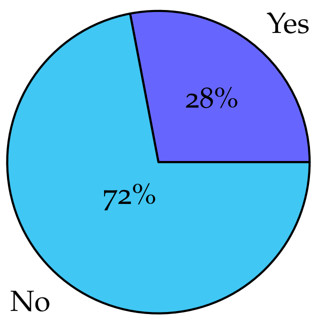


Figure 7.6: Have you ever done any website/app development?

7.4 RESULTS

7.4.1 COMPUTATIONAL THINKING ASSESSMENT

The CT Assessments have been discussed and described in detail in Chapter 4 and can be found in Appendix K and L. They were developed with the aim of testing students Computational Thinking skills and, although aimed at secondary school students, the skills should be ones which CS students at third level aim to improve during their studies. For the purposes of this section we will refer to Test 1 as the test taken in the third week of semester one (September 2017) and Test 2 will refer to the test taken in the tenth week of semester two (April 2018). A total of 292 students took at least one of the two CT Assessments. Of those 292, 263 took Test 1 and 180 took Test 2. The decrease in numbers is due to students changing course, only needing to complete one semester of CS and other unrelated circumstances.

Table 7.1: CT Assessment scores

Whole population	Average of those who took at least one test (whole population)	Average of those who took Test 1 and Test 2 (n = 174)
Test 1 (n = 263)	7.734	7.988
Test 2 (n = 180)	8.05	8.035

From Table 7.1 it can be seen that a total of 174 took both tests. Students performed marginally better in Test 2 compared to Test 1, but this isn't a significant difference (t-value = 0.196, p-value = 0.845). This increase is also found across the whole population with the averages being 7.734 for Test 1 (n=263) and 8.05 for Test 2 (n=180) out of 13, although again this is not a significant change (t-score = 1.485, p-value = 0.138). For the following results we will focus on the 174 people who took Test 1 and Test 2. Table 7.2 presents some demographic data for these students which we will be referring to for the rest of this analysis. Of those 174 students, 14 did not fill out the survey so this data is for the remaining 160. For the list of courses, only those with 10 or more students enrolled are analysed, the next largest group was seven students.

Table 7.2: Demographic Data

	Demographic	N
Gender	Male	123
	Female	37
Mathematics Level	Studied OL Maths in Secondary School	44
	Studied HL Maths in Secondary School	111
	Studied FL Maths in Secondary School	1
	N/A to Maths Level	4
Previous Experience	Had previous programming experience	72
	Had no previous programming experience	88
Programme of Study	Bachelor of Science - MH201	41
	Robotics - MH306	19
	Bachelor of Arts - MH101	23
	CSSE - MH601/602	56

HL = Studied Higher Level mathematics, OL = Studied Ordinary Level mathematics, FL = Studied Foundation Level mathematics

Table 7.3 presents the results of t-tests comparing the different demographic groups with each other. To make reading the data easier, each demographic group is separated by a horizontal line. The right-most column (T-Test result), compares the Test 1 average with the Test 2 average of the group presented in that row. The final row in each group compares the results of Test 1 and Test 2 of the groups against each other e.g. Male scores in Test 1 compared to Female scores in Test 1. For the maths level question only higher and ordinary level are compared.

Table 7.3: CT Assessment scores - Demographic Breakdown Part 1

Demographic	Test 1 Average	Test 2 Average	T-Test Result (test 1 v test 2)
Male (n = 121)	8.164	8.189	T-score = 0.087 P-Value = 0.931
Female (n = 37)	7.703	7.784	T-Score = 0.167 P-Value = 0.868
T-Test: Male v Female	T-score = 1.097 P-Value = 0.274	T-score = 1.021 P-Value = 0.309	
Studied OL (n = 44)	7.386	7.273	T-score = 0.257 P-Value = 0.798
Studied HL (n = 110)	8.445	8.509	T-score = 0.225 P-Value = 0.822
T-Test: OL v HL	T-score = 2.768 P-Value = 0.006	T-score = 3.409 P-Value = 0.001	
Previous programming experience (n = 71)	8.225	8.085	T-score = 0.385 P-Value = 0.701
No previous programming experience (n = 88)	7.92	8.102	T-score = 0.553 P-Value = 0.581
T-Test: PPE v NPPE	T-score = 0.853 P-Value = 0.395	T-score = 0.053 P-Value = 0.958	

PPE = Previous programming experience, NPPE = No previous programming experience

Table 7.4 presents the Computational Thinking Assessment score for the four main course groups. The T-tests presented there compare each courses progression from Test 1 to Test 2.

Table 7.4: CT Assessment scores - Demographic Breakdown Part 2

Demographic	Test 1 Average	Test 2 Average	T-Test Result (test 1 v test 2)
Bachelor of Science - MH201 (n=41)	8	8.146	T-Score = 0.329 P-Value = 0.742
Robotics - MH306 (n=19)	8.7	8.6	T-Score = 0.072 P-Value = 0.943
Bachelor of Arts - MH101 (n=23)	7.957	7.565	T-Score = 0.506 P-Value = 0.615
CSSE - MH601/602 (n=56)	7.589	7.768	T-Score = 0.472 P-Value = 0.638

7.4.2 VIEW OF CS SURVEY

Students completed a survey to determine their views of Computer Science both in week 3 of semester 1 and week 10 of semester 2. The questions are described in Section 4.2 and the full survey can be found in Appendix J and were the same as those used by the Transition Year students with the exception of one. The question "Would you/have you considered studying CS in college/university?" was removed. The total number of people who completed both tests was 147. These people are not necessarily a perfect subset of the group who completed problem solving Test 1 and Test 2 described in the previous section, however it is in general. Table 7.5 provides a demographic breakdown of this group of students. It can be seen that the majority of students are male (78%), and that most studied Higher Level maths (70%). In terms of previous programming experience it is nearly a 50-50 split. The four largest courses that students are enrolled in are presented, which accounts for 86% of students with the other 14% being distributed in small numbers over other courses.

Table 7.6 presents the initial results of the various Likert scale questions. The T-Tests of the Likert scale question are also presented. A score of one indicated that the participant strongly disagreed with the statement and a score of 5 indicated that the participant strongly agreed with the statement.

Table 7.7 presents the number and percentage of Yes and No responses to questions 1, 2 and 12.

Table 7.5: Demographic Data - those who completed both View of CS surveys

	Demographic	N
Gender		
	Male	114
	Female	33
Mathematics Level		
	Studied OL Maths in Secondary School	39
	Studied HL Maths in Secondary School	104
	N/A to Maths Level	4
Previous Experience		
	Had previous programming experience	70
	Had no previous programming experience	77
Programme of Study		
	MH201	39
	MH306	19
	MH101	17
	MH601/602	52

One thing to note is that the results and percentages shown in Table 7.6 and Table 7.7 are roughly in line with the whole population that took each survey in terms of percentages and trends in the changes. These results are presented in Table 7.8 and Table 7.9. For Q1, Q2 and Q12 the difference between the whole population and those who completed both surveys is between 0-7%. The trends also match with the percentage answering yes to Q2 and Q12 increasing in both and Q1 decreasing in both.

When we compare the Likert rated questions (Q3-Q11) of those who took both surveys, displayed in Table 7.6, and those who took at least one, displayed in Table 7.8 we find the results are very similar. All but one of the questions are within 0.1 (out of 5) when comparing the two samples, this outlier was Q9. For Q9 the whole population started with an average of 3.205 compared to 3.926 for those who completed both. It is hard to see any reason for this, but it should be noted that the gap is closed

Table 7.6: View of CS results Part 1 - those who completed both surveys

Question	Avg Survey 1 (n = 149)	Avg Survey 2 (n = 149)	T-Test
Q3. Using the internet is central to Computer Science	3.309	3.403	T-Score = 0.831 P-Value = 0.407
Q4. Using Word, Excel etc. is central to Computer Science	2.415	2.04	T-Score = 3.665 P-Value = 0.000
Q5. Installing software (e.g. Windows, iTunes) is central to Computer Science	3	2.725	T-Score = 2.101 P-Value = 0.036
Q6. Programming is central to Computer Science	4.658	4.631	T-Score = 0.411 P-Value = 0.682
Q7. Being able to solve different problems is central to Computer Science	4.678	4.812	T-Score = 2.184 P-Value = 0.029
Q8. Computer Science is an area related to maths	4.174	4.101	T-Score = 0.819 P-Value = 0.413
Q9. A computer scientist should be good at working with others	3.926	3.913	T-Score = 0.111 P-value = 0.912
Q10. Boys/men are more likely to study Computer Science than girls/women	3.175	3.215	T-Score = 0.287 P-Value = 0.774
Q11. Work in Computer Science can be done without a computer	2.846	3.168	T-Score = 2.465 P-Value = 0.014

Table 7.7: View of CS results Part 2 - those who completed both surveys

Question	No. of Yeses in Survey 1 (n = 149)	Percent	No. of Yeses in Survey 2 (n = 149)	Percent
Q1. Is CS interesting to you? (Yes, No, Maybe)	130	87.25%	122	81.88%
Q2. Is CS challenging? (Yes, No, Maybe)	85	57.05%	126	84.56%
Q12. Have you heard the term CT? (Yes, No)	91	61.07%	107	71.81%

considerably by survey 2 with the whole population average being 3.874 and those who completed both having an average of 3.913. The statistically significant changes, namely to Q4, Q5, Q7 and Q11, are also consistent across both samples. Averages for Q4 and Q5 both decreased from survey 1 to survey 2 with averages for Q7 and Q11 increasing from survey 1 to survey 2.

Moving back to the results of those who took both surveys, presented in Table 7.6, it is both encouraging and expected that students view for Q4 and Q5 would move towards a stronger disagreement. After being exposed to CS for a whole year through programming, logic, computer systems and more introductory CS concepts the students would hopefully understand that CS is more linked to programming and problem solving (Q6 & Q7) which can be seen to be true in Table 7.6. Students responses to Q6 increased slightly but not significantly, however their responses to Q7 changed significantly from 4.627 in survey 1 to 4.77 in survey 2 (T-Score = 2.47, P-Value = 0.014). Of interest with Q11 is that views about work with CS changed significantly with participants agreeing more strongly with the statement. In survey 1 the average was 2.846 (out of 5) and this shifted to 3.168 in survey 2 (T-Score = 2.465, P-value = 0.014). Although a rating of around 3 implies an almost neutral response, the shift is still present. This could be down to several factors including pen and paper exercises in the various modules or having a better understanding of what CS is and what it involves.

With the three yes/no questions (Q1, Q2 & Q12) shown in Table 7.7 some interesting observations can be made. Firstly, the increase in those having heard of CT is

Table 7.8: View of CS results Part 1 - all who completed a survey

Question	Avg Survey 1 (n = 268)	Avg Survey 2 (n = 183)	T-Test
Q3. Using the internet is central to Computer Science	3.313	3.371	T-Score = 0.569 P-Value = 0.569
Q4. Using Word, Excel etc. is central to Computer Science	2.388	2.011	T-Score = 4.482 P-Value = 0.000
Q5. Installing software (e.g. Windows, iTunes) is central to Computer Science	3.034	2.65	T-Score = 3.457 P-Value = 0.000
Q6. Programming is central to Computer Science	4.589	4.617	T-Score = 0.453 P-Value = 0.651
Q7. Being able to solve different problems is central to Computer Science	4.627	4.77	T-Score = 2.470 P-Value = 0.014
Q8. Computer Science is an area related to maths	4.075	4.04	T-Score = 0.394 P-Value = 0.694
Q9. A computer scientist should be good at working with others	3.205	3.874	T-Score = 0.611 P-Value = 0.542
Q10. Boys/men are more likely to study Computer Science than girls/women	3.175	3.224	T-Score = 0.159 P-Value = 0.873
Q11. Work in Computer Science can be done without a computer	2.6646	3.098	T-Score = 4.069 P-Value = 0.000

Table 7.9: View of CS results Part 2 - all who completed a survey

Question	No. of Yes answers in Survey 1 (n = 268)	Percent	No. of Yes answers in Survey 2 (n = 183)	Percent
Q1. Is CS interesting to you? (Yes, No, Maybe)	224	83.58%	143	78.14%
Q2. Is CS challenging? (Yes, No, Maybe)	161	60.07%	155	84.69%
Q12. Have you heard the term CT? (Yes, No)	146	54.78%	107	67.76%

encouraging but it is worrying that this skill hasn't been made clear to almost 30% of the cohort. As described in Table 7.9 this percentage was higher for the whole population. CT has been stated as a central skill for all students [97] and if this is true then students should be taught and reminded about how to use this problem-solving methodology throughout the first-year course. It should be noted that CT is not explicitly mentioned in any module. The increase in those agreeing that CS is challenging (Q2) is to be expected as many students wouldn't have had previous experience of it before and so after a year they can appreciate the difficulties of the subject. This finding alongside the high drop-out ratio for CS students make the new Leaving Certificate course, and the Junior Cert coding course, even more important as this will allow students to experience the subject before entering third-level education.

As previously discussed in Section 6.4.3 it is our belief that the large drop-out rate in CS undergraduate courses [77] is, in part, down to a lack of understanding of the subject. Many see CS as highly employable, which it is with a large number of jobs available[35], and so feel that this is a smart or safe option without experiencing either programming or other areas before entering an undergraduate course. The subject then may not live up to their expectations or be what they expected, which can lead not only to them changing course but also completely disengaging with the subject.

35 https://eu.usatoday.com/story/tech/talkingtech/2017/03/28/tech-skills-gap-huge-graduates-survey-says/99587888/

This has been reported colloquially through all levels of teaching from the lecturers, lab demonstrators and volunteer tutors.

There were also some interesting differences between the different demographic groups. The results for each question are presented in individual tables with three separate demographic groups being considered. These are Gender (Male, Female), Maths level completed at Secondary school (Ordinary Level, Higher Level) and previous programming experience (has previous experience, has no previous experience). The results of each group (e.g. Male) are compared from survey 1 to survey 2 as well as comparing the demographics in each survey (e.g. Male v Female in survey 1). These results are presented over a number of tables and we will discuss these now.

One major demographic comparison group investigated was in relation to those who reported having some experience programming prior to the course. Looking at these two groups we can get an insight to some misconceptions people might have about CS to go alongside those seen when looking at the genders. Firstly, those who had no previous experience felt more strongly that using word, excel etc. (Q4 in Table 7.11) was central to CS. The interesting point with this is that this shifts dramatically by the end of the course, with there being a significant difference in the opposite direction. This could show that the course has successfully changed their view of what CS is and isn't.

Male respondents views changed more significantly over the year than females when related to the centrality of using and installing software to CS (Q4 & Q5 in Tables 7.11 and Table 7.12). It can be seen that males views dropped from an average of 2.377 to 2 for Q4 and from 2.964 to 2.675 for Q5. These drops are significant but also both averages start lower than their female peers. This means that any misconception is shared prior to the course but males see them as less central after a year of introduction to the subject.

When looking at the two different streams of maths level at second level we can observe two interesting findings. Firstly, those who studied Ordinary level (OL) maths see CS as more strongly linked to maths than their Higher-level (HL) peers (Q8 in Table 7.15). This isn't a shift from the first survey results as the same result was found but the difference has increased with those who studied OL shifting slightly towards strongly agreeing with those who studied HL shifting the other direction.

The difference between male and female responses include that males agree more strongly that a Computer Scientist should be good at working with others (Q9 in Table 7.16). The gap increases slightly over the course of the year with the Male rating increasing from 4.018 to 4.044 and females decreasing from 3.606 to 3.545. Also, of interest, though not particularly significant statistically, is that those who studied OL

Table 7.10: Q3 Using the Internet is central in Computer Science

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	3.307 (n = 114)	3.421 (n = 114)	T-Score = 0.794 P-Value = 0.428
Female	3.303 (n = 33)	3.394 (n = 33)	T-Score = 0.372 P-Value = 0.711
T-Test Result Male v Female	T-Score = 0.021 P-Value = 0.983	T-Score = 0.118 P-Value = 0.906	
Studied OL Maths in Secondary School	3.487 (n = 39)	3.462 (n = 39)	T-Score = 0.095 P-Value = 0.925
Studied HL Maths in Secondary School	3.221 (n = 104)	3.375 (n = 104)	T-Score = 1.063 P-Value = 0.289
T-Test Result OL v HL	T-Score = 1.489 P-Value = 0.139	T-Score = 0.395 P-Value = 0.693	
Had previous programming experience	3.343 (n = 70)	3.529 (n = 70)	T-Score = 1.044 P-Value = 0.299
Had no previous programming experience	3.273 (n = 77)	3.312 (n = 77)	T-Score = 0.229 P-Value = 0.819
T-Test Result No exp v Had exp	T-Score = 0.443 P-Value = 0.658	T-Score = 1.135 P-Value = 0.258	

Table 7.11: Q4 Using Word, Excel etc. is central in Computer Science

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	2.377 (n = 114)	2 (n = 114)	T-Score = 3.355 P-Value = 0.000
Female	2.545 (n = 33)	2.182 (n = 33)	T-Score = 1.465 P-Value = 0.148
T-Test Result Male v Female	T-score = 0.964 P-Value = 0.336	T-Score = 1.033 P-Value = 0.303	
Studied OL Maths in Secondary School	2.538 (n = 39)	2.026 (n = 39)	T-Score = 2.316 P-Value = 0.023
Studied HL Maths in Secondary School	2.375 (n = 104)	2.029 (n = 104)	T-Score = 2.894 P-Value = 0.004
T-Test Result OL v HL	T-score = 0.977 P-Value = 0.330	T-Score = 0.019 P-Value = 0.985	
Had previous programming experience	2.229 (n = 70)	2.171 (n = 70)	T-Score = 0.397 P-Value = 0.692
Had no previous programming experience	2.584 (n = 77)	1.922 (n = 77)	T-Score = 4.583 P-Value = 0.000
T-Test Result No exp v Had exp	T-score =2.485 P-Value = 0.014	T-Score = 1.707 P-Value = 0.09	

Table 7.12: Q5 Installing software (e.g. Windows, iTunes) is central in CS

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	2.964 (n = 114)	2.675 (n = 114)	T-Score = 1.919 P-Value = 0.056
Female	3 (n = 33)	2.812 (n = 33)	T-Score = 0.683 P-Value = 0.497
T-Test Result Male v Female	T-Score = 0.161 P-Value = 0.872	T-Score = 0.628 P-Value = 0.531	
Studied OL Maths in Secondary School	2.923 (n = 39)	2.641 (n = 39)	T-Score = 1.076 P-Value = 0.316
Studied HL Maths in Secondary School	2.962 (n = 104)	2.716 (n = 104)	T-Score = 1.626 P-Value = 0.105
T-Test Result OL v HL	T-Score = 0.186 P-Value = 0.853	T-Score = 2.712 P-Value = 2.641	
Had previous programming experience	2.929 (n = 70)	2.829 (n = 70)	T-Score = 0.542 P-Value = 0.589
Had no previous programming experience	3.013 (n = 77)	2.597 (n = 77)	T-Score = 2.239 P-Value = 0.027
T-Test Result No exp v Had exp	T-Score = 0.464 P-Value = 0.643	T-Score = 1.221 P-Value = 0.224	

Table 7.13: Q6 Programming is central in Computer Science

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	4.623 (n = 114)	4.614 (n = 114)	T-Score = 0.114 P-Value = 0.909
Female	4.788 (n = 33)	4.697 (n = 33)	T-Score = 0.728 P-Value = 0.469
T-Test Result Male v Female	T-Score = 1.584 P-Value = 0.115	T-Score = 0.701 P-Value = 0.485	
Studied OL Maths in Secondary School	4.564 (n = 39)	4.692 (n = 39)	T-Score = 1.009 P-Value = 0.316
Studied HL Maths in Secondary School	4.692 (n = 104)	4.615 (n = 104)	T-Score = 0.974 P-Value = 0.331
T-Test Result OL v HL	T-Score = 1.286 P-value = 0.201	T-Score = 0.682 P-Value = 0.497	
Had previous programming experience	4.657 (n = 70)	4.623 (n = 70)	T-Score = 0.144 P-Value = 0.886
Had no previous programming experience	4.662 (n = 77)	4.643 (n = 77)	T-Score = 0.443 P-Value = 0.659
T-Test Result No exp v Had exp	T-Score = 0.059 P-Value = 0.953	T-Score = 0.197 P-Value = 0.844	

Table 7.14: Q7 Being able to solve different problems is central in CS

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	4.675 (n = 114)	4.842 (n = 114)	T-Score = 1.887 P-Value = 0.067
Female	4.667 (n = 33)	4.758 (n = 33)	T-Score = 0.671 P-Value = 0.505
T-Test Result Male v Female	T-Score = 0.072 P-Value = 0.942	T-Score = 1.025 P-Value = 0.307	
Studied OL Maths in Secondary School	4.589 (n = 39)	4.846 (n = 39)	T-Score = 1.917 P-Value = 0.059
Studied HL Maths in Secondary School	4.692 (n = 104)	4.808 (n = 104)	T-Score = 1.651 P-Value = 0.01
T-Test Result OL v HL	T-Score = 0.886 P-Value = 0.377	T-Score = 0.484 P-Value = 0.629	
Had previous programming experience	4.671 (n = 70)	4.886 (n = 70)	T-Score = 2.611 P-Value = 0.01
Had no previous programming experience	4.675 (n = 77)	4.766 (n = 77)	T-Score = 1.018 P-Value = 0.310
T-Test Result No exp v Had exp	T-Score = 0.039 P-Value = 0.969	T-Score = 1.747 P-Value = 0.083	

Table 7.15: Q8 Computer Science is an area related to maths

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	4.184 (n = 114)	4.132 (n = 114)	T-Score = 0.501 P-Value = 0.617
Female	4.182 (n = 33)	4.03 (n = 33)	T-score = 0.846P-Value = 0.401
T-Test Result Male v Female	T-Score = 0.016 P-Value = 0.987	T-Score = 0.629 P-Value = 0.529	
Studied OL Maths in Secondary School	4.231 (n = 39)	4.282 (n = 39)	T-Score = 0.309 P-Value = 0.758
Studied HL Maths in Secondary School	4.144 (n = 104)	4.019 (n = 104)	T-Score = 1.139 P-Value = 0.256
T-Test Result OL v HL	T-score = 0.621 P-Value = 0.536	T-Score = 1.734 P-Value = 0.085	
Had previous programming experience	4.243 (n = 70)	4.2 (n = 70)	T-Score = 0.325 P-Value = 0.746
Had no previous programming experience	4.129 (n = 77)	4.026 (n = 77)	T-Score = 0.835 P-Value = 0.405
T-Test Result No exp v Had exp	T-Score = 0.924 P-Value = 0.357	T-Score = 1.301 P-Value = 0.195	

Table 7.16: Q9 A computer scientist should be good at working with others

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	4.018 (n = 114)	4.044 (n = 114)	T-Score = 0.239 P-Value = 0.811
Female	3.606 (n = 33)	3.545 (n = 33)	T-Score = 0.239 P-Value = 0.812
T-Test Result Male v Female	T-Score = 2.349 P-Value = 0.02	T-Score = 2.882 P-Value = 0.005	
Studied OL Maths in Secondary School	4.102 (n = 39)	4.077 (n = 39)	T-Score = 0.127 P-Value = 0.899
Studied HL Maths in Secondary School	3.846 (n = 104)	3.856 (n = 104)	T-Score = 0.155 P-Value = 0.877
T-Test Result OL v HL	T-Score = 1.511 P-Value = 0.132	T-Score = 1.259 P-Value = 0.21	
Had previous programming experience	4.129 (n = 70)	4.057 (n = 70)	T-Score = 0.53 P-Value = 0.597
Had no previous programming experience	3.74 (n = 77)	3.818 (n = 77)	T-Score = 0.493 P-Value = 0.663
T-Test Result No exp v Had exp	T-Score = 2.668 P-Value = 0.008	T-Score = 1.623 P-Value = 0.107	

agree more strongly that a Computer Scientist should be good at working with others. The gap closes slightly from survey one to two but is still present. A final finding in relation to this question is that those with previous experience agree more strongly that a Computer Scientist needs to be good at working with others. This gap remains at the end of the course but is again narrowed. One of the misconceptions about CS is that it can be a very lonely job with someone working in a dark room all alone typing at a terminal. This is rarely the case as Computer Scientists must work not only as a team if developing a solution or software program but also with clients, management and the public.

One major debate in both CS education and the wider STEM education areas is the engagement and teaching of female students of all ages. When looking at the feedback to Q10 in Table 7.17 we can see that there is a significant difference between male and female participants response to the question "Boys/men are more likely to study Computer Science than girls/women". Males tend to disagree more strongly with the statement going into the course and at the courses conclusion. They responded with an average of 3.044, compared to 3.606 for Females. From survey 1 to survey 2 males opinions shift slightly, but not significantly, towards agreement (3.044 in survey 1 to 3.105 for survey 2), whereas female's views didn't change at all on average (3.606 in both surveys).

We believe there could be several reasons for this. One could be the fact that, numerically, it is almost always the case in third-level CS courses that there are more males than females. This can be seen in this cohort, even though the whole-college population of Maynooth is majority female (56% according to official university figures[36]). Other reasons could include the stereotyped view of CS which has extensively been reported on [2, 9, 78] and has been seen in other areas of this wider study. It is interesting however that females report a stronger agreement than males with Q10, although we shouldn't be quick to draw conclusions on this sample; this could show that more work needs to be done in second-level to show girls that CS isn't a male-only discipline, even if at present it is a male-dominated one. In relation to Q10, those who had no previous CS experience agreed more strongly on average than those with experience in survey 1 (3.325 compared to 3, T-Score = 1.565, P-value = 0.119). The gap is effectively gone by survey 2 with averages of 3.2 for those with previous experience and 3.234 for those without. This again could show another misconception about CS, that those with no previous experience assume that it is a more male-dominated field.

Those with previous experience agreed more strongly that CS can be done without computers (Q11 - Table 7.18). This gap was narrowed statistically by the end of the

36 `https://content.yudu.com/web/4304h/0A4305u/UGHandbook2019/html/index.html`

Table 7.17: Q10 Boys/men are more likely to study CS than girls/women

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	3.044 (n = 114)	3.105 (n = 114)	T-Score = 0.385 P-Value = 0.701
Female	3.606 (n = 33)	3.606 (n = 33)	T-Score = 0 P-Value = 1
T-Test Result Male v Female	T-Score = 2.285 P-Value = 0.024	T-Score = 2.233 P-Value = 0.027	
Studied OL Maths in Secondary School	3 (n = 39)	2.923 (n = 39)	T-Score = 0.249 P-Value = 0.804
Studied HL Maths in Secondary School	3.221 (n = 104)	3.308 (n = 104)	T-Score = 0.547 P-Value = 0.585
T-Test Result OL v HL	T-Score = 0.93 P-Value = 0.354	T-Score = 1.798 P-Value = 0.074	
Had previous programming experience	3 (n = 70)	3.2 (n = 70)	T-Score = 0.943 P-Value = 0.347
Had no previous programming experience	3.325 (n = 77)	3.234 (n = 77)	T-Score = 0.486 P-Value = 0.628
T-Test Result No exp v Had exp	T-Score = 1.565 P-Value = 0.119	T-Score = 0.177 P-Value = 0.859	

Table 7.18: Q11 Work in Computer Science can be done without a computer

Demographic	Survey 1 Average	Survey 2 Average	T-Test Result (survey 1 v survey 2)
Male	2.895 (n = 114)	3.289 (n = 114)	T-score = 2.746 P-Value = 0.007
Female	2.756 (n = 33)	2.788 (n = 33)	T-score = 0.128 P-Value = 0.899
T-Test Result Male v Female	T-score = 0.639 P-Value = 0.524	T-score = 2.455 P-Value = 0.015	
Studied OL Maths in Secondary School	2.872 (n = 39)	3.103 (n = 39)	T-score = 0.981 P-Value = 0.329
Studied HL Maths in Secondary School	2.856 (n = 104)	3.212 (n = 104)	T-score = 2.347 P-Value = 0.019
T-Test Result OL v HL	T-score = 0.0779 P-value = 0.938	T-score = 0.547 P-value = 0.585	
Had previous programming experience	3.02 (n = 70)	3.3 (n = 70)	T-score = 1.488 P-Value = 0.139
Had no previous programming experience	2.714 (n = 77)	3.065 (n = 77)	T-score = 2.079 P-Value = 0.039
T-Test Result No exp v Had exp	T-score = 1.769 P-value = 0.079	T-score = 1.358 P-value = 0.177	

course but still exists. Alongside this is the fact that males also agreed more strongly that problem solving is central to CS as well as that CS can be done without a computer (Q9 - Table 7.16). This could show that the pen and paper exercises as well as theoretical parts of the course impact males more strongly than females.

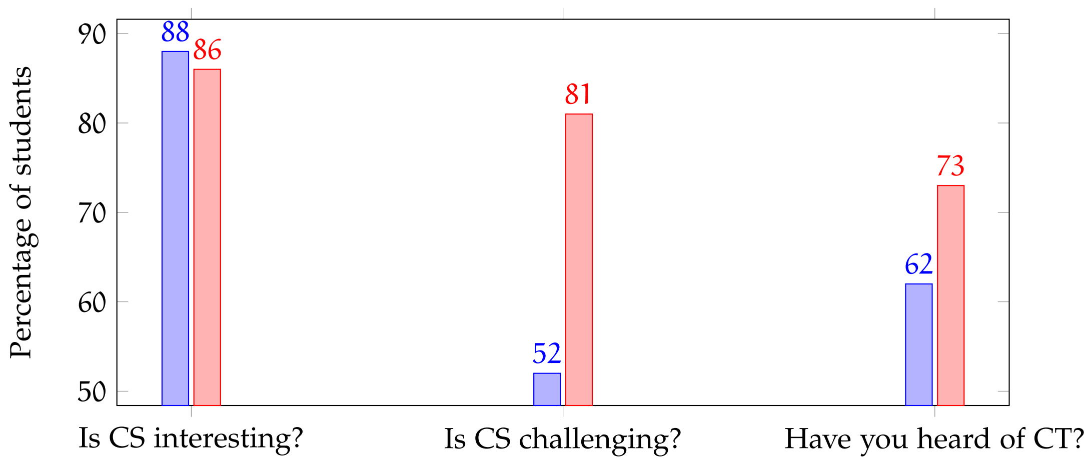


Figure 7.7: Higher Level Maths responses -Percentage of Yes responses

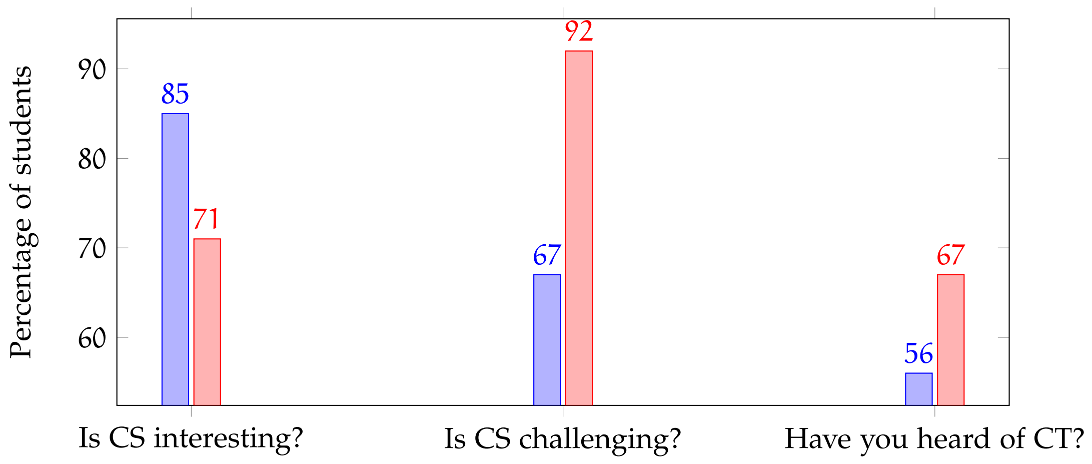


Figure 7.8: Ordinary Level Maths - Percentage of Yes responses

Of note is the responses to Q1, Q2 & Q12 in relation to the level of maths studied shown in Figure 7.7 and Figure 7.8. In these figures the blue (left-hand) bar is the for survey 1 and the red (right-hand) bar is for survey 2. More of those who studied OL initially saw CS as challenging in survey 1 compared to their HL peers (67% for OL compared to 52% for HL). Both percentages increased from survey one to two, but the gap from OL to HL is still present. OL responded "Yes" 81% compared to 92% for HL in survey 2. The drop in those who found CS interesting in survey two is larger in those who studied OL compared to those who studied HL. OL "Yes" responses dropped from 85% in survey 1 to 71% in survey 2 compared to 88% to 86% for HL. CT is also clearly more widely understood by those who studied HL, with 62% having heard of it before the course compared to 56% for those who studied OL.

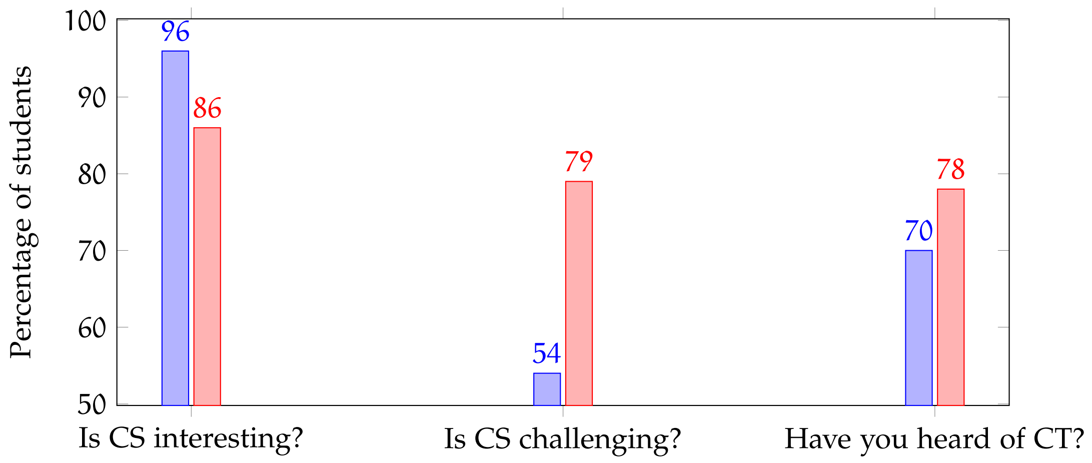


Figure 7.9: Previous programming experience - Percentage of Yes responses

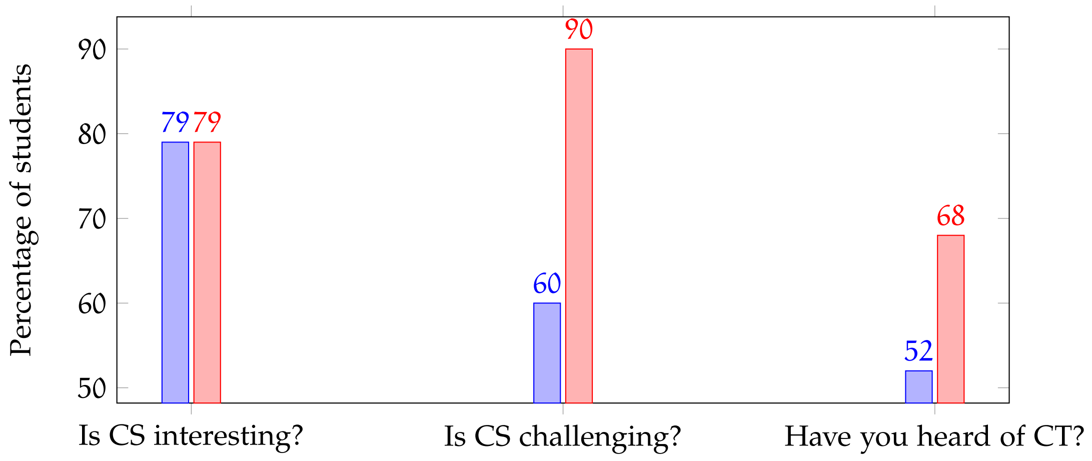


Figure 7.10: No Previous programming experience - Percentage of Yes responses

The answers for Q12 are presented in Figures 7.9 and 7.10 for the two groups: with and without previous programming experience. As previously, in these figures the blue (left-hand) bar is the for survey 1 and the red (right-hand) bar is for survey 2. shows that CT is more widely known by those with previous experience of CS (70% compared to 51.9% for those who had no previous experience). This is to be expected as it is a skill that is not yet widely taught in other subjects. Those with no previous experience didn't change in terms of finding the course interesting (Q1) with 79.2% finding it interesting in both surveys. However, there was a significant gain in those who found it challenging (Q2), this was true in both groups (30% increase for those with no previous experience and 24% increase for those with experience).

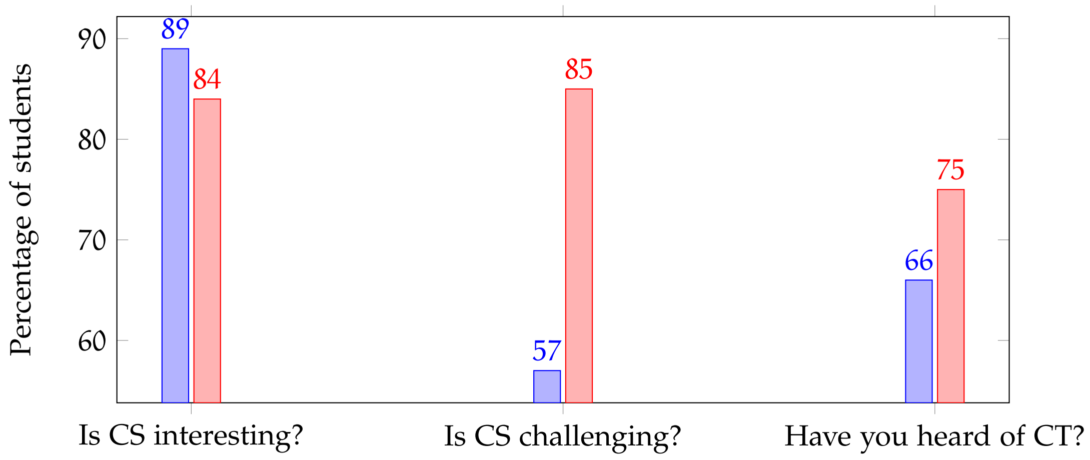


Figure 7.11: Male - Percentage of Yes responses

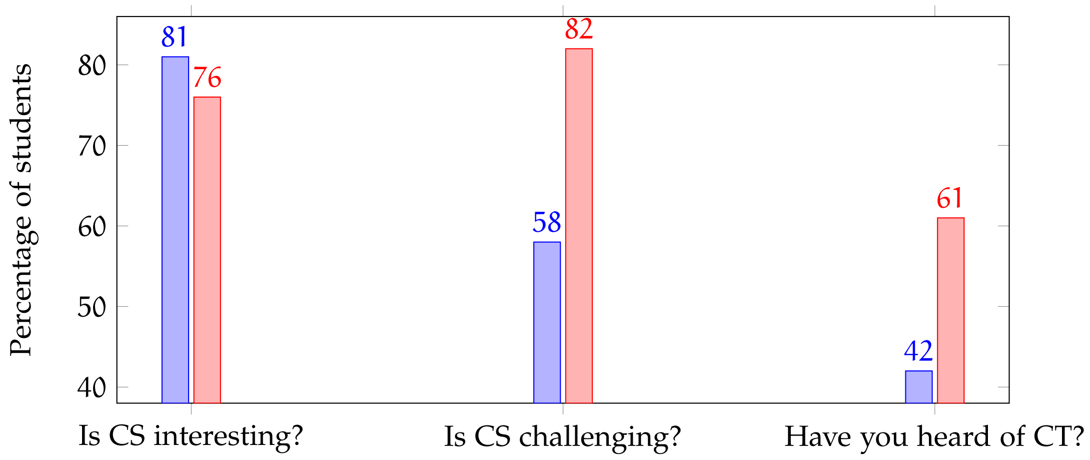


Figure 7.12: Female - Percentage of Yes responses

Looking at the responses to Q1, Q2 and Q12 for the gender groups in Figures 7.11 and 7.12, there isn't much of a difference between both groups except in Q12. Again, the left-hand or blue bar represents survey 1, with the right-hand or red bar representing survey 2. Here we find that males are much more likely to have heard of CT prior to the course (65.8% compared to 42.4%).

7.5 DISCUSSION

The Results sections provides a large amount of qualitative data on the students' scores and results from both the CT Assessment and the Views of CS survey. In this section we will briefly present and discuss a few findings that we believe are of most interest.

7.5.1 MATHS LEVEL AND A LINK TO CT ASSESSMENT SCORES

It has been shown that students with stronger mathematical abilities often do better in introductory programming courses [79]. These links can be seen when it comes to students' scores in our CT assessment. This is interesting from two points. Firstly, the fact that students who studied Higher Level maths did better in both Test 1 and Test 2, compared to their Ordinary Level peers, can help to support the fact that our test, is challenging students in the right areas. The Bebras problems are designed to test Computational Thinking and students who take Higher Level maths would, in general, be stronger in maths than their Ordinary Level peers. One of the hopes of the LC Maths curriculum is to teach students to be critical thinkers and creative problem solvers[37]. Higher Level maths students will need to be stronger in these areas (and in Computational Thinking) to cope with the harder curriculum. With that being said, students who studied Higher Level maths should perform better in the CT Assessment, which is encouraging to find in the data.

Secondly, it has been shown that students Mathematical abilities are a factor when predicting success when learning to program [79]. The fact that our test shows a significant difference between stronger and weaker students in maths could allow it to help as a predictor for success. More analysis is required in this area, including obtaining students Leaving Certificate maths results as well as their first-year grades in both CS and Mathematics. If this CT Assessment could be shown to help predict success or failure when learning to programme, this would allow strategic and timely interventions to be recommended and made available to students.

37 https://www.curriculumonline.ie/getmedia/f6f2e822-2b0c-461e-bcd4-dfcde6decc0c/SCSEC25_Maths_syllabus_examination-2015_English.pdf

7.5.2 people who haven't programmed before have misconceptions about what cs is

As shown in Section 7.4.2 people who have no previous programming experience may have misconceptions of what CS is. They agreed more strongly that installing software and using software like Word and Excel were central to CS (Table 7.11), that men are more likely to study it than women (Table 7.16) and that a Computer Scientist doesn't have to be good at working with others (Table 7.17). A higher percentage also found CS less interesting than those who had previous experience with programming (Figure 7.9 and 7.10). As previously discussed, CS has one of the highest drop-out rates of all university courses. We believe that is partly down to students not being aware of what the subject is before taking it. Students are rarely introduced to the topic through formal education at either primary or secondary schools. This is changing but there is still a lot of work to be done to introduce students through both formal and informal sectors. With the introduction of CS to the Leaving Certificate, the coding short course at Junior Cycle level, and more teachers and schools wanting to introduce some form of CS, these misconceptions could be altered sooner. This could also include the misconceptions about genders and computer scientists working alone. Those who hadn't studied programming before agreed more strongly that men were more likely to study CS and that a computer scientist doesn't have to be good at working with others.

7.5.3 the gender question

As with all STEM subjects, there is a major push in CS to improve the uptake of female students across all levels of education and also in industry. Female students generally make up a small minority of CS classes and this study has shown that with the significantly smaller number of female respondents. Of interest is the answers students gave to Q10 of the Views of CS survey shown in Table 7.17. This shows that, to begin with, female respondents felt that CS is a subject that males are more likely to study than females. This could be for a number of reasons but one likely reason is that fewer female students are introduced to CS in primary or secondary school. This can lead females (and males) to have stereotyped views based on media portrayals of CS-related jobs, teachers' misunderstandings of CS as a subject and many others. There are currently many initiatives to encourage girls to get involved with CS and coding and hopefully these, along with the new Leaving Certificate subject, will allow more students to have an opportunity to discover CS for themselves.

7.6 future work

7.6.1 re-run the tests next year

The tests described in this chapter are reasonably easy to administer in the university CS course setting as we have described. Students have assigned lab times, and this allows us to obtain data from many students with little effort in terms of scheduling and timetabling. We aim to run these tests again on next year's undergraduate students to compare and evaluate them more. This will allow us to see whether the findings discussed in this chapter are specific to this cohort or more representative of the "normal" first year group.

7.6.2 compare to a more consistent cohort of secondary school students

One plan for this study was to compare the first year undergraduate CS students to the Transition Year students who were taking the same assessments as part of the larger study. As described in Chapter 1, CS2Go has been developed to introduce secondary school students to CS concepts, including programming. The lessons are taught in a different style from the lectures that the undergraduate students attend but a lot of the same concepts are covered with the secondary school students getting a taster of what the undergraduates do in more depth. It would be interesting and informative to both CS2Go and the CS degree at Maynooth University to see how each course impacts participants' views of CS and their CT skills. This could influence the teaching methodologies and allow both to be improved.

The reason this did not come about is due to the lack of secondary school students taking both tests and taking the course for an extended period. However enough data will have been collected to perhaps allow a small comparison in the near future. One of the hopes of the project is to have a more consistent cohort of secondary school students using the content in a more structured and time-managed way, for example 15-20 hours over a 6-12-week period. We hope to have around 120 students taking part in this study, this would provide enough data to allow us to compare the two groups.

7.6.3 COMPARE TO ANOTHER THIRD-LEVEL INSTITUTION

Another interesting comparison would be with another third-level institution. This could be with one who teaches in a similar setting e.g. lectures and labs or one with a different setup e.g. online or distance learning. These different teaching styles are both very common and with online courses being pursued more by institutions, including Maynooth University, showing that they can have a similar effect on CT skills could be something to encourage this mode of learning. One drawback from distance learning can be the lack of critical thinking and debate that happens between peers, and this is something that needs to be encouraged in these courses [12]. If this can be seen with our tests, then interventions could be developed e.g. by providing problem-solving tasks that can be done as a group online.

Of interest would be comparing to other institutions that use Java as a first programming language and those who use a different one (e.g. Python). Much discussion has been had over recent years over which programming language is the "best" as an introductory language, and whilst all have their own merits, it would be interesting to see if one appears to impact students CT skills more than another.

7.7 IMPACT OF THIS STUDY ON CS2GO

One of the hopes of this study, as well as the findings from the students feedback and surveys themselves, was that it could influence the development of the CS2Go course. The most obvious way is the larger sample size for the Computational Thinking assessment that completed it compared to the secondary school cohort. This has allowed us to show that the test can work in both the time-frame desired and it appears to test the right skills; this is evident from the fact those who had previously programmed and studied Higher Level mathematics performed better than their peers. Additionally, the response to Q10 ("Boys/men are more likely to study CS than girls/women") in the view of CS survey, summarised in Section 7.5.3, provoke some interesting questions that warrant further study and research. As shown in Table 7.17 there is a significant difference between male and female participants response to the question "Boys/men are more likely to study Computer Science than girls/women". Males tended to disagree more strongly with the statement going into the course and at the courses conclusion. They responded with an average of 3.044, compared to 3.606 for Females. From survey 1 to survey 2 males opinions shift slightly, but not significantly, towards agreement (3.044 in survey 1 to 3.105 for survey 2), whereas female's views didn't change at all on average (3.606 in both surveys).

It would be worth exploring whether students are well-informed on the gender imbalance in CS (and STEM more widely) and whether this would then lead to less female students taking courses in CS? Additionally, are students simply making observations based on the demographics within their current class setting or is there a wider stereotype associated with CS that leads it to be a male-dominated subject of study and work? This response could have many influences and is likely to be a combination of many factors, but we believe introducing all students to CS early on in the academic career can allow us to show both male and female students that anyone can study and succeed in CS and also in CS-related courses and jobs. It can be seen from responses to a number of questions in the Views of CS survey that there are misconceptions about what CS is from those who haven't programmed before, this is summarised in Section 7.5.2.

Part IV

CONCLUSIONS

8 DISCUSSION AND FUTURE WORK

8.1 CONCLUSIONS

8.1.1 COURSE CONTENT

One of the main goals of this research, and the main deliverable, is the generation of a Computational Thinking course and associated content. This includes lesson plans, printable exercise sheets, slides and teacher guides. In total, over 80 hours of content have been created and the majority has been tested in schools. The feedback in general has been very positive both from those students taking the course and from the teachers who have used and delivered the content, as well as anecdotal feedback from educators and others who have viewed or been shown the content.

The content has been developed with flexibility and adaptability in mind for teachers whilst also reaching as many of the learning goals of the newly developed Leaving Certificate curriculum specification as possible. This can be seen in Appendix O. From feedback from teachers we believe this has been achieved and teachers have generally given positive feedback on the lesson plan layouts and the exercises and activities contained in them.

The development of a designed-for-purpose web system allows us to continue to develop and promote course content with full control of the site. The system will also allow us to collect user data far more easily than was possible during the course of this study. The course will become live this term (Autumn 2018) and will be expanded on and improved over the coming academic year (2018-19).

8.1.2 PROBLEM SOLVING ASSESSMENT

One of the major developments from this research is the development of an assessment for Computational Thinking skills. This is based on Bebras problems and was described in detail in Chapter 4. Although in the early stages of development, this assessment could be of great benefit to educators and teachers when trying to test this "new" skill of Computational Thinking. Very little work has been done in this domain as yet and this work could help to contribute to more robust and comprehensive assessments being developed in the future.

8.1.3 SCHOOL STUDY

As described in Chapter 6, the school study gave mainly positive results when looking at the course content and the perceived and actual learning of students. One of the main disappointments was that a more extensive study wasn't possible. Ideally a selection of schools/classes would have taken the pre-surveys and post-surveys and problem-solving tests whilst being taught using a number of the course lessons and activities. What we can say is that the lessons are generally enjoyable, based on both student and teacher feedback as well as anecdotal data and observations. The lessons are also very usable and comprehensive for teachers. They also appear to teach students CS topics and concepts based on student responses to questionnaires given at the end of lessons.

From the one structured group of students' (see Section 6.3.1.1), it was observed that student's views of what CS is and what is central to CS are impacted by using the CS2Go lessons. This is a positive outcome as correcting and informing students of the misconceptions surrounding CS is one of the hopes of this course. In the same section, promising results were shown in the Computational Thinking Assessment. Students' scores increased by the end of the course, from 7.25 in the first assessment to 8.294 in the second (see Table 6.20). This was only a small sample but it is a positive result and with further research and usage of both the CS2Go lessons and the Computational Thinking Assessment we can obtain a better understanding of what areas the lessons impact and what areas are lacking in the content. With this knowledge we can expand upon and adapt the material to improve the educational experience for students.

8.1.4 FIRST YEAR UNDERGRADUATE STUDY

The study of the first year undergraduate CS students was a very successful study and is described in detail in Chapter 7. Having a large cohort to test the Computational Thinking Assessment and the View of CS survey proved invaluable, especially when the the issues involved in working with schools and secondary school classes are taken in to account. From a practical standpoint it showed that the Computational Thinking assessment and View of CS survey are usable and can be administered easily to a large group of students.

There were also a number of interesting findings including evidence to support the case that there is a link between students problem-solving skills and their Maths ability. Students who studied Higher Level maths did better on both tests than their Ordinary Level peers. It was also seen that those who haven't studied CS previously have misconceptions about the subject. This includes thinking CS is heavily linked to installing software and that Computer Scientists don't have to be good at working with others. These misconceptions on what CS is could be down to a number of reasons but it is our belief that introducing students to CS earlier (in Primary or Secondary school) can change these misconceptions. This may help to lower the high drop-out rate in CS courses as students will be more aware of the subject and what it involves. There were also some findings related to gender views in CS, namely that males tended to disagree more strongly that "Boys/men are more likely to study Computer Science than girls/women" both at the beginning and end of the course. Males responded with an average of 3.044, compared to 3.606 for Females. From survey 1 to survey 2 males opinions shift slightly, but not significantly, towards agreement (3.044 in survey 1 to 3.105 for survey 2), whereas female's views didn't change at all on average (3.606 in both surveys).

As discussed further in both Chapter 7 and Chapter 8 the hope is to continue and expand this study over the coming academic year.

8.1.5 CONTRIBUTION TO THE STATE OF THE ART

The peer-reviewed publications that arose as a result of the research contained in this thesis are as follows:

J. Lockwood and A. Mooney. "Computational Thinking in Secondary Education: Where does it fit? A systematic literary review." In: *International Journal of Computer Science Education in Schools* 2018 (2018), pp. 41–60
This publication was a broad view of the current state of CT education in both the research and in practice. This review was more comprehensive than previous attempts and will allow researchers and educators to adapt their pedagogy, their research focus and efforts to fill the gaps and opportunities found.

J. Lockwood and A. Mooney. "Developing a Computational Thinking Test using Bebras problems". In: *TACKLE: the 1st Systems of Assessments for Computational Thinking Learning workshop at EC-TEL 2018*. Vol. 2018. 2018
CT assessment was on the key areas found to be lacking in the literary review

conducted as part of this research. The development of a assessment tool will allow educators and researchers to test students CT skills in a simple and repeatable way. It is a user-friendly assessment and doesn't require any specific tools or software which can allow it to be used widely and easily.

J. Lockwood and A. Mooney. "A Pilot Study Investigating The Introduction Of A Computer-Science Course at Second Level Focusing on Computational Thinking". In: *Irish Journal of Education* (2017), pp. 108–127
The presentation of the data from the pilot study conducted in this project shows the ease of use and positive feedback gained on the CS2Go content. This study influenced the continued development of CS2Go and allowed the content to be improved and expanded on.

8.2 FUTURE WORK

8.2.1 CONTINUE COURSE AND WEBSITE DEVELOPMENT

It is our belief that resources like CS2Go are going to be vital with the newly commenced Leaving Certificate course in Computer Science and further expansion of CS education in Ireland and worldwide. Teachers have mixed abilities in CS and therefore ready-to-go resources and well-explained lesson plans can help those without a strong background in CS to learn and teach the content effectively. Although a large quantity of lesson plans and resources have been developed there is huge scope to expand and add to the existing content. Examples include expanding the Web Development module, developing further modules such as App Development and also creating projects which align to the current four projects in the LC curriculum.

The CS2Go Web Portal will be a great tool for promoting, distributing and assessing the course content and those who use it. This first iteration is very much a starting point with the hope to continue to develop the web system. Improvements could include turning the lesson plans, which are currently uploaded as PDF's, into interactive and customisably downloaded PDF's. This would allow teachers to pick the various activities in a lesson they want to teach and download a self-created PDF. Other improvements could include forums, a space for teachers to upload resources and the ability for teachers to rate the activities. It is our hope that CS2Go becomes a portal and one-stop shop for CS teachers.

8.2.2 RE-DO THE FIRST YEAR UNDERGRADUATE STUDY AND EXPAND IT

Using the first year undergraduate course as a test sample proved useful in assessing the various assessment forms for CS2Go. It is our plan to re-run these tests in the next academic year to help validate the results we've found. It would also be interesting to compare the CS undergraduates to another cohort, such as an English or Music class, to see whether we can show CS has any significant impact on the metrics assessed.

We also plan to run the assessment in other third-level institutes as well with their CS students. This would again help to validate any results as well as show any differences between teaching methods that might exists such as programming language used or teaching methods employed.

8.2.3 STRUCTURED SECONDARY SCHOOL STUDY

One of the major disappointments of this study is that a reasonably large sample of secondary school students didn't complete the pre-assessment and post-assessment either side of a considerable exposure to the lesson content (e.g. 20 hours). There has been interest from a number of teachers to take part in a study like this and so it is our plan to run some training sessions before the commencement of the next school year and give the teachers a lesson timetable for them to follow. This will allow most students to get a similar experience and allow us to group results together. Alongside a control group, this could allow us to show that the content included in CS2Go impacts (or doesn't impact) the various metrics tested for.

A QUALITY ASSESSMENT OF PAPERS FOR LITERATURE REVIEW

Table A.1: Quality assessment of papers

Reference	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Total**
[1]	Y Y	Y	Y	N	4	
[3]	Y	Y	Y	P	P/Y	4/4.5
[4]	Y	Y	Y	Y	P	4.5
[5]	Y	Y	N	Y	P	3.5
[6]	P	Y	Y	Y	Y	4.5
[10]	Y	Y	Y	Y	Y	5
[11]	Y	P	N/A	Y	P	3
[15]	Y	Y	N/A	Y	N	3
[16]	Y	Y	N/A	P	P	3
[18]	Y	Y	P	Y	P	4
[19]	P	Y	Y	Y	P	4
[20]	Y	Y	Y	Y	N	4
[23]	Y	Y	Y	Y	P	4.5
[22]	Y	Y	P	Y	N	3.5
[25]	Y	Y	Y	Y	Y	5
[28]	Y	Y	Y	Y	P	4.5
[29]	Y	Y	P	Y	P	4
[30]	Y	Y	N	Y	P	3.5
[35]	Y	Y	P	Y	P	4
[34]	Y	Y	Y	Y	Y	5
[36]	Y	Y	Y	Y	Y	5
[37]	Y	Y	Y	Y	Y	5

Continued on next page

Table A.1 – continued from previous page

Reference	Q1	Q2	Q3	Q4	Q5	Total
[39]	P	Y	Y	Y	Y	4.5
[44]	Y	Y	Y	Y	N	4
[46]	Y	Y	Y	Y	N	4
[47]	Y	Y	Y	Y	P	4.5
[49]	Y	P	Y	Y	P	4
[50]	Y	Y	P	Y	Y	4.5
[52]	Y	Y	Y	Y	N	4
[53]	P	Y	N/A	Y	P	3
[54]	P	Y	P	Y	N	3
[55]	Y	Y	Y	Y	Y	5
[63]	Y	Y	Y	Y	N	4
[64]	P	Y	Y	Y	P	4
[65]	Y	P	N	Y	P	3
[66]	Y	Y	Y	Y	P	4.5
[68]	Y	Y	P	Y	P	4
[69]	Y	Y	P	Y	P	4
[70]	Y	Y	N	Y	P	3.5
[71]	Y	Y	Y	Y	Y	5
[76]	Y	Y	P	Y	P	4
[80]	Y	Y	N/A	Y	Y	4
[81]	P	Y	N/A	Y	Y	3.5
[83]	Y	Y	P	Y	P	4
[85]	P	Y	N/A	P	Y	3
[86]	Y	Y	P	Y	Y	4.5
[87]	P	Y	N/A	Y	P	3
[90]	Y	Y	Y	Y	Y	4.5
[92]	Y	Y	Y	Y	P	4.5
[93]	Y	Y	N/A	Y	Y	4

Continued on next page

Table A.1 – continued from previous page

Reference	Q1	Q2	Q3	Q4	Q5	Total
[94]	Y	Y	Y	Y	P	4.5
[96]	Y	Y	Y	Y	Y	5
[99]	Y	Y	Y	Y	P	4.5
[100]	Y	Y	Y	Y	Y	5
[101]	Y	Y	P	Y	P	4
[102]	Y	Y	Y	Y	Y	5
[103]	Y	Y	Y	Y	Y	5
[105]	Y	Y	N	Y	N	3
[106]	Y	Y	Y	Y	Y	5

- Question 1: Y (yes), the findings are very credible due to where the study is published, P (partly), the findings are credible but the source is questionable, N (no), the findings not the source are credible.
- Question 2: Y (yes), the evaluation addresses the original aims and objectives, P (partly), the evaluation addresses the original aims and objectives implicit, N (no), the evaluation does not address the original aims and objectives.
- Question 3: Y (yes), the data collection was carried out well and outlined clearly, P (partly), the data collection was carried out well but not outlined clearly, N (no), the data collection was not carried out well.
- Question 4: Y (yes), the paper was clear and coherent, P (partly), the paper was somewhat clear and coherent, N (no), the paper was not clear and coherent.
- Question 5: Y (yes), knowledge or understanding has been extended, P (partly), knowledge or understanding have been somewhat extended, N (no), knowledge or understanding has not been extended.

The scoring used was as follows. Where a question received a Y a score of 1.0 was applied, where a P was received a score of 0.5 was applied and where a N was received a score of 0.0 was applied.

B

EXTERNAL RESOURCES USED IN CS2GO

https://classic.csunplugged.org/searching-algorithms/#Battleships

https://classic.csunplugged.org/finite-state-automata/

https://classic.csunplugged.org/graph-colouring/

https://classic.csunplugged.org/cryptographic-protocols/

https://classic.csunplugged.org/public-key-encryption/

https://classic.csunplugged.org/information-hiding/

https://classic.csunplugged.org/dominating-sets/

https://bit.ly/2xWP39E (Locked In Activity)

https://teachinglondoncomputing.org/the-tour-guide-activity/

https://bit.ly/2R29Ukv (The Knights Tour Activity)

https://bit.ly/2OVlnkb (The Box Variable Activity)

https://bit.ly/2xWPfpo (The Assignment Dry Run Activity)

https://bit.ly/2N6u8Gn (Spit-Not-So Activity)

https://studio.code.org/s/course3/stage/1/puzzle/1

https://researchparent.com/coding-a-lego-maze/

https://makecode.microbit.org/courses/csintro/iteration/unplugged

C

CONCEPTS AND INTRODUCTION LESSONS

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Introduction to Computer Science

Strand: Computer Science Concepts *Duration of Lesson: 40 mins - 1 hour*

Learning Outcomes

Students will be able to define, more correctly, what Computer Science (CS) is.
Students will be able to describe the role of a "Computer Scientist".

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.2 explain how the power of computing enables different solutions to difficult problems

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.7 develop algorithms to implement chosen solutions

1.8 evaluate the costs and benefits of the use of computing technology in automating processes

1.12 compare the positive and negative impacts of computing on culture and society

1.18 recognise the diverse roles and careers that use computing technologies

1.2 collaborate and assign roles and responsibilities within a team to tackle a computing task

Junior Certificate Coding Short Course

1.1 present and share examples of what computers are used for and discuss their importance in modern society and in their lives

1.3 explain how computers are devices for executing programs via the use of programming languages

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

2.7 identify a topic or a challenge in computer science that inspires them

Computational Thinking Skills used

Algorithms

Computer Science concepts/topics involved

Problem solving

Resources / Materials

Pen, paper etc.
A3 Paper
Colouring pencils
Locked-in activity sheet
Introduction to Computer Science Slideshow

The Class

Activity - Introduction – 10 minutes

Ask the students if they've heard of the term CS. Hopefully most have and have some sort of notion about what it is/isn't. In pairs/groups the students should write down everything they know about CS. As prompts you could write the following questions on the board/screen:

- What is CS?
- Who does/uses CS?
- How is CS used?
- What does CS involve
- Why is CS important?

(plus, the reverse of these)

Another interesting exercise is to get students to draw a Computer Scientist. This could allow you to talk about a number of misconceptions about who a Computer Scientist is, for example gender (most people will draw a man), nerdy etc.

Give the students 5-10 minutes to exhaust their ideas.

Discussion – Introduction - 5-10 minutes

Bring the class together and have a brief discussion about what they think and why they think these things. Most students will have misconceptions about CS, these are discussed in the slideshow, talk through them with the class.

Most students will think of programming when they hear the term CS. Hopefully they can expand a little more but it's true that most people envisage a person sitting typing at a computer all day as a Computer Scientist. Although programming is a major tool of a Computer Scientist it is not the whole field. CS is more about the problem and figuring out a way to solve it. Below are some examples of problems (credit code.org).

- Too much information produced by a study for one human to be able to sort through it all in a lifetime.
- An elderly person finds it hard to physically connect with long-distance relatives.

- An organisation that needs to find the best route to get the most miles out of its airplanes using the least amount of fuel possible.

Activity – Development – 5-10 minutes (can be cut if time is an issue)

The goal of a Computer Scientist is to use computers, and all the power and potential they have, to help solve these problems. Again, in pairs get students to come up with a problem that they think could be solved or has been solved by a Computer Scientist. Again, discuss them as a class.

Activity – Development – 10-20 minutes

Students are now going to get the opportunity to be Computer Scientists. They're going to solve the problem of communicating with someone who has "Locked-in syndrome" (credit: https://teachinglondoncomputing.org/resources/inspiring-unplugged-classroom-activities/the-locked-in-activity/).

A person with "Locked-in syndrome can't move any part of their body, but they are fully conscious. Usually they can still blink so this is what we're going to use to allow students to communicate. Split students into pairs/threes, one of them will be the patient and the others will be the doctor. Hand each patient a random word on a piece of paper (examples below), the aim is for the doctor to figure out the word, all the patient can do is blink. Give them a few minutes to try and see what ideas they come up with.

If most aren't making much progress call the class together and explain the following solution to the problem. You could go through the whole alphabet and ask the patient to blink if that is their letter. Repeat until they get the word. Get the students to try this method out and hopefully they'll all get the word in a few minutes!

Now the idea is to improve the solution, to make it quicker and easier for the person to communicate. The pairs can work together for a while to come up with a faster solution, some examples are below. Get them to swap roles and give out a sentence this time to each "locked-in" student and repeat the process (examples below).

Examples of words:

Technology
Excitement
Whiteboard
Notebook
Elephant

Examples of sentences:

"I am very hungry"
"Can you help me"
"I want to go home"
"Is it sunny outside"

Examples of ways to improve the process:

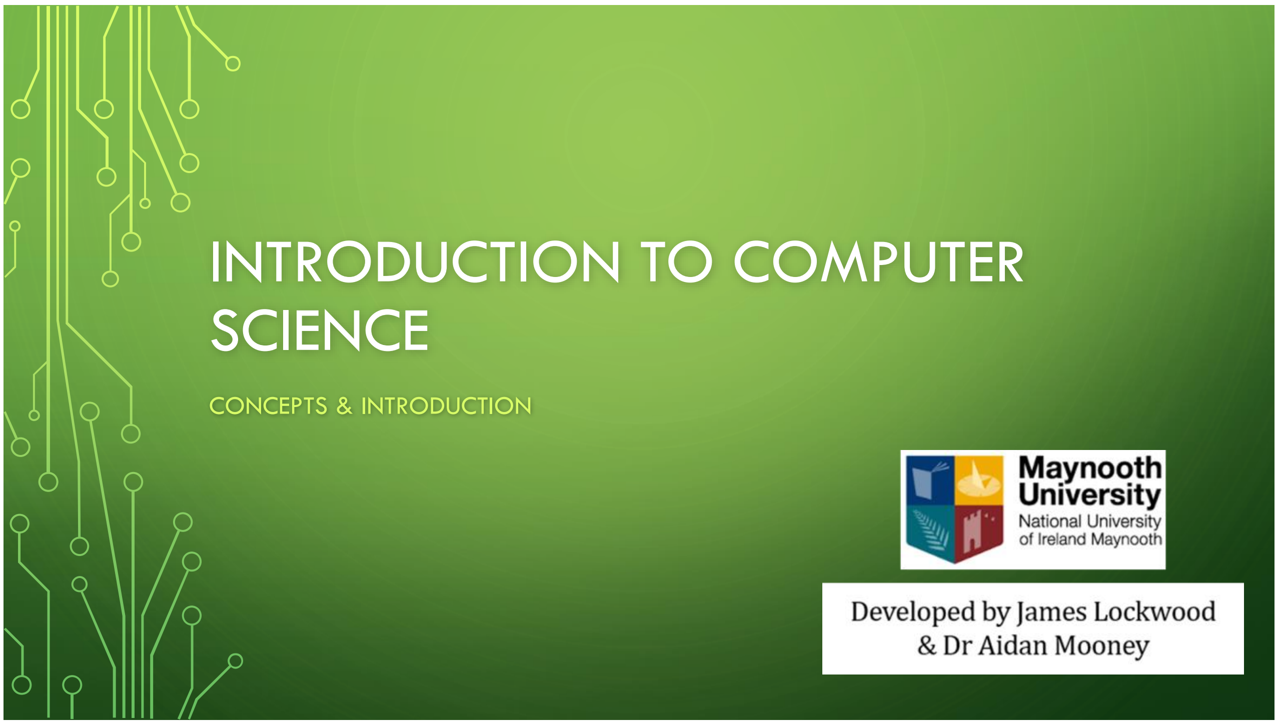
Get them to say whether it’s a vowel/consonant.
Is it near the beginning of the alphabet or the end?
Most common letters (E is the most common letter in English)
Write up the alphabet and point, for groups of 3 this could be quicker as they can have one move along the alphabet and the other looking at the person to see when they blink.
Linear Search – does it come after or before N? Then so on.

Discussion – Conclusion – 5 minutes

It’s important to point out that even though they didn’t use any computers or other technology, that they still acted as Computer Scientists! The way of thinking, which we’ll call Computational Thinking, is a vital part of what it means to be a Computer Scientist and can be used in all other parts of life! But we’ll talk more about what that means in the next class!

Feedback – Conclusion – 5 minutes

Please get your class to fill out the Computer Science feedback forms.

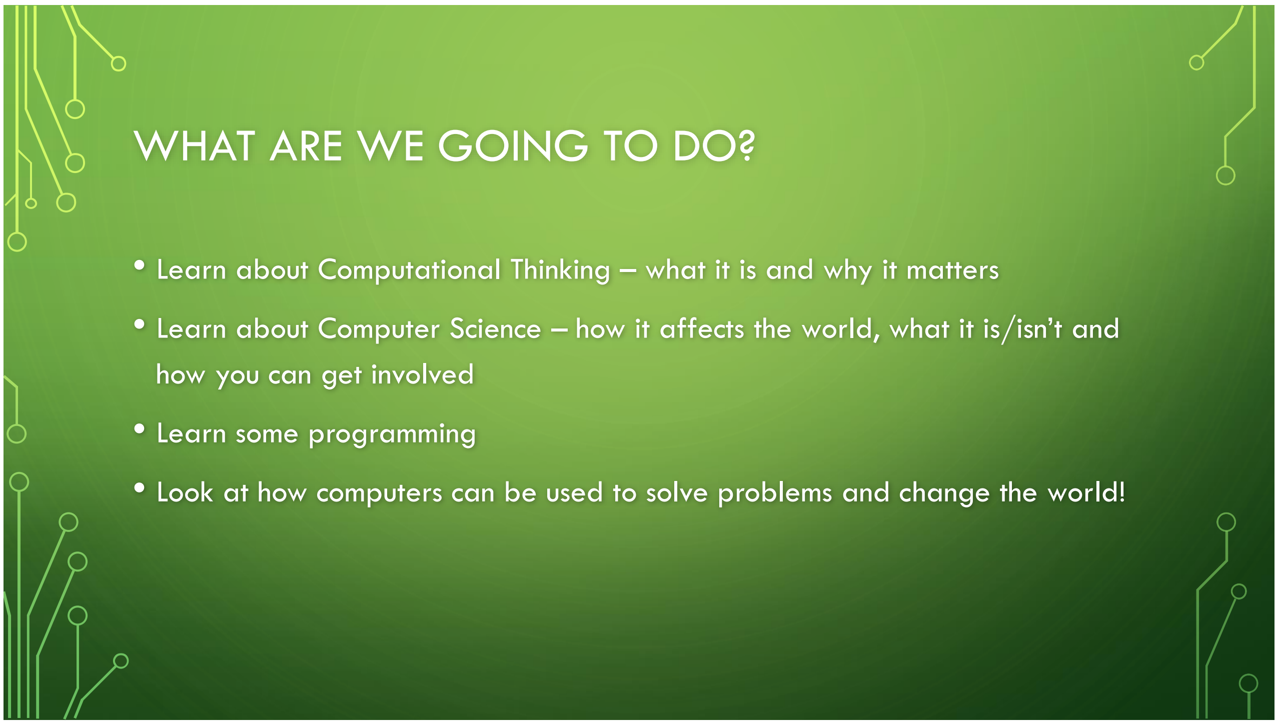


INTRODUCTION TO COMPUTER SCIENCE

CONCEPTS & INTRODUCTION

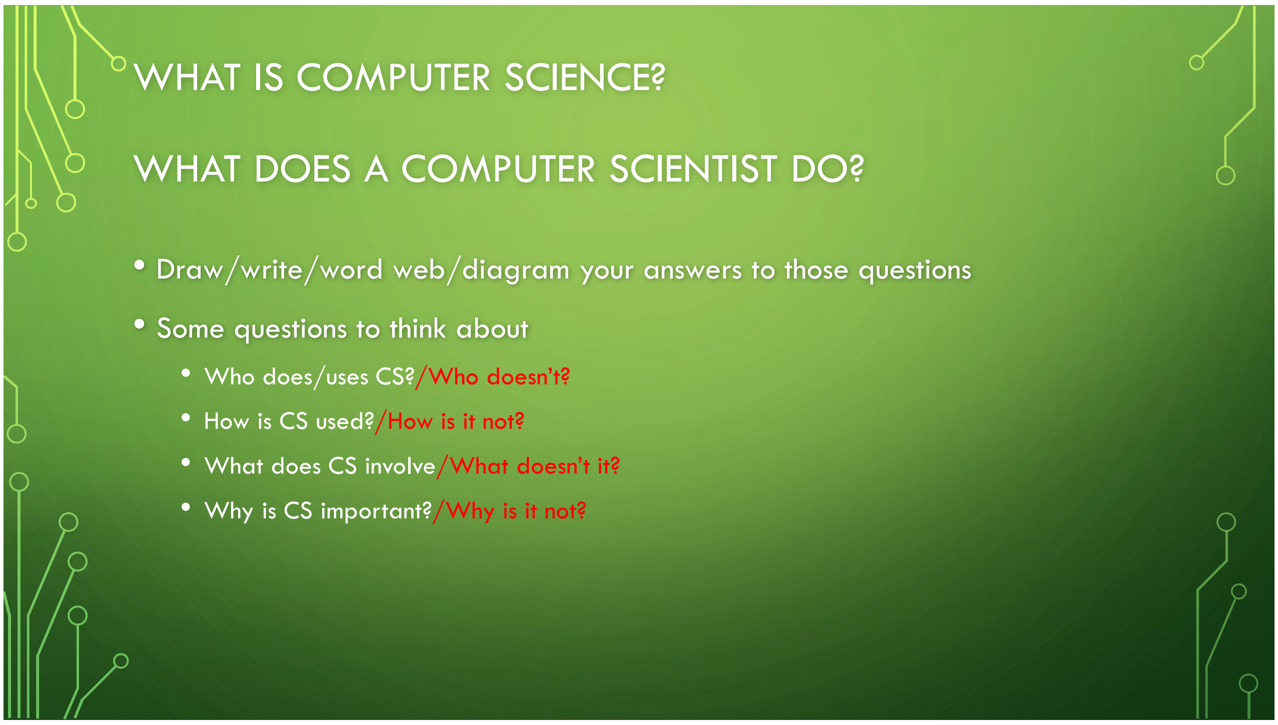
Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



WHAT ARE WE GOING TO DO?

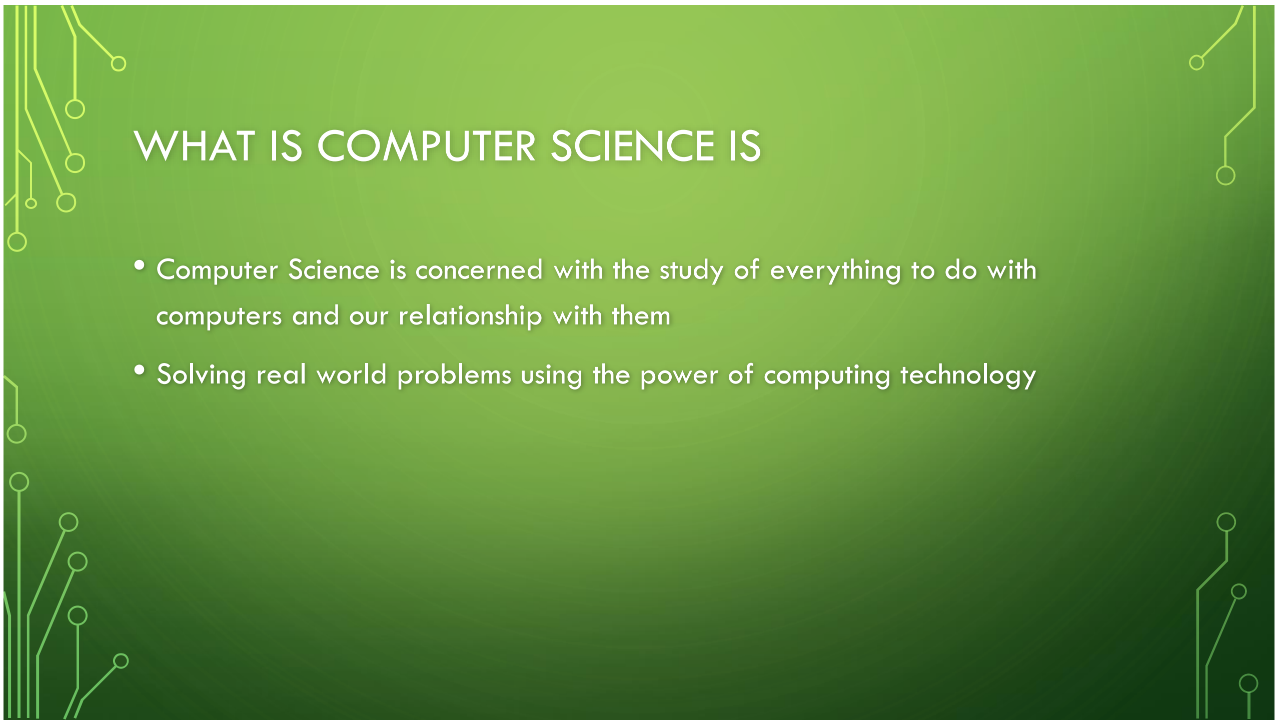
- Learn about Computational Thinking – what it is and why it matters
- Learn about Computer Science – how it affects the world, what it is/isn't and how you can get involved
- Learn some programming
- Look at how computers can be used to solve problems and change the world!



WHAT IS COMPUTER SCIENCE?

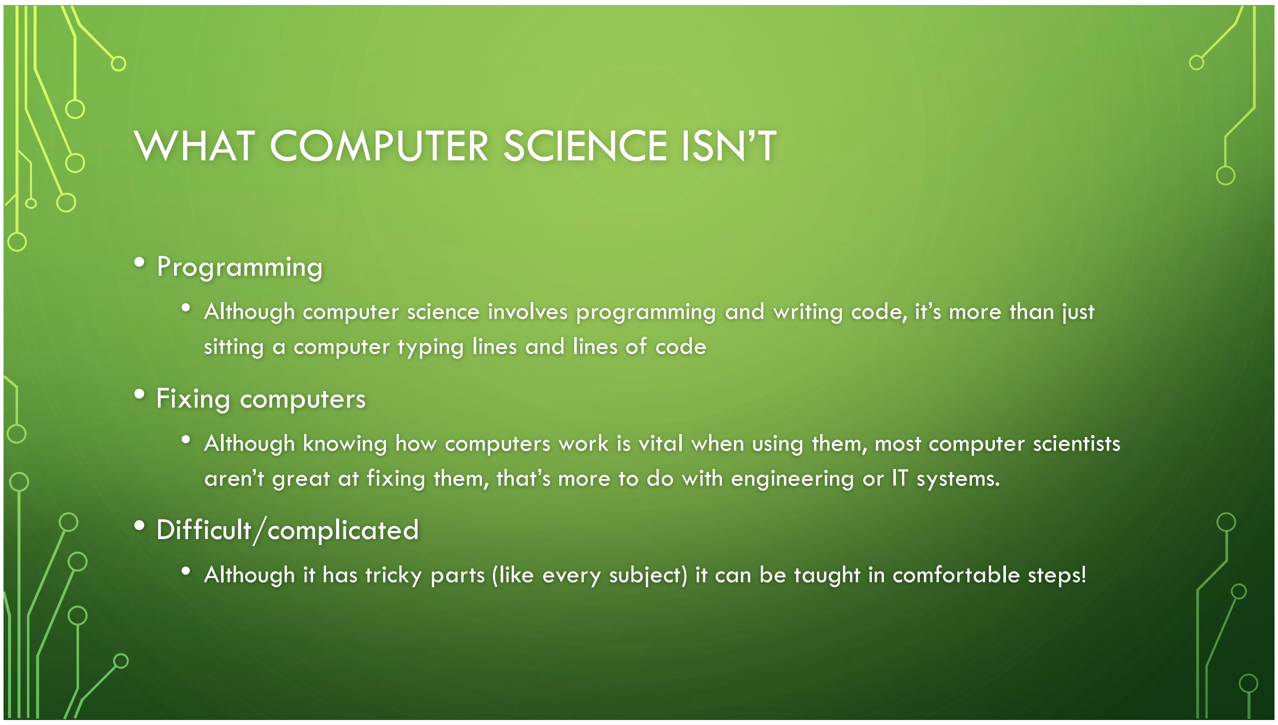
WHAT DOES A COMPUTER SCIENTIST DO?

- Draw/write/word web/diagram your answers to those questions
- Some questions to think about
  - Who does/uses CS?/Who doesn't?
  - How is CS used?/How is it not?
  - What does CS involve/What doesn't it?
  - Why is CS important?/Why is it not?



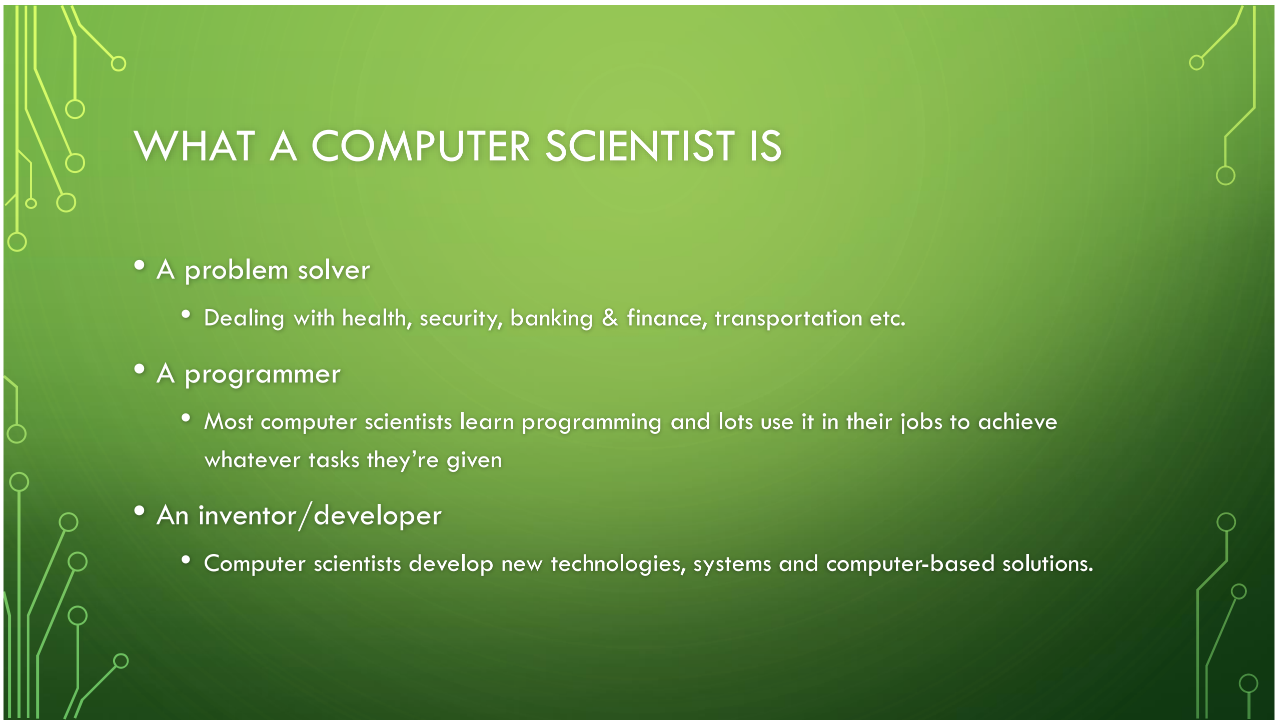
WHAT IS COMPUTER SCIENCE IS

- Computer Science is concerned with the study of everything to do with computers and our relationship with them
- Solving real world problems using the power of computing technology



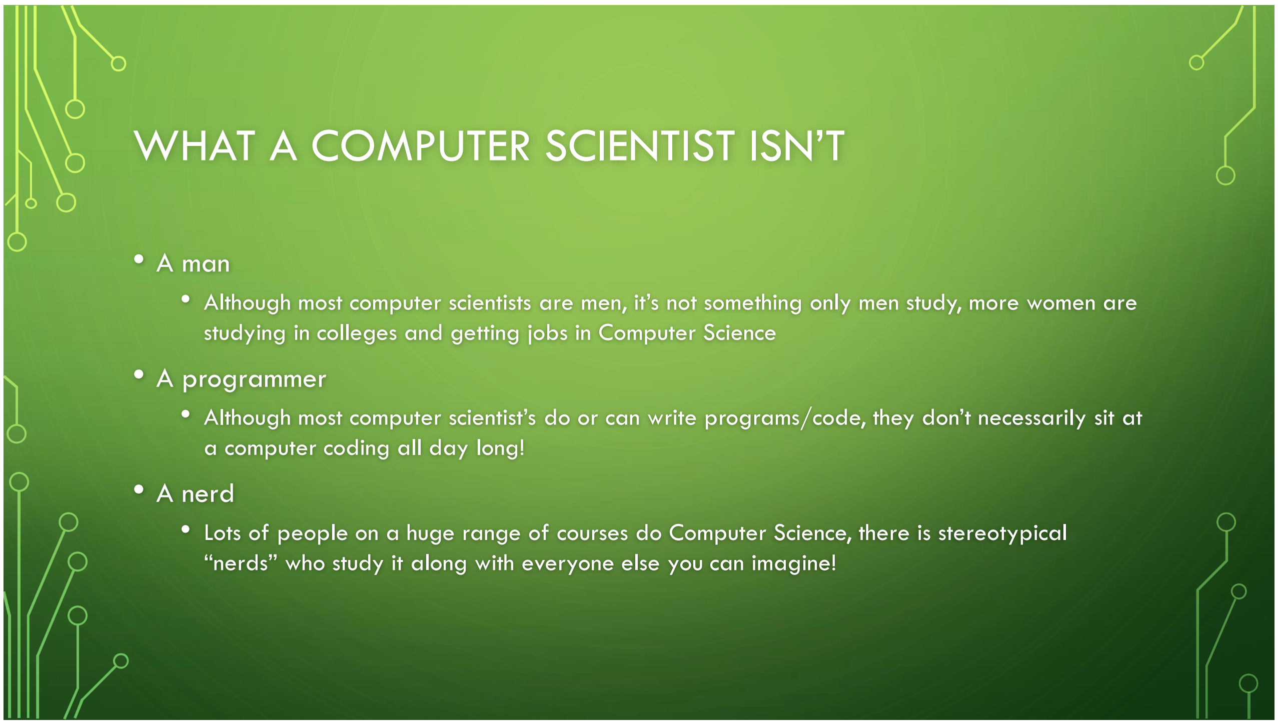
WHAT COMPUTER SCIENCE ISN'T

- Programming
  - Although computer science involves programming and writing code, it's more than just sitting a computer typing lines and lines of code
- Fixing computers
  - Although knowing how computers work is vital when using them, most computer scientists aren't great at fixing them, that's more to do with engineering or IT systems.
- Difficult/complicated
  - Although it has tricky parts (like every subject) it can be taught in comfortable steps!



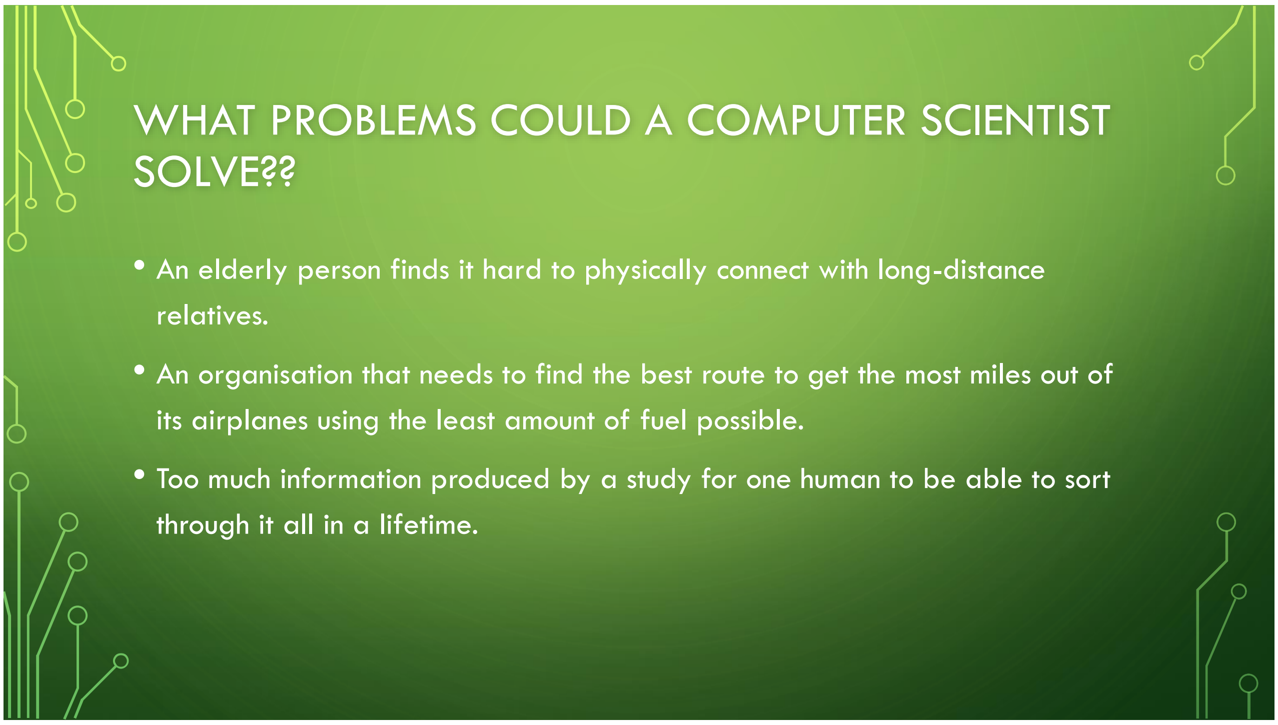
WHAT A COMPUTER SCIENTIST IS

- A problem solver
  - Dealing with health, security, banking & finance, transportation etc.
- A programmer
  - Most computer scientists learn programming and lots use it in their jobs to achieve whatever tasks they're given
- An inventor/developer
  - Computer scientists develop new technologies, systems and computer-based solutions.



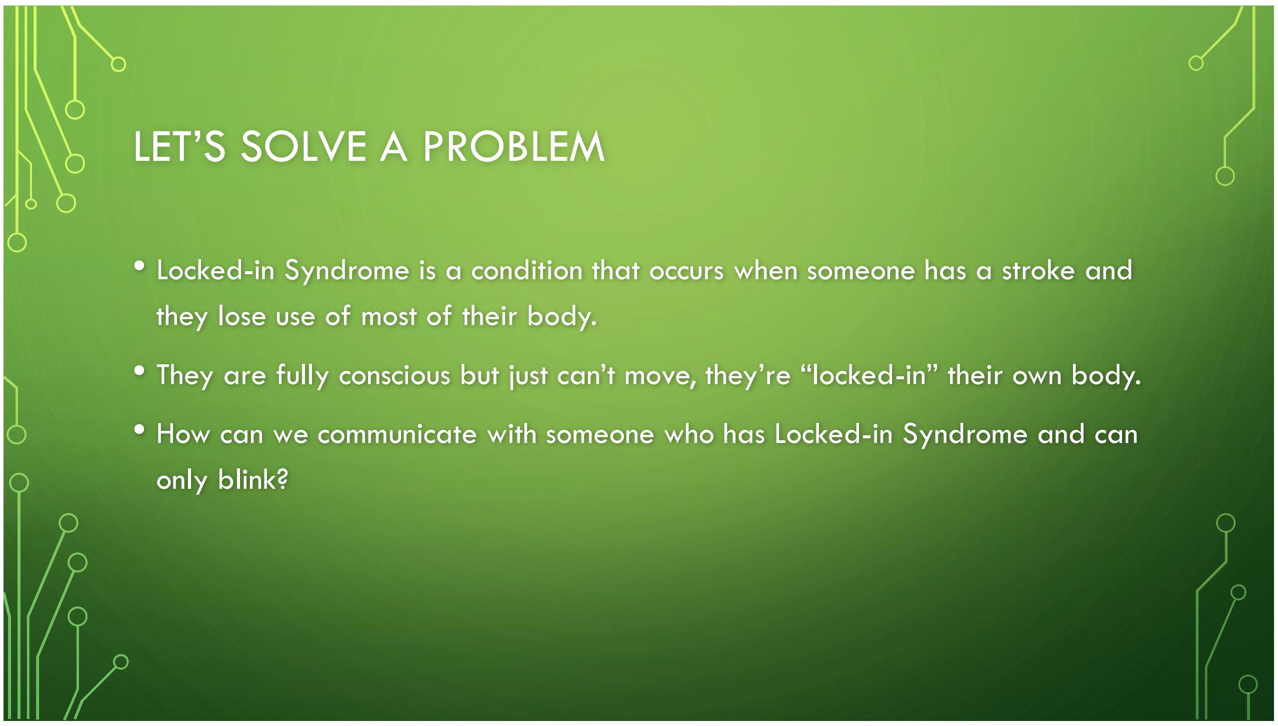
WHAT A COMPUTER SCIENTIST ISN'T

- A man
  - Although most computer scientists are men, it's not something only men study, more women are studying in colleges and getting jobs in Computer Science
- A programmer
  - Although most computer scientist's do or can write programs/code, they don't necessarily sit at a computer coding all day long!
- A nerd
  - Lots of people on a huge range of courses do Computer Science, there is stereotypical "nerds" who study it along with everyone else you can imagine!



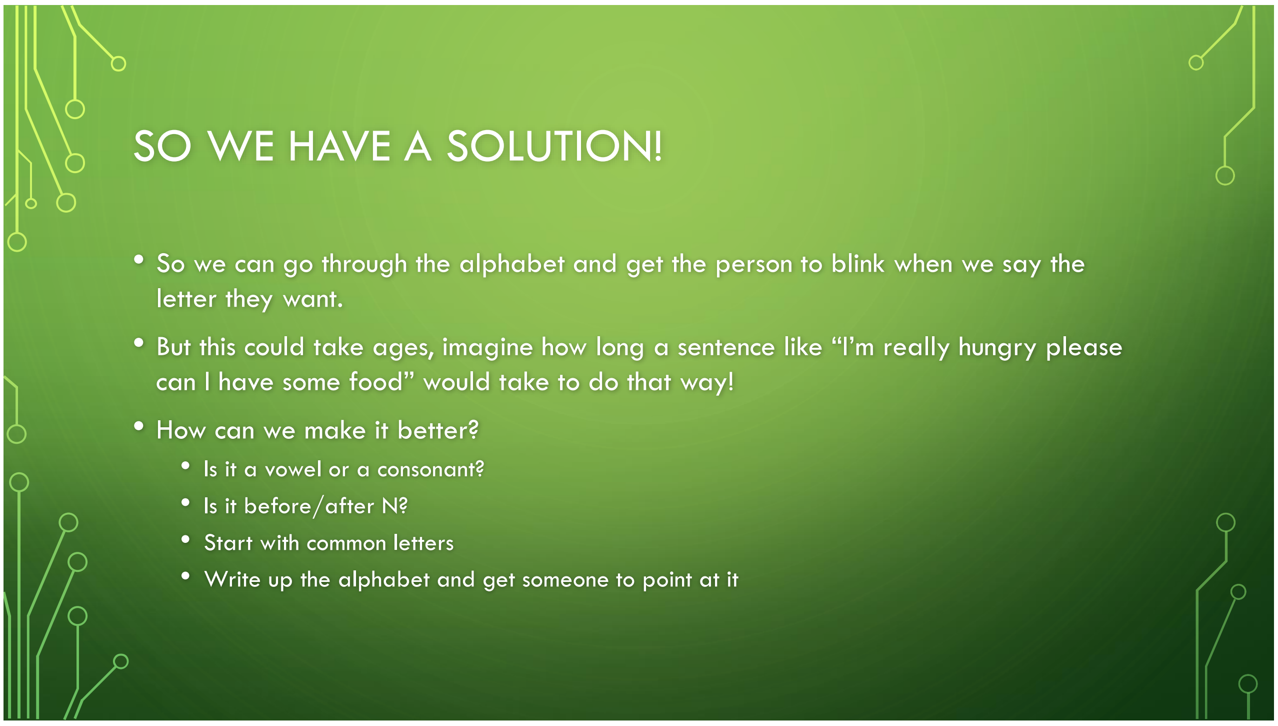
WHAT PROBLEMS COULD A COMPUTER SCIENTIST SOLVE??

- An elderly person finds it hard to physically connect with long-distance relatives.
- An organisation that needs to find the best route to get the most miles out of its airplanes using the least amount of fuel possible.
- Too much information produced by a study for one human to be able to sort through it all in a lifetime.



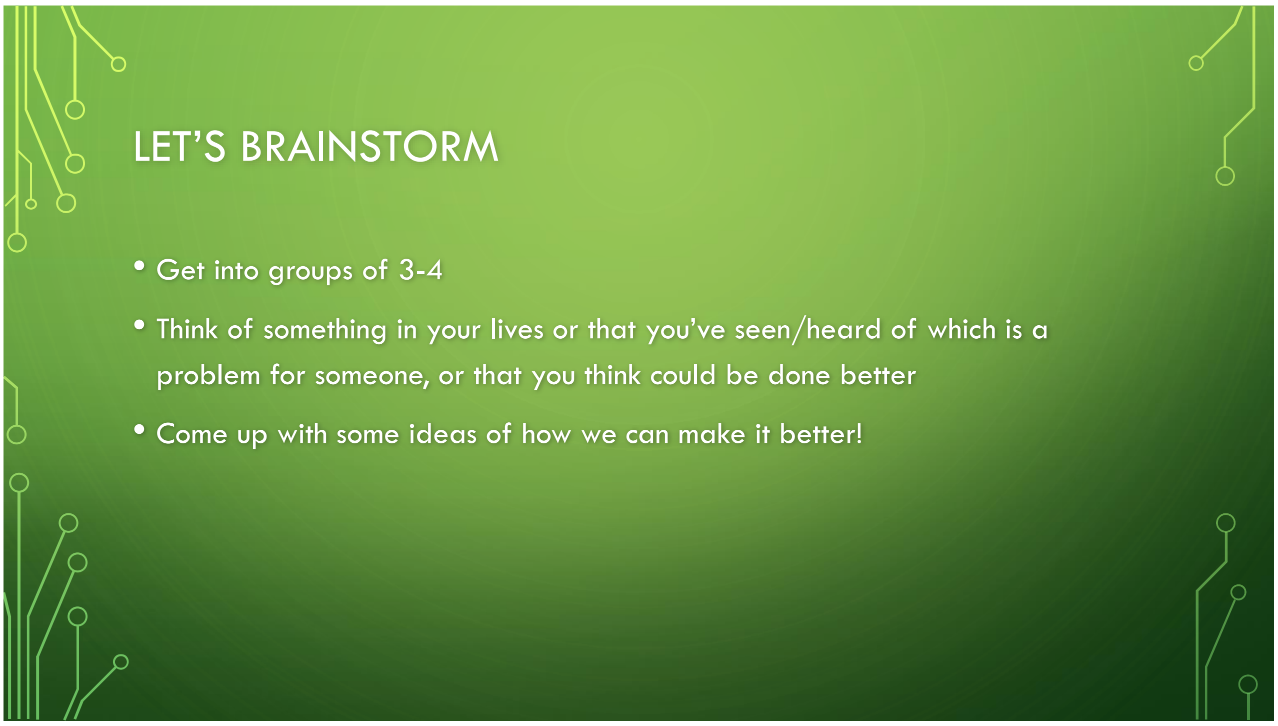
LET'S SOLVE A PROBLEM

- Locked-in Syndrome is a condition that occurs when someone has a stroke and they lose use of most of their body.
- They are fully conscious but just can't move, they're "locked-in" their own body.
- How can we communicate with someone who has Locked-in Syndrome and can only blink?



SO WE HAVE A SOLUTION!

- So we can go through the alphabet and get the person to blink when we say the letter they want.
- But this could take ages, imagine how long a sentence like "I'm really hungry please can I have some food" would take to do that way!
- How can we make it better?
  - Is it a vowel or a consonant?
  - Is it before/after N?
  - Start with common letters
  - Write up the alphabet and get someone to point at it



LET'S BRAINSTORM

- Get into groups of 3-4
- Think of something in your lives or that you've seen/heard of which is a problem for someone, or that you think could be done better
- Come up with some ideas of how we can make it better!

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Introduction to Computational Thinking

Strand: Computer Science Concepts *Duration of Lesson: 40 mins - 1 hour*

Learning Outcomes

Students will be able to define what Computational Thinking is.
Students will be able to describe some of the different elements involved in Computational Thinking.
Students will see that they already use Computational Thinking, and that it can be helpful in lots of areas of school and life.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

Computational Thinking Skills used

Algorithms
Decomposition
Pattern matching
Abstraction

Computer Science concepts/topics involved

Problem solving

Resources / Materials

Pen, paper etc.
Introduction to Computational Thinking Slideshow
Printed Mad Lib sheets

The Class

Activity & Discussion - Introduction – 10-20 minutes

To show the students four of the widely recognised components of Computational Thinking (abstraction, algorithm, decomposition and pattern matching) we're going to use a maths problem (credit: code.org).

Initially ask the students you want them to add the numbers 1-200, and they only have 1 minute to do it! Chances are they'll be shocked/think it's impossible but let them try. Ask them how they got on.

Tell them that it's possible once you take several steps. Firstly, we're going to break the problem up into smaller parts, this is called ***decomposition***.

So how about we start at the two ends, we have 200 + 1, 199 + 2, and so on. Can they see a pattern? What's the last pair we get? Then how many pairs? (this is the ***pattern matching*** part)

Hopefully you end up concluding that you have 100 pairs of 201. So how do we get the answer?

201*100 = 20100, easy!

What about if we wanted to add the numbers from 1-2000? Or 20000? What will stay the same and what will change?

With some guidance, hopefully the students will say that the 201 will change to 2001/20001 respectively. Also, the 100 will change to 1000 and 10000 respectively.

What is the relation of the 201, 2001 and 20001 to our problem? What about the 100, 1000 and 10000?

Hopefully, again with guidance you can show them that if N is the number we're adding up to, then our problem can always be solved using the following formula: (N+1) * (N/2)

This last step uses ***abstraction*** to take out what is specific to each formula and generalises it. Another way of looking at it would be that we look for what's the same in the problem, so that we don't have to do the same work over and over, this gives us our formula. The final formula is also an ***algorithm*** for our solution.

So now we've used Computational Thinking to solve a problem! Go back through the problem, explaining what each step is called and involves.

Discussion – Development - 5-10 minutes

Ask the students if they can think of anywhere in life or in other subjects where they use any of the four parts of Computational Thinking. Either give students some time individually or in pairs to come up with some or open it up to the whole class. The slideshow has examples of some of these. It would be helpful to leave the slide up with the names and definitions of the four parts during this discussion.

Activity – Development – 10-20 minutes

We will have a couple of lessons on Algorithms next and decomposition is used in almost all stages of the course and it's important you continually encourage and remind students to break a problem into smaller parts as they get into programming especially.

To explain the concept of abstraction, use the Mad Lib activities. These are short stories in which the specifics of the story (nouns, verbs, adjectives etc.) have been taken out and generalised. The idea is that, in pairs, students create their own unique story from the same templates. One person in the pair has the story sheet and asks the other person for whatever the story asks (e.g. if it says noun under a blank line then the student gives them a noun and they write it in). At the end, a hilariously different story from the one originally planned should appear! Give both a chance to make the story by swapping after they've finished one.

Feedback – Conclusion – 5 minutes

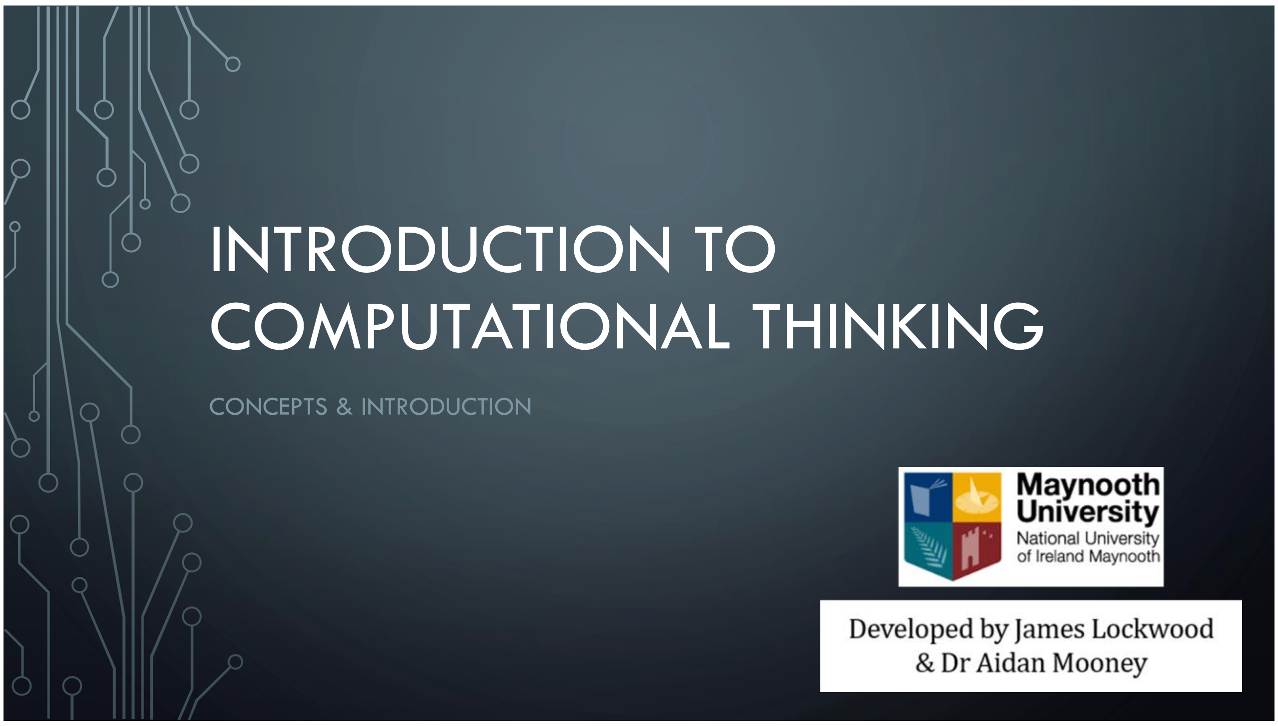
Please get your class to fill out the Computational Thinking feedback form.

Additional Activities

If you want, you could get the students to write their own Mad Lib stories. Get them to write a short story, similar in length to the ones given, and then remove the specifics. This could be especially good to do if you can get an English teacher on board to do it during their class!

Mad Libs have also been done as one-scene plays by US late-night talk show host Jimmy Fallon. Below is a link to one of them. Again, this is something you could get students to do especially if they have a drama class.

https://www.youtube.com/watch?v=pbCXQKRvr8c&t=216s

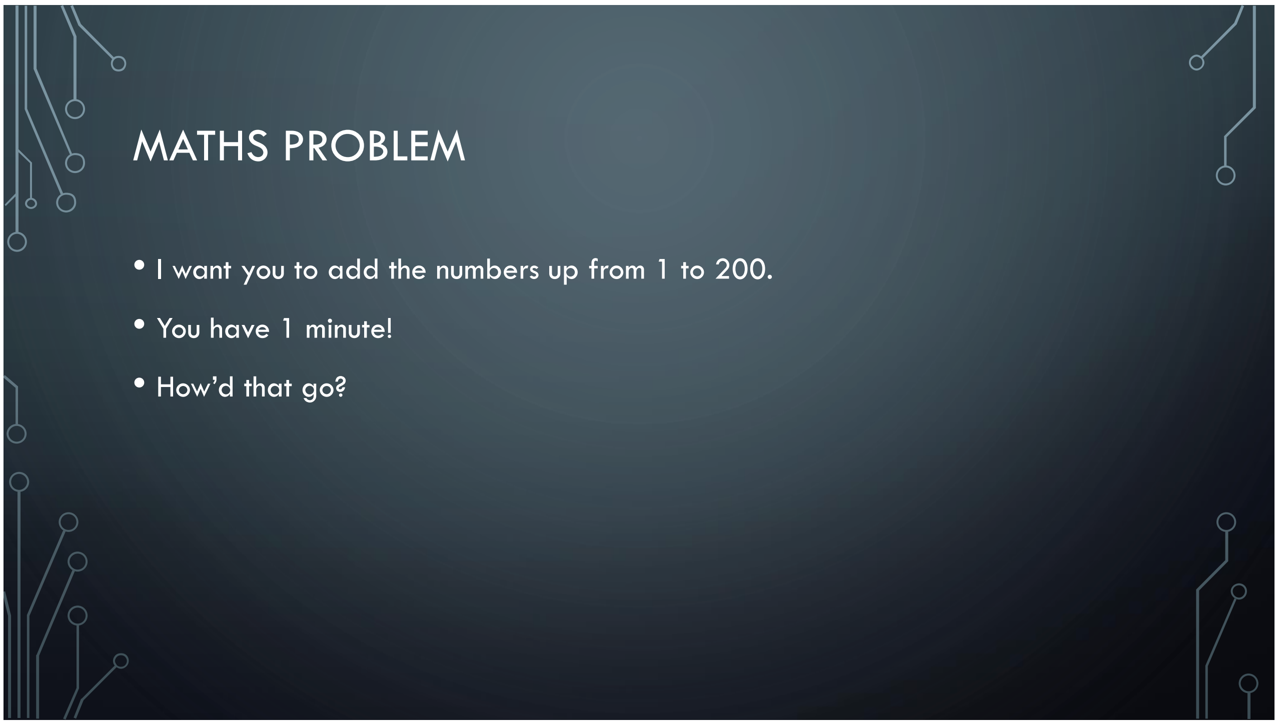


INTRODUCTION TO COMPUTATIONAL THINKING

CONCEPTS & INTRODUCTION

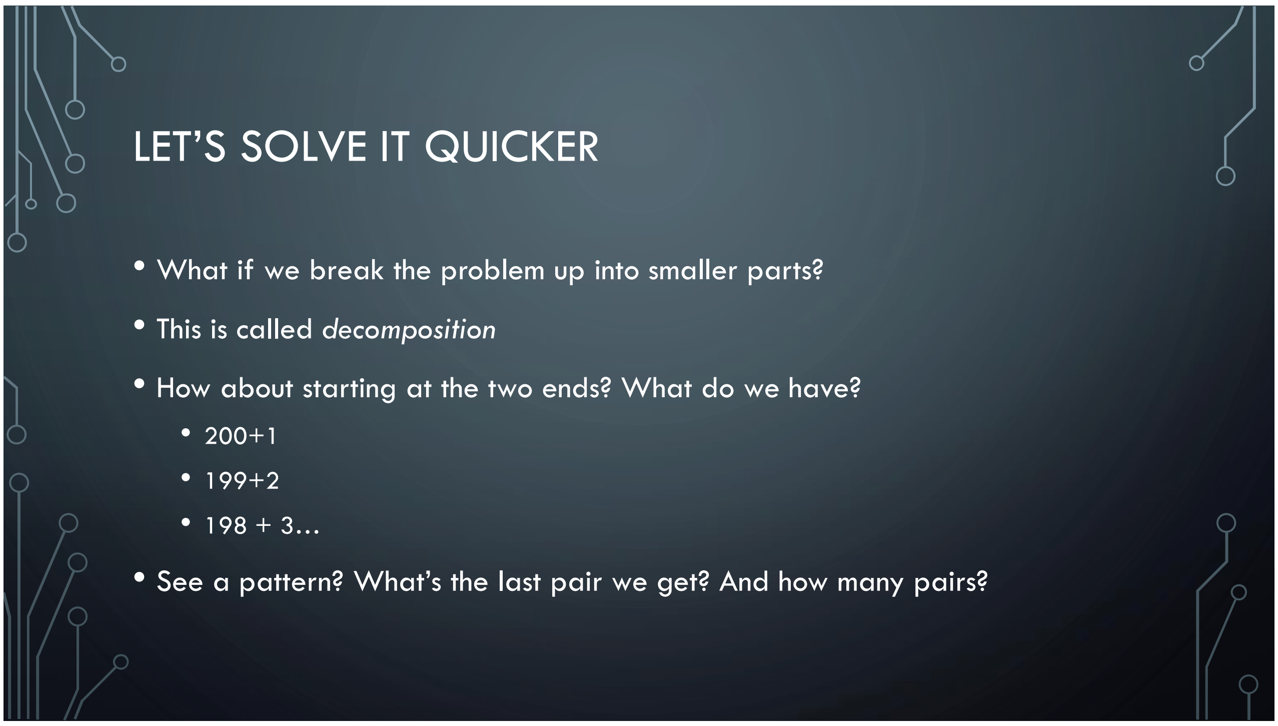
Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



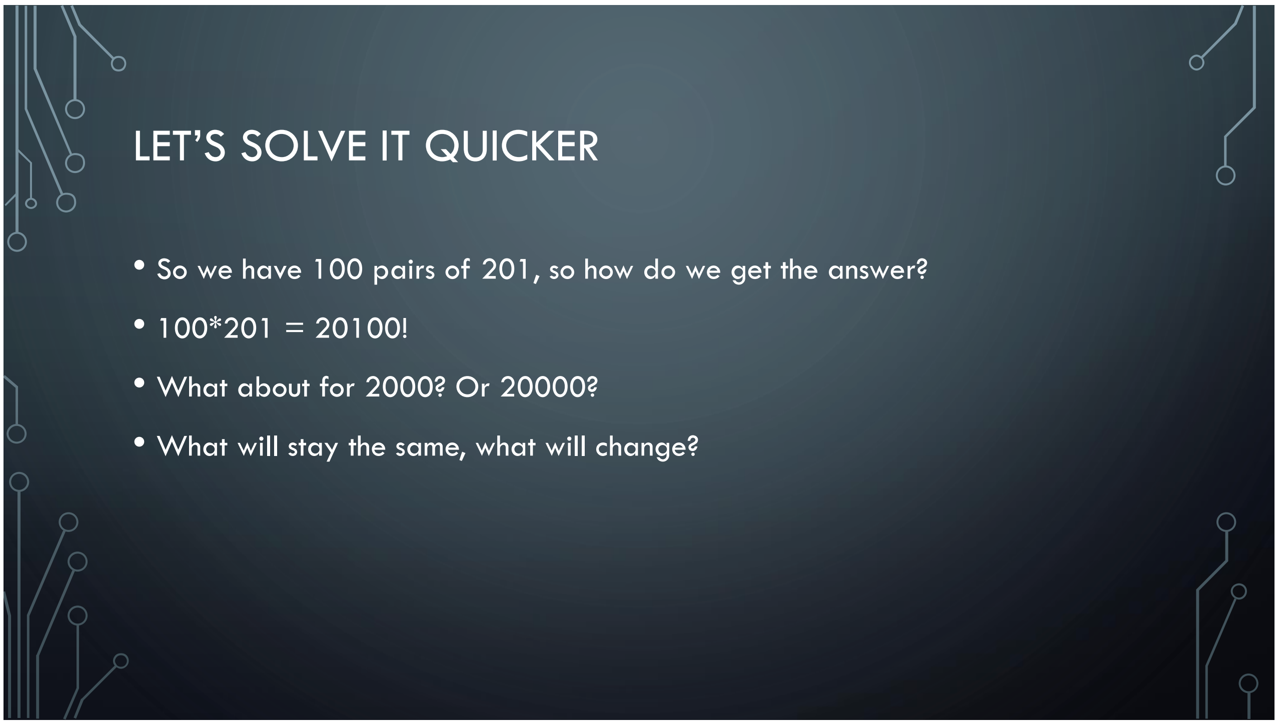
MATHS PROBLEM

- I want you to add the numbers up from 1 to 200.
- You have 1 minute!
- How'd that go?



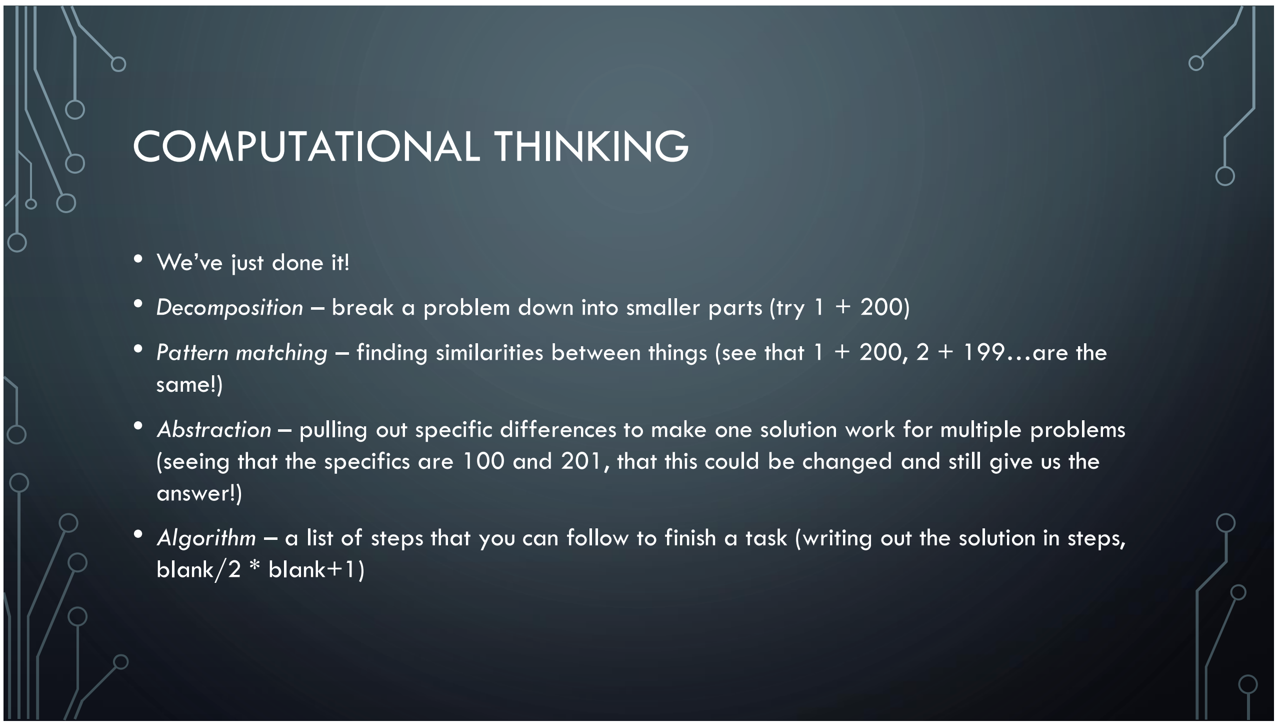
LET'S SOLVE IT QUICKER

- What if we break the problem up into smaller parts?
- This is called *decomposition*
- How about starting at the two ends? What do we have?
  - 200+1
  - 199+2
  - 198 + 3…
- See a pattern? What's the last pair we get? And how many pairs?



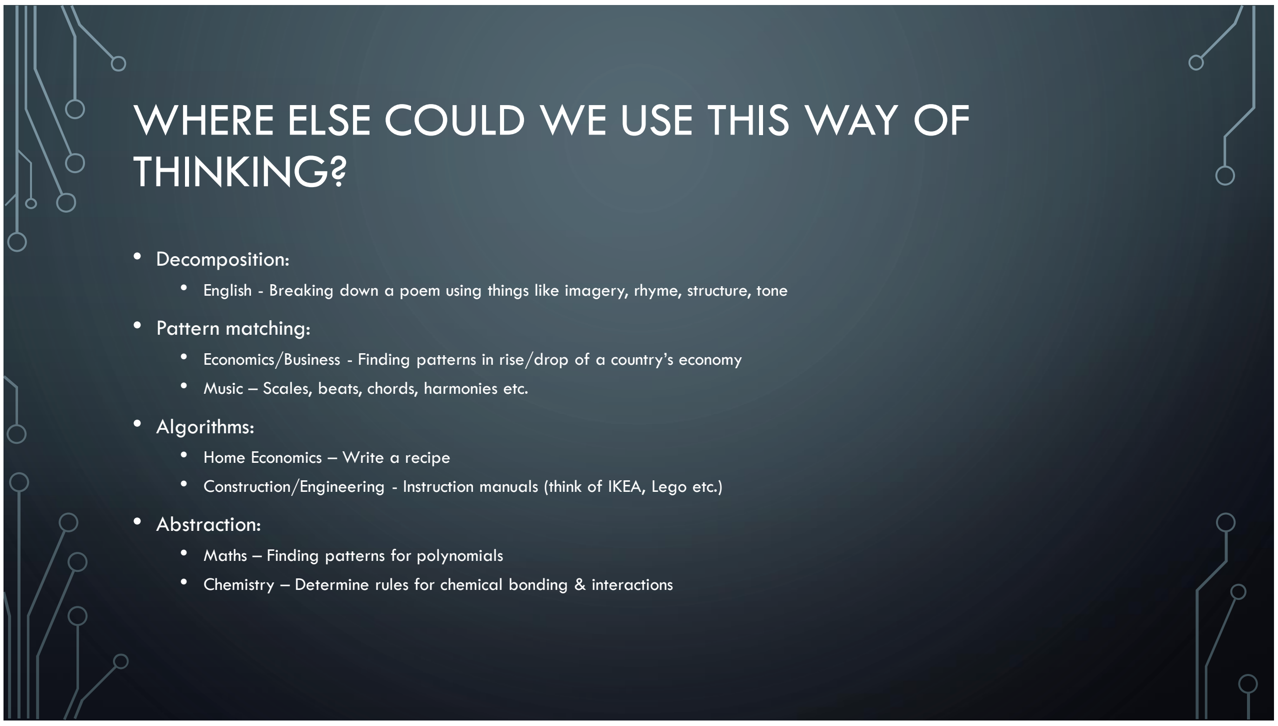
LET'S SOLVE IT QUICKER

- So we have 100 pairs of 201, so how do we get the answer?
- 100*201 = 20100!
- What about for 2000? Or 20000?
- What will stay the same, what will change?



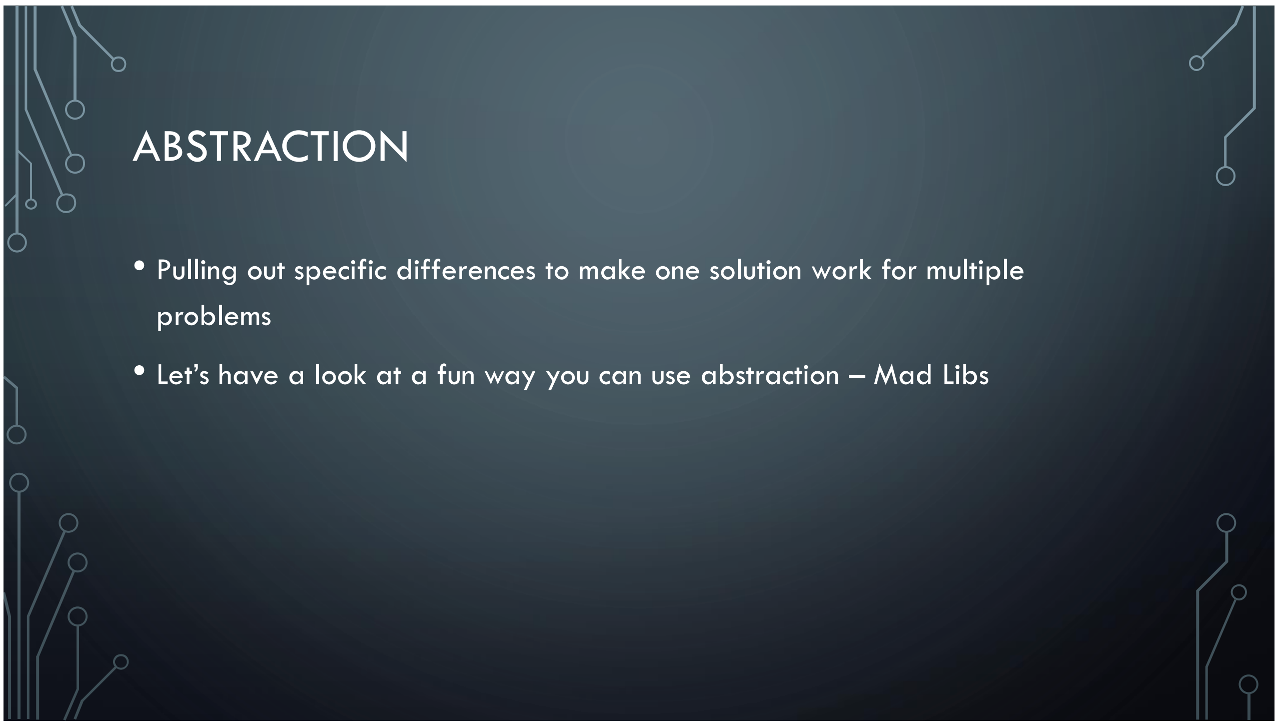
COMPUTATIONAL THINKING

- We've just done it!
- *Decomposition* – break a problem down into smaller parts (try 1 + 200)
- *Pattern matching* – finding similarities between things (see that 1 + 200, 2 + 199…are the same!)
- *Abstraction* – pulling out specific differences to make one solution work for multiple problems (seeing that the specifics are 100 and 201, that this could be changed and still give us the answer!)
- *Algorithm* – a list of steps that you can follow to finish a task (writing out the solution in steps, blank/2 * blank+1)



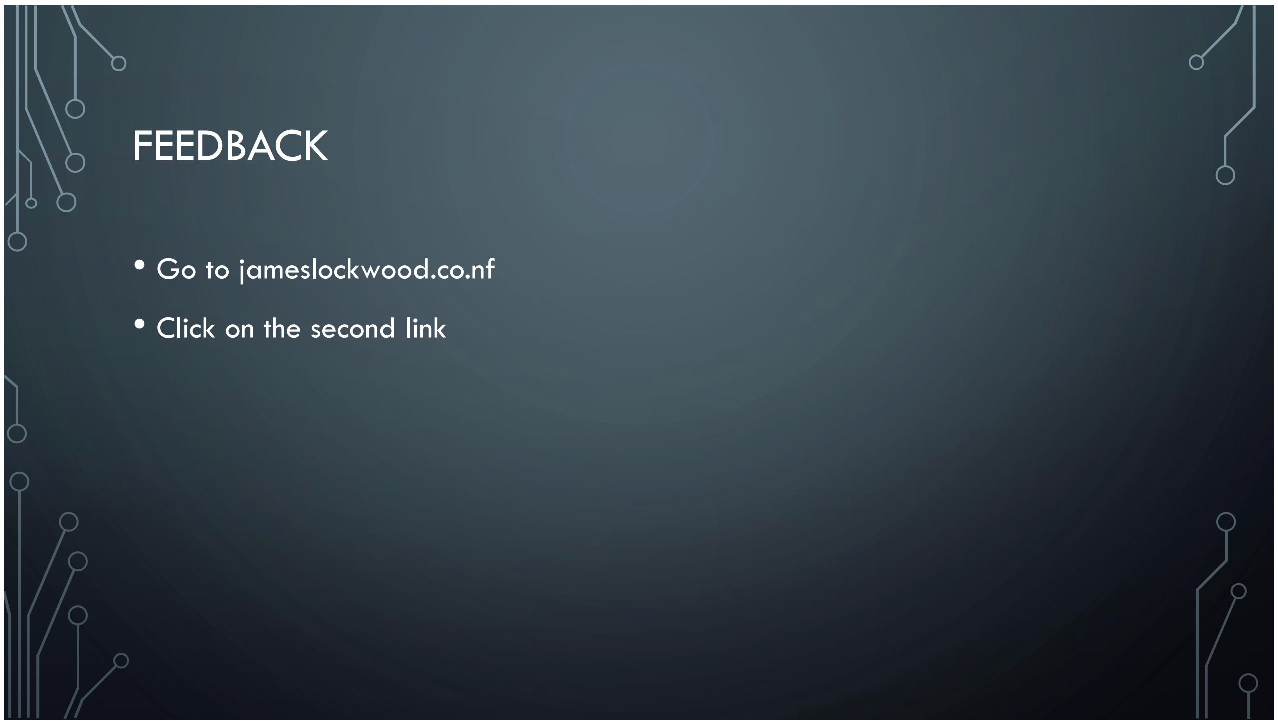
WHERE ELSE COULD WE USE THIS WAY OF THINKING?

- Decomposition:
  - English - Breaking down a poem using things like imagery, rhyme, structure, tone
- Pattern matching:
  - Economics/Business - Finding patterns in rise/drop of a country's economy
  - Music – Scales, beats, chords, harmonies etc.
- Algorithms:
  - Home Economics – Write a recipe
  - Construction/Engineering - Instruction manuals (think of IKEA, Lego etc.)
- Abstraction:
  - Maths – Finding patterns for polynomials
  - Chemistry – Determine rules for chemical bonding & interactions



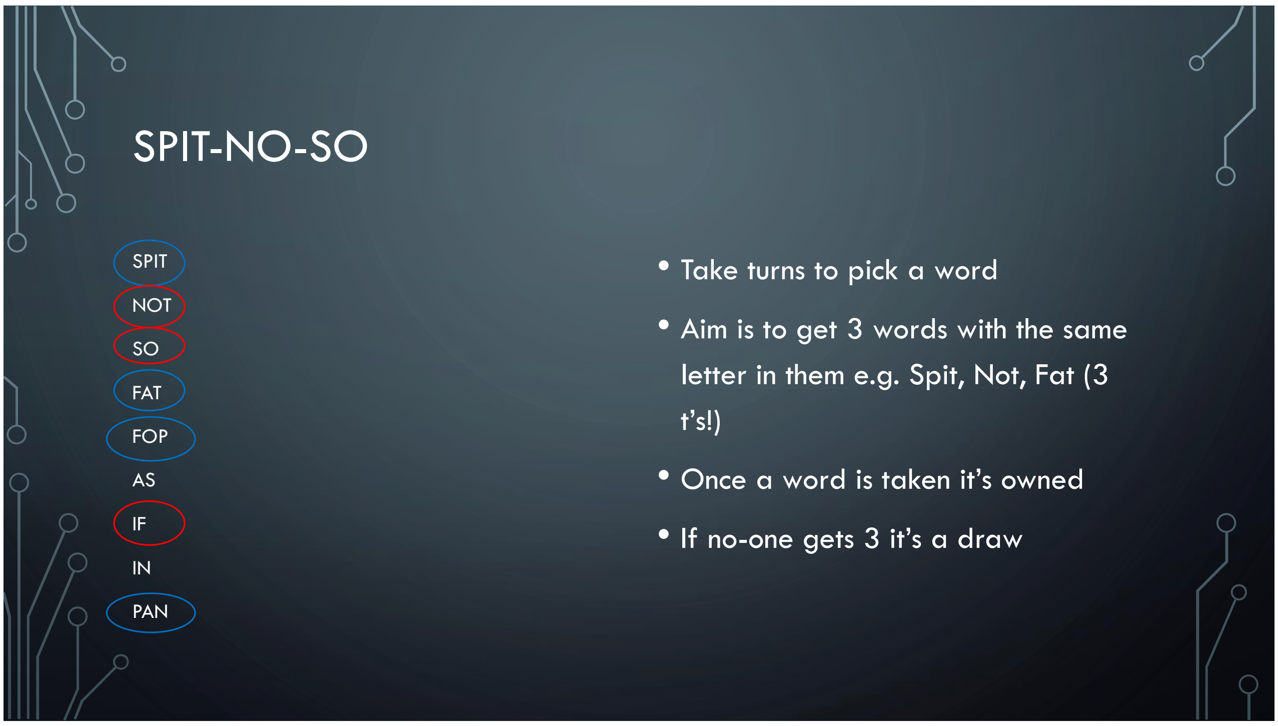
ABSTRACTION

- Pulling out specific differences to make one solution work for multiple problems
- Let’s have a look at a fun way you can use abstraction – Mad Libs



FEEDBACK

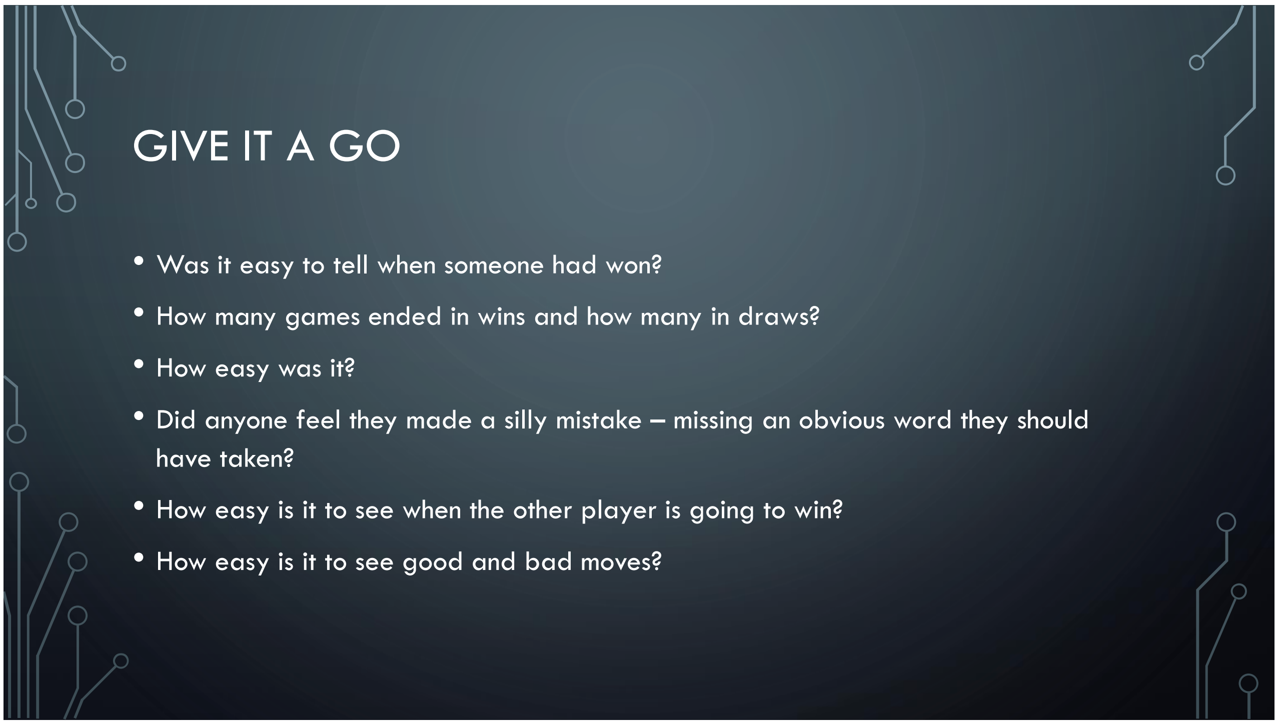
- Go to jameslockwood.co.nf
- Click on the second link



SPIT-NO-SO

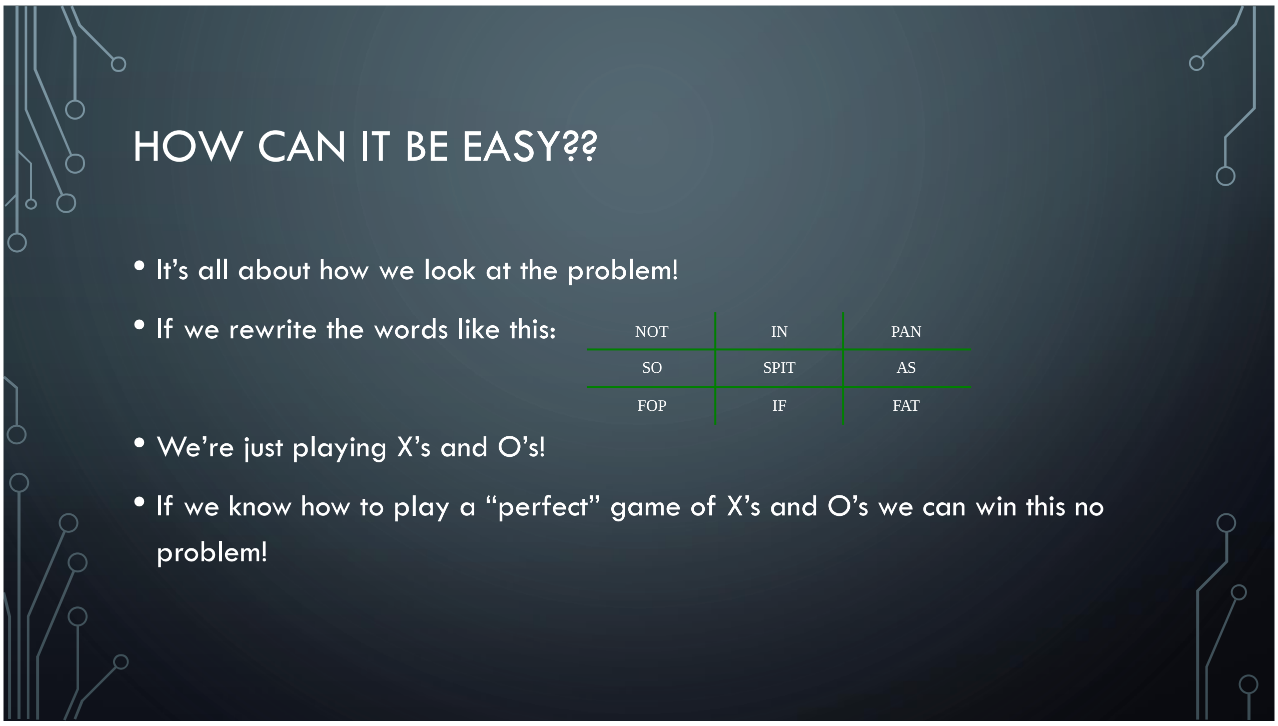
SPIT
NOT
SO
FAT
FOP
AS
IF
IN
PAN

- Take turns to pick a word
- Aim is to get 3 words with the same letter in them e.g. Spit, Not, Fat (3 t's!)
- Once a word is taken it's owned
- If no-one gets 3 it's a draw



GIVE IT A GO

- Was it easy to tell when someone had won?
- How many games ended in wins and how many in draws?
- How easy was it?
- Did anyone feel they made a silly mistake – missing an obvious word they should have taken?
- How easy is it to see when the other player is going to win?
- How easy is it to see good and bad moves?



HOW CAN IT BE EASY??

- It's all about how we look at the problem!
- If we rewrite the words like this:

NOT	IN	PAN
SO	SPIT	AS
FOP	IF	FAT

- We're just playing X's and O's!
- If we know how to play a "perfect" game of X's and O's we can win this no problem!

Developed by James Lockwood
& Dr Aidan Mooney

Algorithms 1

Strand: Computer Science Concepts *Duration of Lesson: 40 mins - 1 hour*

Learning Outcomes

Students will understand what an algorithm is.
Given a problem/situation they can develop an algorithm.
Should understand and can use stepwise refinement and decomposition.
Will be introduced to debugging but this won't explain this yet.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.7 develop algorithms to implement chosen solutions

1.20 use modelling and simulation in relevant situations

1.21 collaborate and assign roles and responsibilities within a team to tackle a computing task

2.2 use a range of methods for identifying patterns and abstract common features

2.5 use pseudo code to outline the functionality of an algorithm

2.6 construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement

2.19 test solutions and decisions to determine their short-term and long-term outcomes

3.1 understand and list user needs/requirements before defining a solution

Junior Certificate Coding Short Course

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

Computational Thinking Skills used

Algorithm
Abstraction
Decomposition

Computer Science concepts/topics involved

Algorithms
Stepwise Refinement
Debugging

Resources / Materials

Paper, pens etc.
Algorithms 1 Slideshow

The Class

Activity – Introduction – 10 minutes

Working in pairs: Imagine an alien has come to earth, he had heard about tea/cakes/holidays etc. Explain how to do one of the below things in steps:

- Make a cup of tea
- Bake a cake
- Order a pizza
- Pack a bag
- Make up their own

As students are doing this, make sure to go around and test their algorithms out. If some finish early, get them to expand on points like the one described below. Alternatively get them to write up the steps for another activity.

Example: Say the student has a step like this: "Boil the kettle", ask them how they do that, for example: "Fill the kettle with water, plug it in, flick the switch, wait for it to boil". This leads to the next stage of algorithm creation which we'll discuss below.

Also to note is that "wait for it to boil" could be broken down still further, how do we know when it's boiled for example? You can go on forever so don't push them too far as it can get frustrating, but it's important to note that computers will do exactly what you tell them to so instructions have to be precise!

Discussion – Introduction – 5-10 minutes

Show them one made earlier (you can use the one given in the slideshow), make one with two/three steps and another with 10+ steps. How do we know how many is enough? We don't really know!

Explain that this step-by-step process for solving problems is called an **algorithm.**

Show that the 2/3 step example and the longer one are the same, that they even contain the same lines! Say that when designing an algorithm, it's helpful to start with a few very broad and "big" instructions.

Then to refine the algorithm break these bigger instructions into smaller steps. This is called **stepwise refinement** and the process of breaking a big problem into smaller pieces is called **decomposition.**

Activity – Development – 5-10 minutes (this can be cut if time is an issue)

Have another go, taking a different example and starting with two/three steps and then breaking each of those up.

Activity – Development - 10 minutes

Again, in pairs, perhaps different ones, take the game rock, paper, scissors and develop an algorithm for how to play it. Use stepwise refinement.

When pairs have finished, swap the algorithms with another pair and get them to "act out" the instructions given to them.

Pairs should note errors and problems that have been identified.

Swap back the algorithms and fix the problems. (this is the **debugging** part)

Discussion – Development – 10 minutes

Explain that computers are very hard to please, they need to have instructions given in a very specific way and format which is what computer programming is all about. If you tell the computer to do something, like if someone tells you to move your hand, it could do anything, just like you could move it up/left/down and any sort of distance unless you're very specific. This is called **ambiguity.**

Have a discussion about other parts of life where algorithms and ambiguity could occur, *what other subjects involve algorithms*? There are examples in the slideshow. Other questions to talk about could be *why are algorithms useful* and *how do we know if we have a good algorithm*?

Discussion – Conclusion – 5 minutes

Tell them how algorithms are a vital part of computer science, and life in general! If people didn't come up with a way to do things then we wouldn't have recipe books, IKEA furniture, Lego or anything else! Soon we'll look at some specific examples of algorithms but next lesson we'll have a look at some more practical examples of algorithms. *Can what we've learned be applied to any problem?*

Keep in mind that when you start programming it's a great opportunity to revise algorithms if you get them to write one for each program, it's also good practice.

Feedback – Conclusion – 5 minutes

Please get your students to fill out the feedback form.

Additional activities

The fox, the chicken and the corn – 10 minutes

- A man carrying a fox, a chicken and a sack of corn wishes to cross a river
- His canoe can only hold him and one other thing
    - If the fox and chicken are left together the fox will eat the chicken
    - If the chicken and corn are left together the chicken will eat the corn
- Figure out an algorithm to get everything across the river
- The commands are
    - Put x in the canoe
    - Paddle across the river
    - Take x out of the canoe (this is abstraction – is there a way we can get the students to do this?)

LightBot – up to an hour

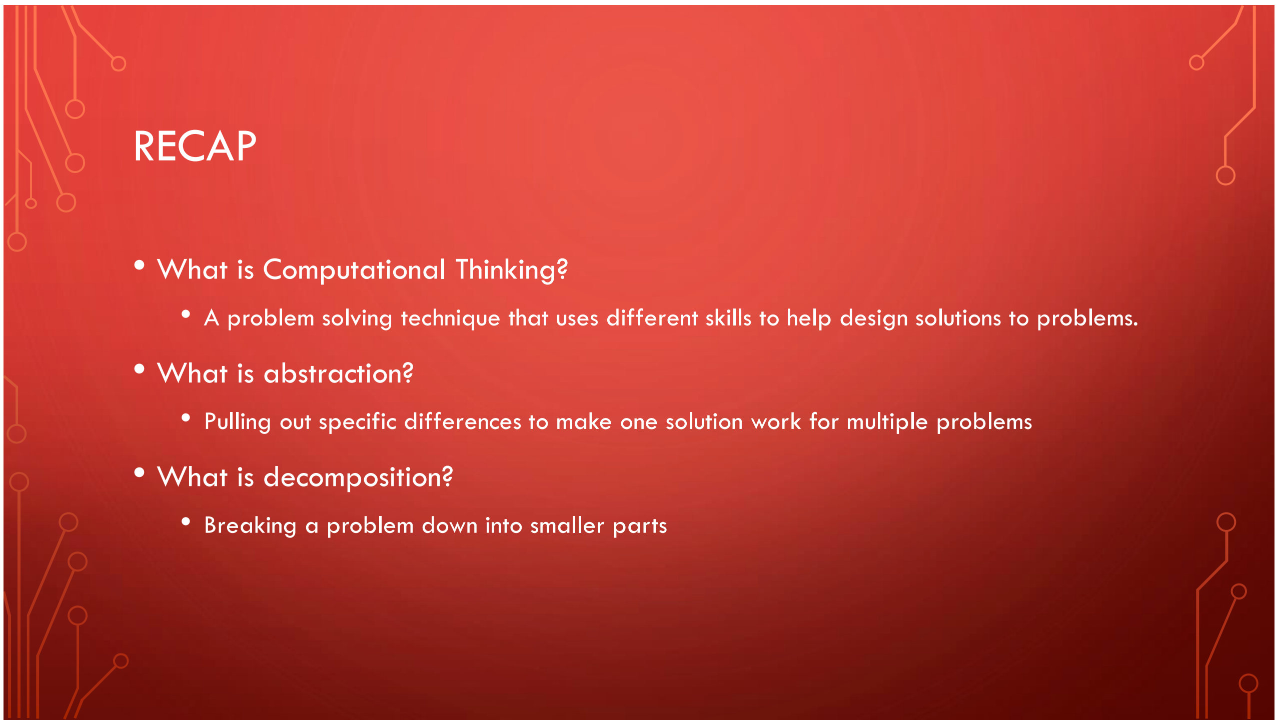
You can direct them to LightBot (https://lightbot.com/flash.html) if you want them to have a go at algorithm design in a very different format. Sadly, LightBot now costs but there is an Hour of Code Activity which is free.

ALGORITHMS 1

CONCEPTS & INTRODUCTION

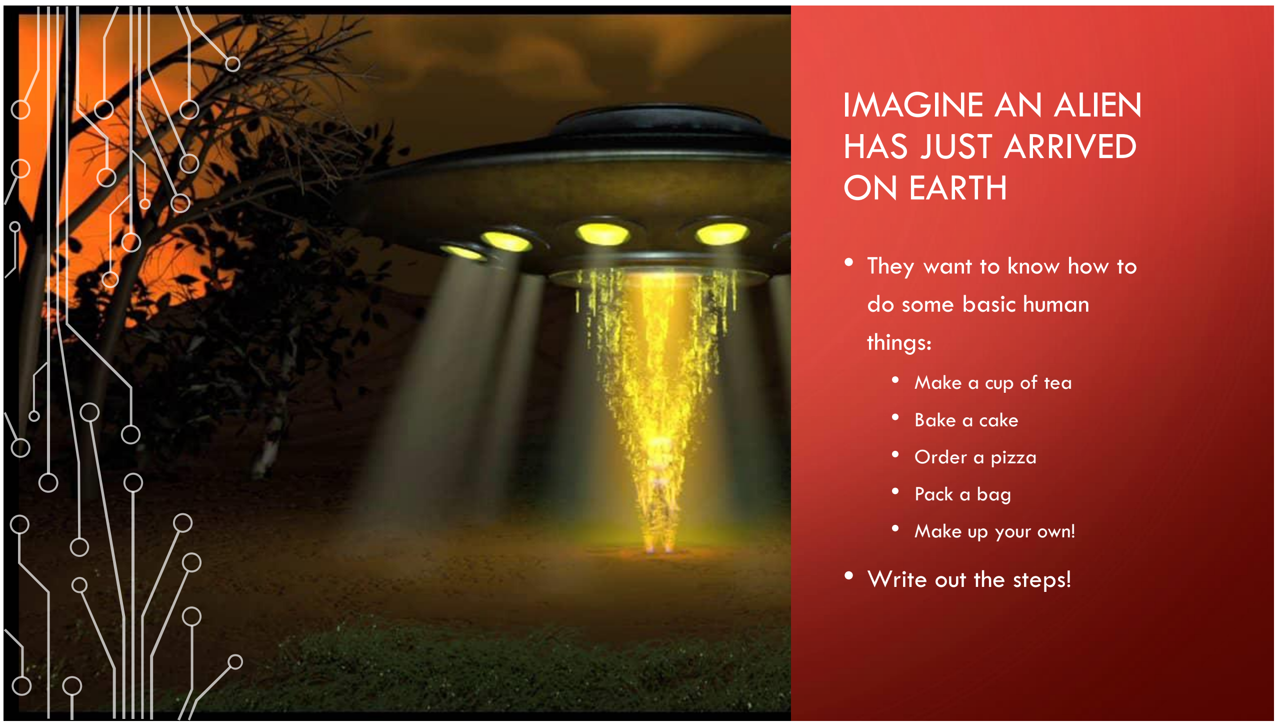
Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



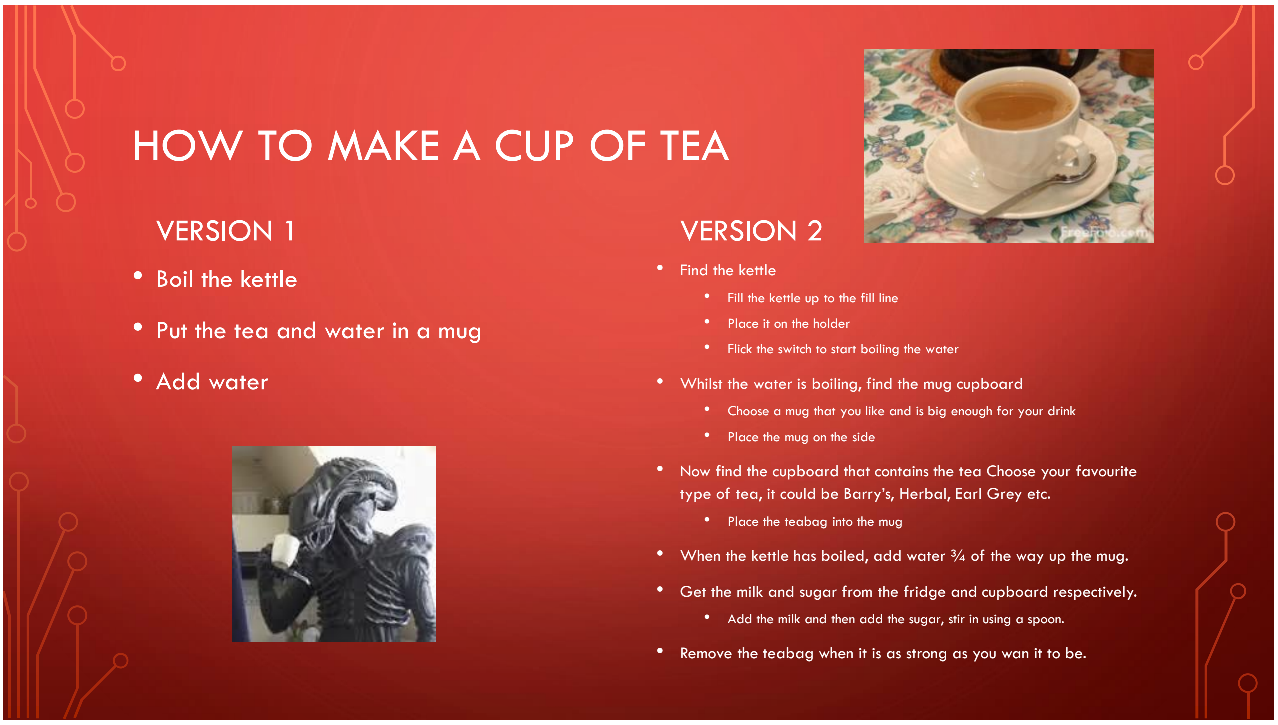
RECAP

- What is Computational Thinking?
  - A problem solving technique that uses different skills to help design solutions to problems.
- What is abstraction?
  - Pulling out specific differences to make one solution work for multiple problems
- What is decomposition?
  - Breaking a problem down into smaller parts



IMAGINE AN ALIEN HAS JUST ARRIVED ON EARTH

- They want to know how to do some basic human things:
  - Make a cup of tea
  - Bake a cake
  - Order a pizza
  - Pack a bag
  - Make up your own!
- Write out the steps!



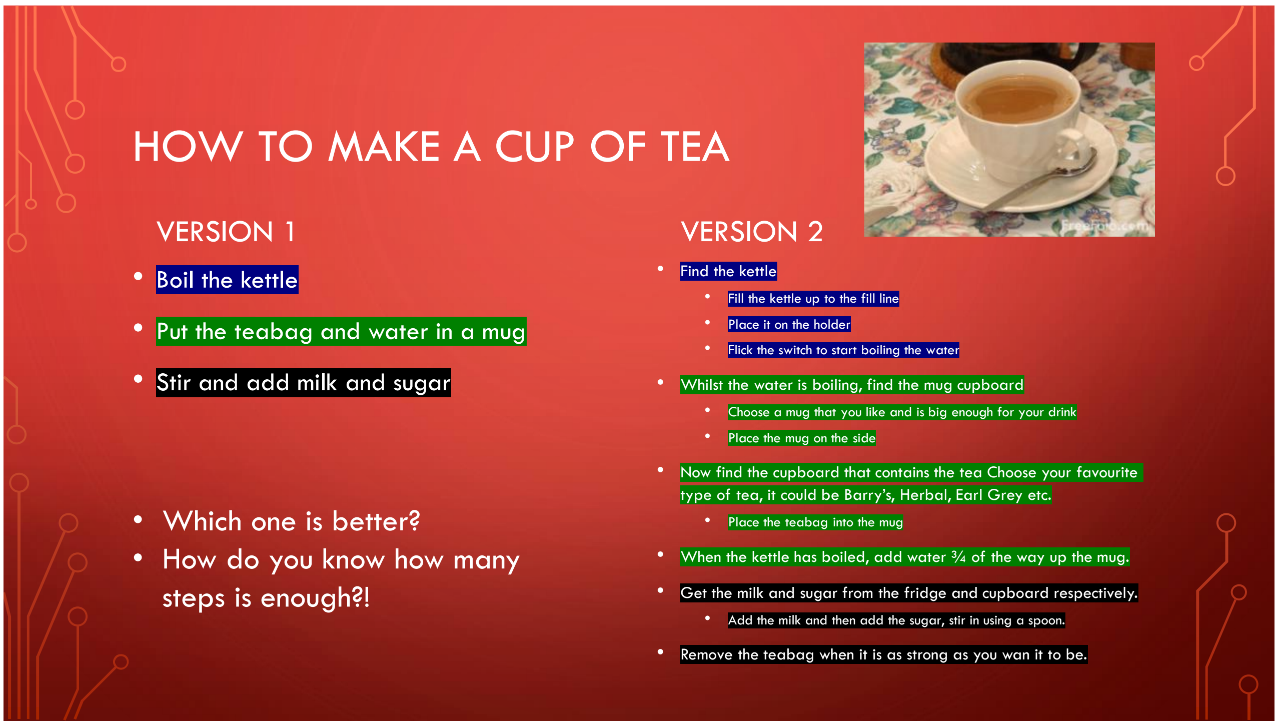
HOW TO MAKE A CUP OF TEA

VERSION 1

- Boil the kettle
- Put the tea and water in a mug
- Add water

VERSION 2

- Find the kettle
  - Fill the kettle up to the fill line
  - Place it on the holder
  - Flick the switch to start boiling the water
- Whilst the water is boiling, find the mug cupboard
  - Choose a mug that you like and is big enough for your drink
  - Place the mug on the side
- Now find the cupboard that contains the tea Choose your favourite type of tea, it could be Barry's, Herbal, Earl Grey etc.
  - Place the teabag into the mug
- When the kettle has boiled, add water ¾ of the way up the mug.
- Get the milk and sugar from the fridge and cupboard respectively.
  - Add the milk and then add the sugar, stir in using a spoon.
- Remove the teabag when it is as strong as you wan it to be.



HOW TO MAKE A CUP OF TEA

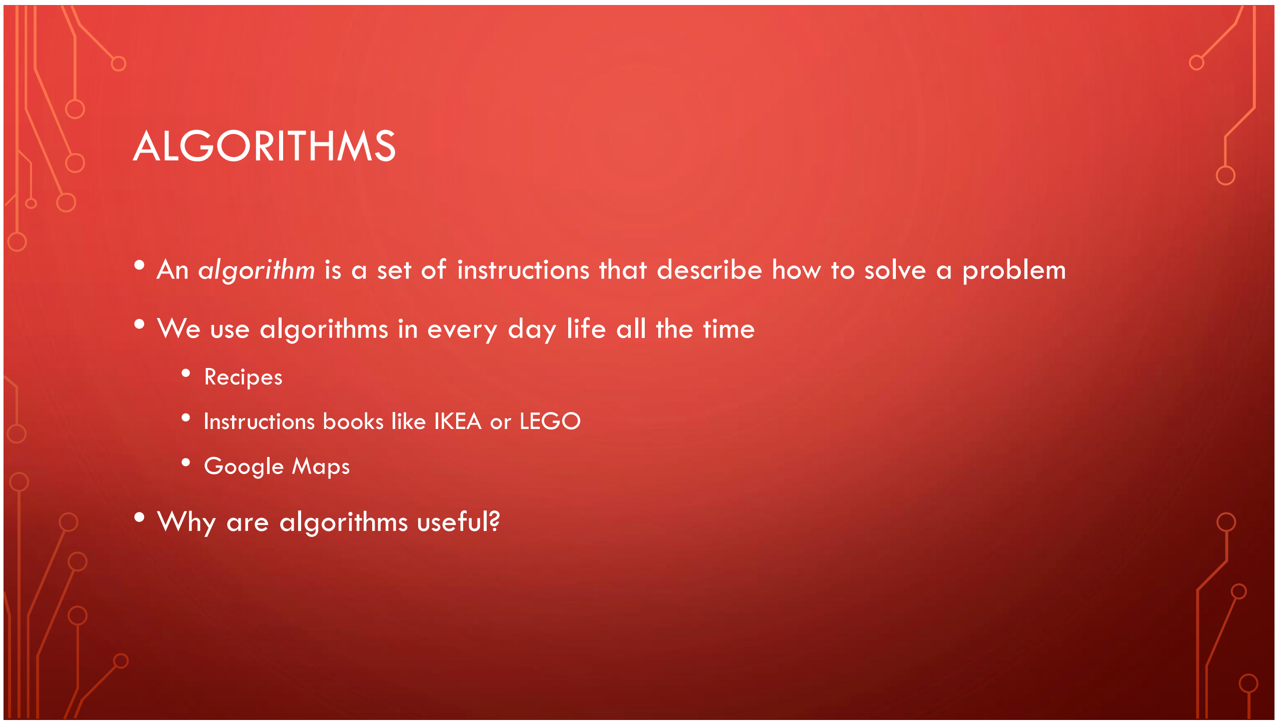
VERSION 1

- Boil the kettle
- Put the teabag and water in a mug
- Stir and add milk and sugar

- Which one is better?
- How do you know how many steps is enough?!

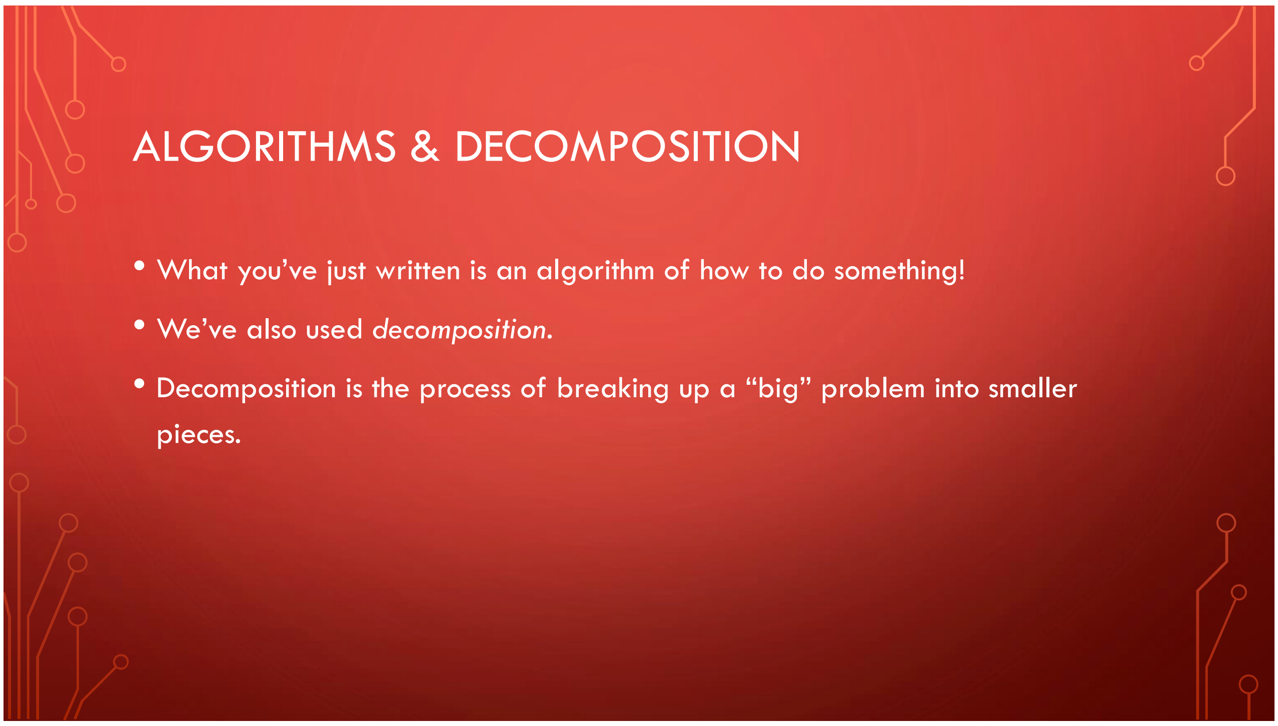
VERSION 2

- Find the kettle
  - Fill the kettle up to the fill line
  - Place it on the holder
  - Flick the switch to start boiling the water
- Whilst the water is boiling, find the mug cupboard
  - Choose a mug that you like and is big enough for your drink
  - Place the mug on the side
- Now find the cupboard that contains the tea Choose your favourite type of tea, it could be Barry's, Herbal, Earl Grey etc.
  - Place the teabag into the mug
- When the kettle has boiled, add water ¾ of the way up the mug.
- Get the milk and sugar from the fridge and cupboard respectively.
  - Add the milk and then add the sugar, stir in using a spoon.
- Remove the teabag when it is as strong as you wan it to be.



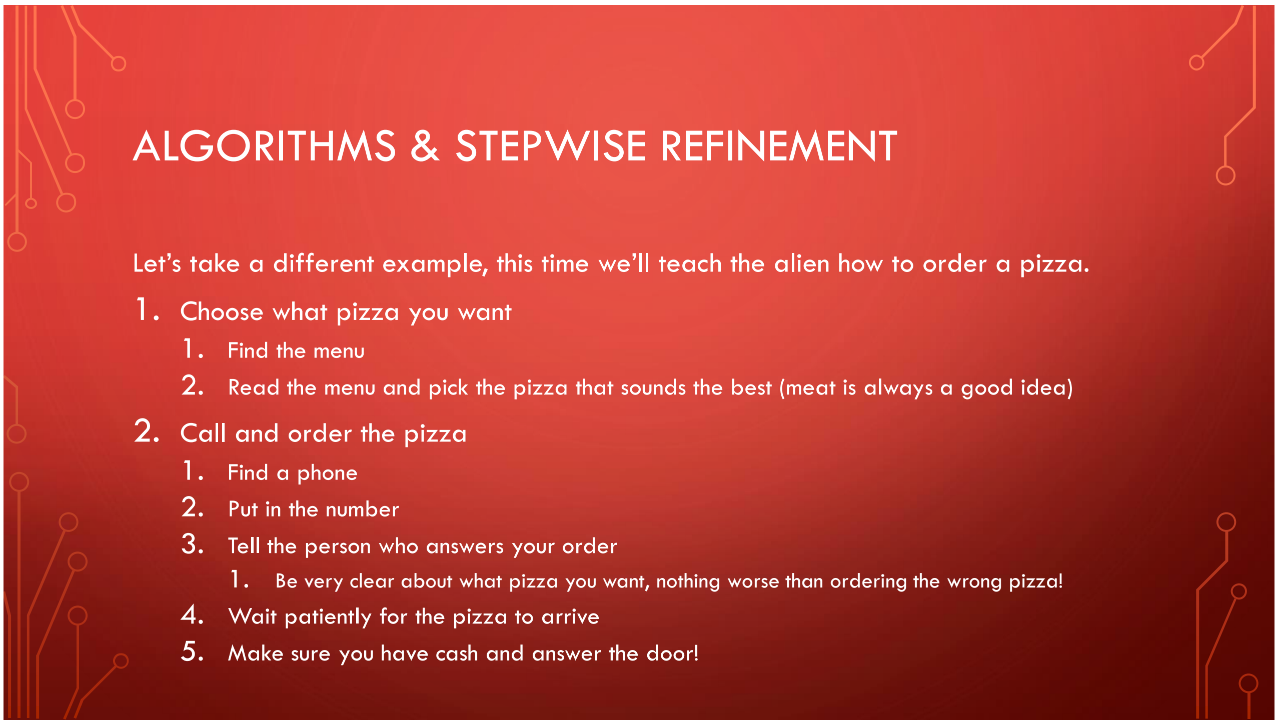
ALGORITHMS

- An *algorithm* is a set of instructions that describe how to solve a problem
- We use algorithms in every day life all the time
  - Recipes
  - Instructions books like IKEA or LEGO
  - Google Maps
- Why are algorithms useful?



ALGORITHMS & DECOMPOSITION

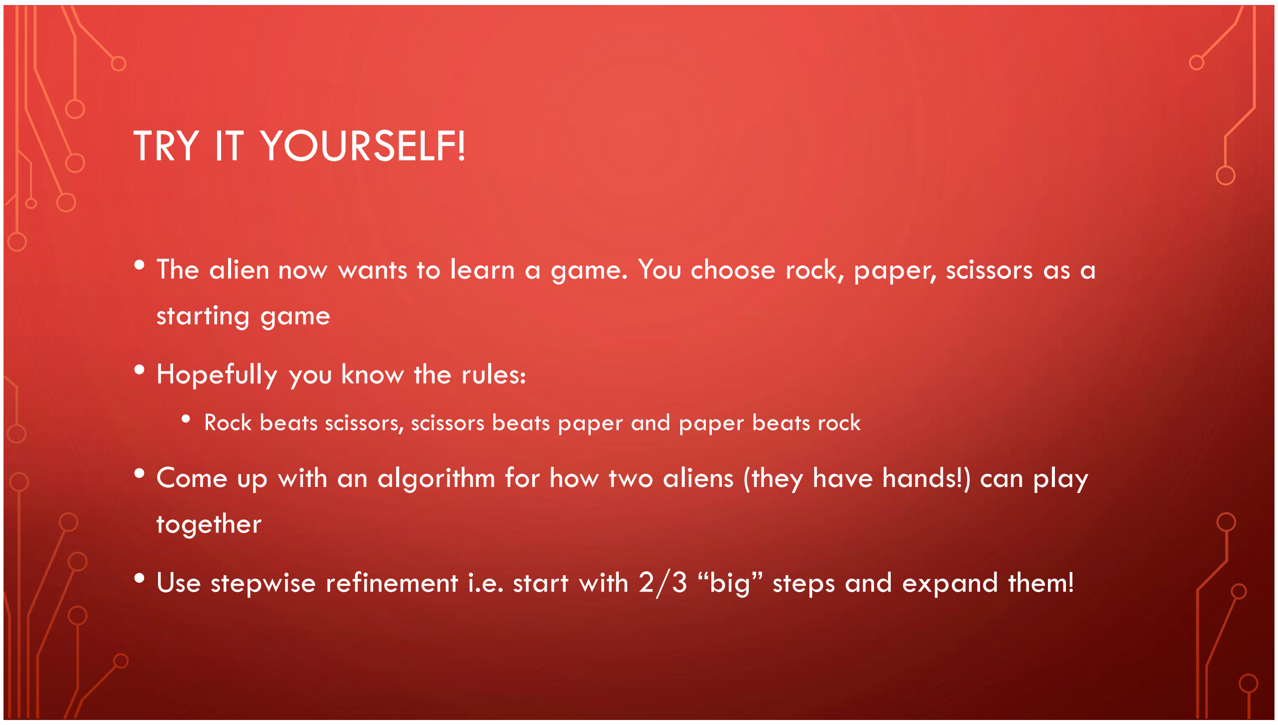
- What you've just written is an algorithm of how to do something!
- We've also used *decomposition.*
- Decomposition is the process of breaking up a "big" problem into smaller pieces.



ALGORITHMS & STEPWISE REFINEMENT

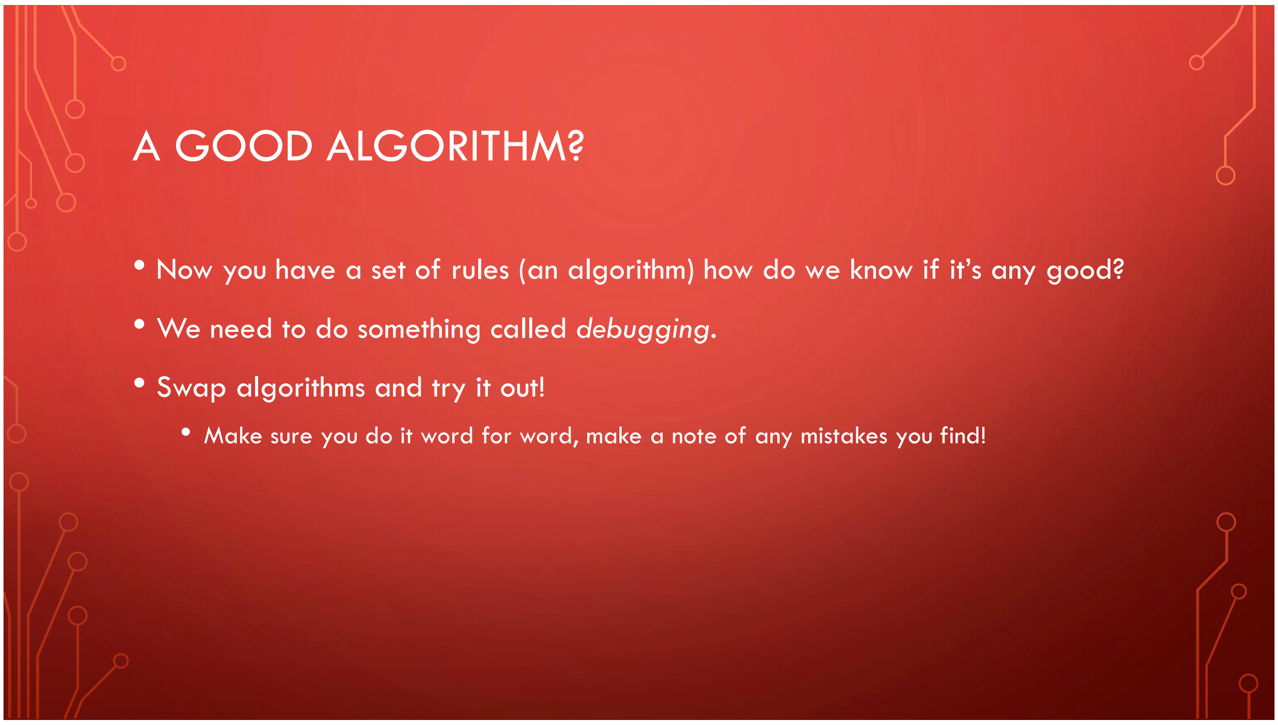
Let's take a different example, this time we'll teach the alien how to order a pizza.

1. Choose what pizza you want
    1. Find the menu
    2. Read the menu and pick the pizza that sounds the best (meat is always a good idea)
2. Call and order the pizza
    1. Find a phone
    2. Put in the number
    3. Tell the person who answers your order
        1. Be very clear about what pizza you want, nothing worse than ordering the wrong pizza!
    4. Wait patiently for the pizza to arrive
    5. Make sure you have cash and answer the door!



TRY IT YOURSELF!

- The alien now wants to learn a game. You choose rock, paper, scissors as a starting game
- Hopefully you know the rules:
    - Rock beats scissors, scissors beats paper and paper beats rock
- Come up with an algorithm for how two aliens (they have hands!) can play together
- Use stepwise refinement i.e. start with 2/3 “big” steps and expand them!



A GOOD ALGORITHM?

- Now you have a set of rules (an algorithm) how do we know if it’s any good?
- We need to do something called *debugging*.
- Swap algorithms and try it out!
    - Make sure you do it word for word, make a note of any mistakes you find!

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Algorithms 2

Strand: Computer Science Concepts *Duration of Lesson: 40 mins - 1 hour*

Learning Outcomes

Students will understand ambiguity, that is, that one thing can have multiple meanings.
Students will gain confidence in designing and implementing algorithms.
Students will learn about testing and debugging.
There's an option for students to learn about encoding.
There's an option for students to learn about unit testing.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.7 develop algorithms to implement chosen solutions

1.20 use modelling and simulation in relevant situations

1.21 collaborate and assign roles and responsibilities within a team to tackle a computing task

2.2 use a range of methods for identifying patterns and abstract common features

2.5 use pseudo code to outline the functionality of an algorithm

2.6 construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement

2.19 test solutions and decisions to determine their short-term and long-term outcomes

3.1 understand and list user needs/requirements before defining a solution

Junior Certificate Coding Short Course

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

Computational Thinking Skills used

Algorithms
Decomposition
Pattern matching
Abstraction

Computer Science concepts/topics involved

Debugging
Ambiguity
Encoding
Algorithms
Testing
Unit testing

Resources / Materials

Paper, pens etc.
Lego bricks or similar, enough for all pairs to have 6-10 pieces each.
Algorithms 2 slideshow
Prebuilt obstacle course made of chairs, tables, etc.

The Class

Discussion – Recap – 5 minutes

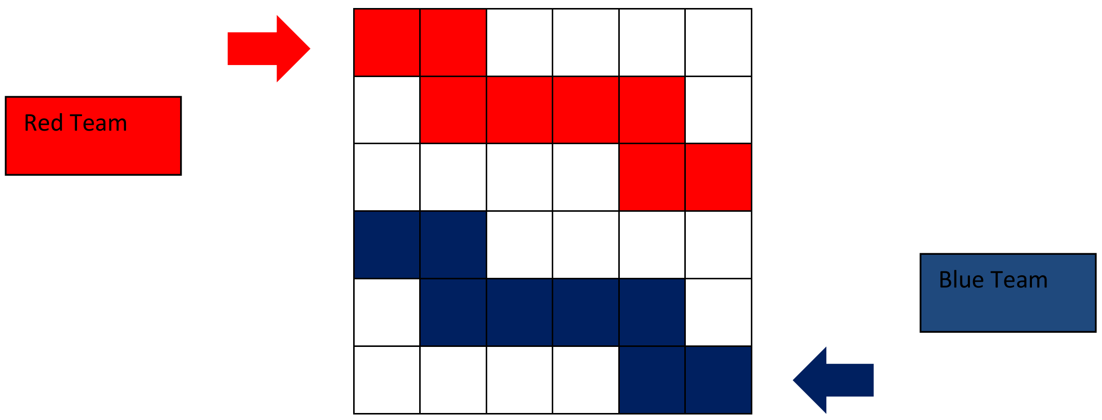
Recap what an algorithm is? What is stepwise refinement?

Activity – Introduction – 10 minutes (can be dropped if time is an issue)

At the end of Algorithms 1 we introduced the idea of **ambiguity**. To expand on this a bit more, we're going to do a task involving Lego bricks. Split the class into pairs and give each group 6-10 Lego bricks each. Give them a mixture of colours and sizes but try and ensure some in each set are the same colour and/or size. They then should construct something using all the bricks and write out an **algorithm** to make it. When they've done this, each pair swaps places with another, leaving the pieces and algorithm (it might be helpful to get groups to take a photo of what they designed so they can "prove" they were right/wrong or in case they forget). Hopefully they can make the correct design but some groups may run into some problems!

Activity – Development – 10-15 minutes

This activity is a fun introduction to the longer maze activity described below. You need a grid-type layout on the floor like the picture below. To do this you could use pieces of paper or masking/duct tape to tape out the design.



The idea is that students are in 2-4 teams, and the idea is they must get from one side (starting for example at the arrow) of the maze to the other. They can do this by moving forward, left, right and back. The catch is that there's only one safe path per team and the rest of the spaces have mines under them. It's a race to see which team can find the safe path first.

Teams take it in turns to have a go, player A for the Red Team continues taking steps until they "die" by stepping off the path. Player A for the Blue team then does the same. Teams should keep track of the safe part as the next time, player B for the Red Team can walk all the way to the last known safe spot.

You will have to come up with a few routes through the minefield for each team. For a best of 3 game I suggest the below setup:

- Game 1: Teams know the start point, paths are mirrored (some students will be smart and figure this out!), keep the paths short enough (see the example above).
- Game 2: Teams know the start point, make paths slightly longer than in Game 1, its often good not to mirror this one.
- Game 3: Teams don't know the start point, keep paths a similar length to Game 2.

If you have enough students that you want four teams, you can either split the group into 4 teams and have them go from one side to the other. Otherwise you can make two separate mazes and have two teams versus each other on each maze.

Activity – Development – 20-30 minutes (can be shortened if time is an issue)

Have a pre-planned obstacle course made up of chairs, tables etc. Make it as hard to easy as you deem fit. If you want you can use the maze from the previous activity. Split the class into groups of 3-4 people, the idea is that one of them will be a "robot" and the other group members give a set of instructions to the robot to get through the maze. Limit the instructions to things such as move forward X steps, move back X steps, step over, pick up etc. It's often good to have them move from the start to some point A where they have to pick up some object, then take the object to the end of the maze.

Give them a short amount of time, 5 minutes to do this. Then get them to trial it. Hopefully it won't work well, the robot must do **EXACTLY** what the instructions say, you should read them out. Tell them that mistakes happen, and this kind of **testing** is often called **debugging**.

If you want to, you could briefly explain **unit testing**. This is when individual sections of a program are tested. In this case, it would be testing the algorithm up to a certain point (the first chair for example) and then confirming that that **unit** is correct.

One alternative for this would be to give them a set of symbols for all the available moves the robot can make, for example ➡ could be for move right and ⬆ could be for moving forwards. This would then also teach them **encoding**, where you take a set of instructions in one form (English) and translate them to another (symbols).

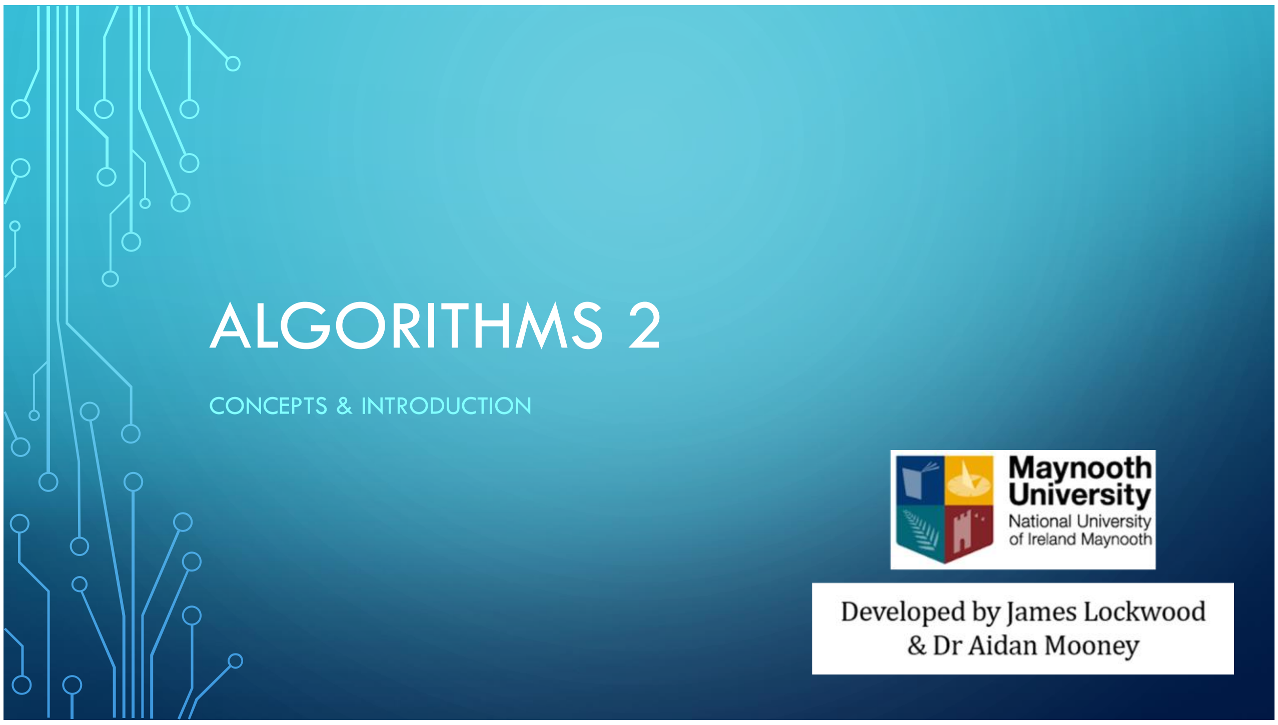
Discussion – Conclusion – 5 minutes

Recap what has been covered in the two Algorithm sections and explain that algorithms are essential when programming or doing any sort of problem solving. As we've seen from these two activities, it's important when using a computer to be specific in the instructions you give it.

We'll come back to Algorithms throughout the course, for example during the Searching & Sorting lesson, the Cryptography lesson as well as the programming sections.

Feedback – Conclusion– 5 minutes

Please get your class to fill out the algorithm feedback forms.

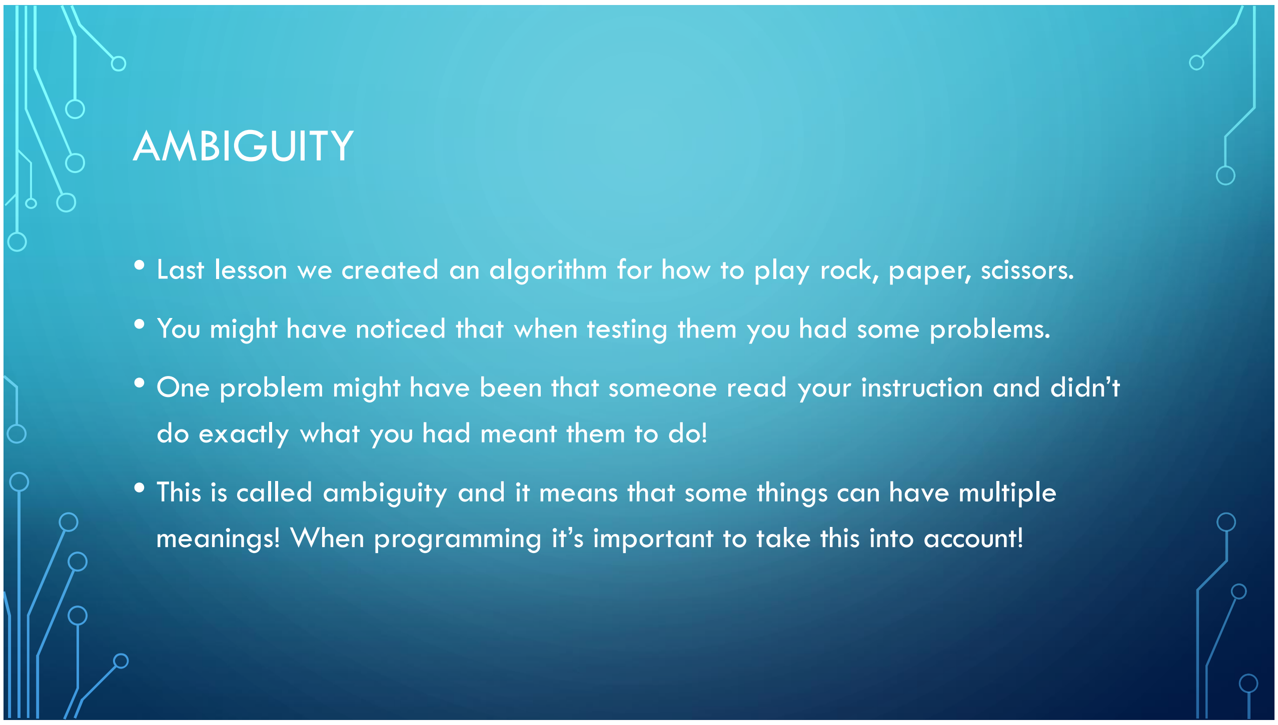


ALGORITHMS 2

CONCEPTS & INTRODUCTION

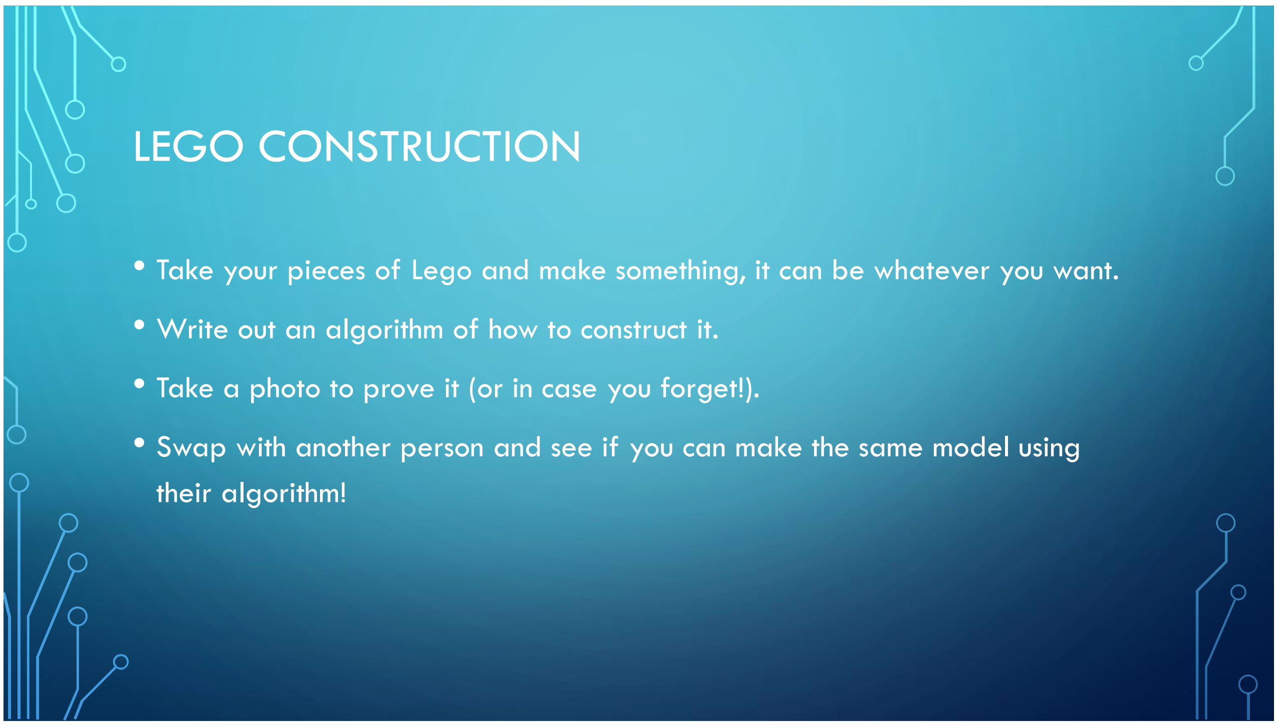
Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



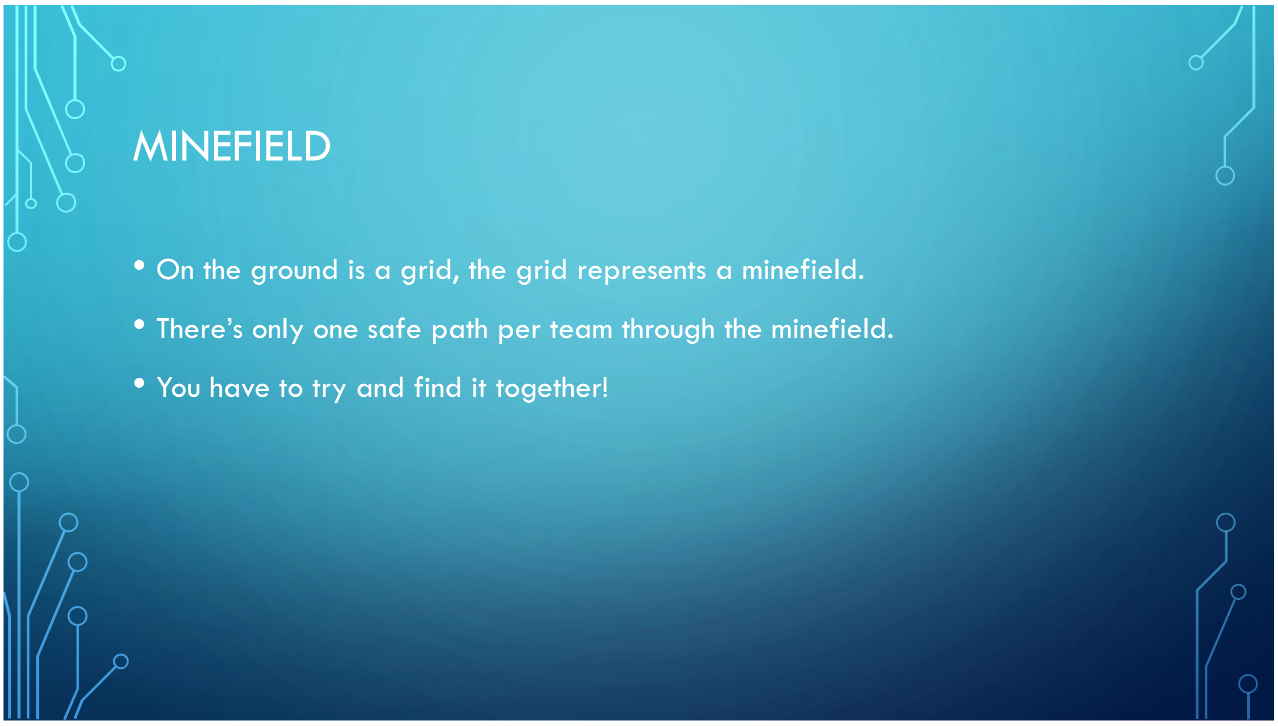
AMBIGUITY

- Last lesson we created an algorithm for how to play rock, paper, scissors.
- You might have noticed that when testing them you had some problems.
- One problem might have been that someone read your instruction and didn't do exactly what you had meant them to do!
- This is called ambiguity and it means that some things can have multiple meanings! When programming it's important to take this into account!



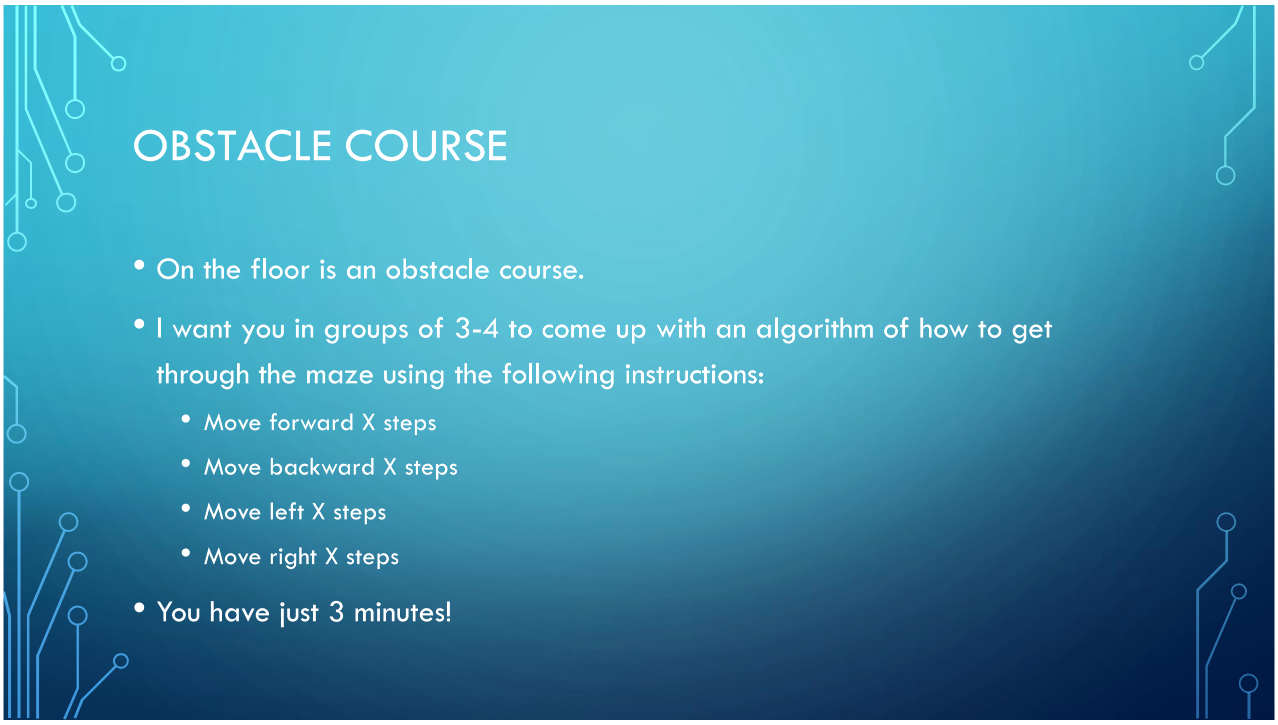
LEGO CONSTRUCTION

- Take your pieces of Lego and make something, it can be whatever you want.
- Write out an algorithm of how to construct it.
- Take a photo to prove it (or in case you forget!).
- Swap with another person and see if you can make the same model using their algorithm!



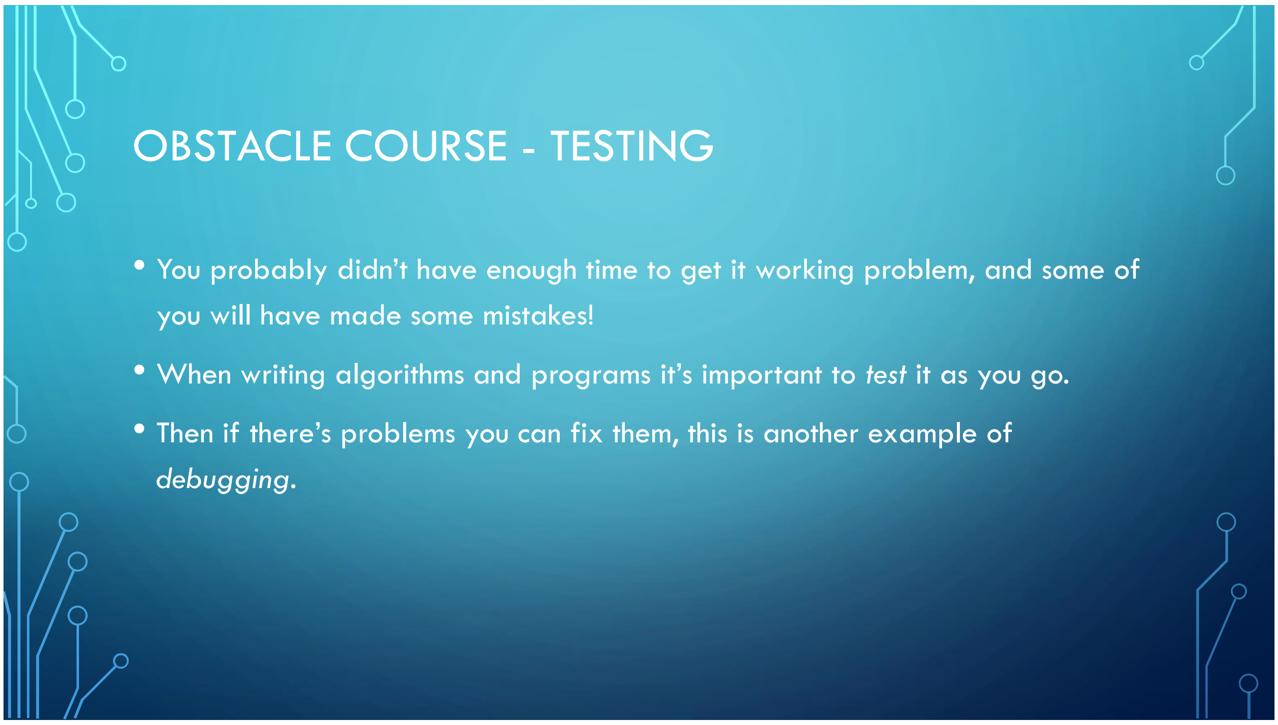
MINEFIELD

- On the ground is a grid, the grid represents a minefield.
- There's only one safe path per team through the minefield.
- You have to try and find it together!



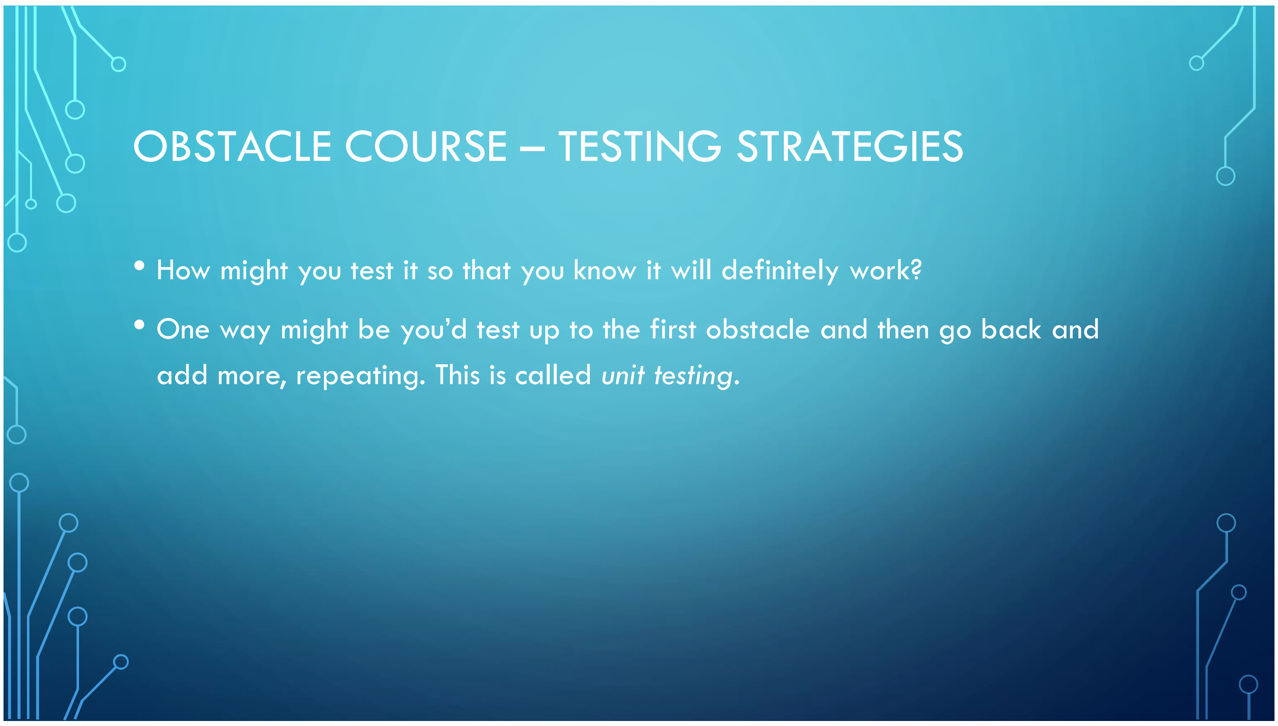
OBSTACLE COURSE

- On the floor is an obstacle course.
- I want you in groups of 3-4 to come up with an algorithm of how to get through the maze using the following instructions:
  - Move forward X steps
  - Move backward X steps
  - Move left X steps
  - Move right X steps
- You have just 3 minutes!



OBSTACLE COURSE - TESTING

- You probably didn't have enough time to get it working problem, and some of you will have made some mistakes!
- When writing algorithms and programs it's important to *test* it as you go.
- Then if there's problems you can fix them, this is another example of *debugging*.



OBSTACLE COURSE – TESTING STRATEGIES

- How might you test it so that you know it will definitely work?
- One way might be you'd test up to the first obstacle and then go back and add more, repeating. This is called *unit testing*.

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Cryptography 1 Lesson Plan

Strand: Computer Science Concepts *Duration of Lesson: 40 mins - 1 hour*

Learning Outcomes

Students will be familiar with what cryptography and cryptanalysis are.
Students will able to use some basic classical ciphers such as Caesar Shift and Substitution Ciphers.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.2 explain how the power of computing enables different solutions to difficult problems

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.7 develop algorithms to implement chosen solutions

1.8 evaluate the costs and benefits of the use of computing technology in automating processes

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

1.21 identify alternative perspectives, considering different disciplines, stakeholders and end users

2.2 use a range of methods for identifying patterns and abstract common features

Junior Certificate Coding Short Course

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

Computational Thinking Skills used

Algorithm

Computer Science concepts/topics involved

Cryptography
Encrypting
Decrypting
Brute force attacks

Frequency analysis

Resources / Materials

CS Unplugged activity 17 – Cryptographic protocols.
Peruvian coin flip sheets
Pen, paper
Tokens

The Class

Discussion – Introduction – 10 mins

Get students to write down everything they can think of when they hear cryptography, encryption, codes, internet security etc. This could be done class or in groups/pairs.

What possible reasons could they have for keeping things they send to people secret? What about in other parts of the world?

Most of the internet security we have today is based on tough maths, hard to solve equations. However, people have wanted to hide information and messages long before we had computers, can you think of any famous examples?

Activity – Introduction – 10 mins

Julius Caesar supposedly encrypted messages using a simple cipher which is known as a **shift cipher**. Every letter in the message is "shifted" by a set number of letters.

Here is an example you can put up on the board:

Hvwg wg o gsqfsh asggous

Get the students to give this a go, reminding them that the letters have all been shifted the same amount. In this case the **key** is 14, and the message is:

This is a secret message

Ask them how they went about it, some may have just randomly tried each letter until they got valid words, this is known as a **brute force attack.** Some may have noted that they had a letter on its own, which in the English language usually means it's an "a" or an "I". Explain to them that what they've just done is **decrypted or deciphered** a message. The unreadable message is known as **ciphertext** and the decrypted message is called **plaintext.**

Activity – Development – 10 mins

Tell them they're going to have a go, they may use the attached alphabet to help them. They're to shift all the letters of a message they can choose, and then write out the **encrypted** message. This first step is called **encrypting or enciphering**. They will then swap ciphertexts, keeping the key hidden and decrypt the given messages.

When they have completed this task discuss that even though it is a simple message it can still take quite a while to decrypt the message. Now imagine if you added more symbols to the alphabet, for example punctuation such as “, . ! ?”. This would get harder and harder; can they think of any other ways in which we could make it harder?

Activity – Development – 10 mins

A Caesar cipher is easy to crack, even if you had a large alphabet (say 200+ characters) a computer can easily try each option quickly until it finds valid English words. One way to make it harder to track is to use something called a **substitution** cipher. This is where each letter is randomly assigned to another one. Below is another example, let students try to decrypt this for a few minutes:

Pon xza bonn lskf ngnukui

After a few minutes, they’ll hopefully realise it’s very hard, so give them the key below and get them to decrypt it.

Original letter	A	B	C	D	E	F	G	H	I	J	K	L	M
Encrypted letter	P	Q	E	Y	N	B	I	S	K	A	T	H	C
Original letter	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Encrypted letter	U	Z	R	D	O	F	L	A	G	M	V	X	W

The message, once decrypted, should read: **Are you free this evening**

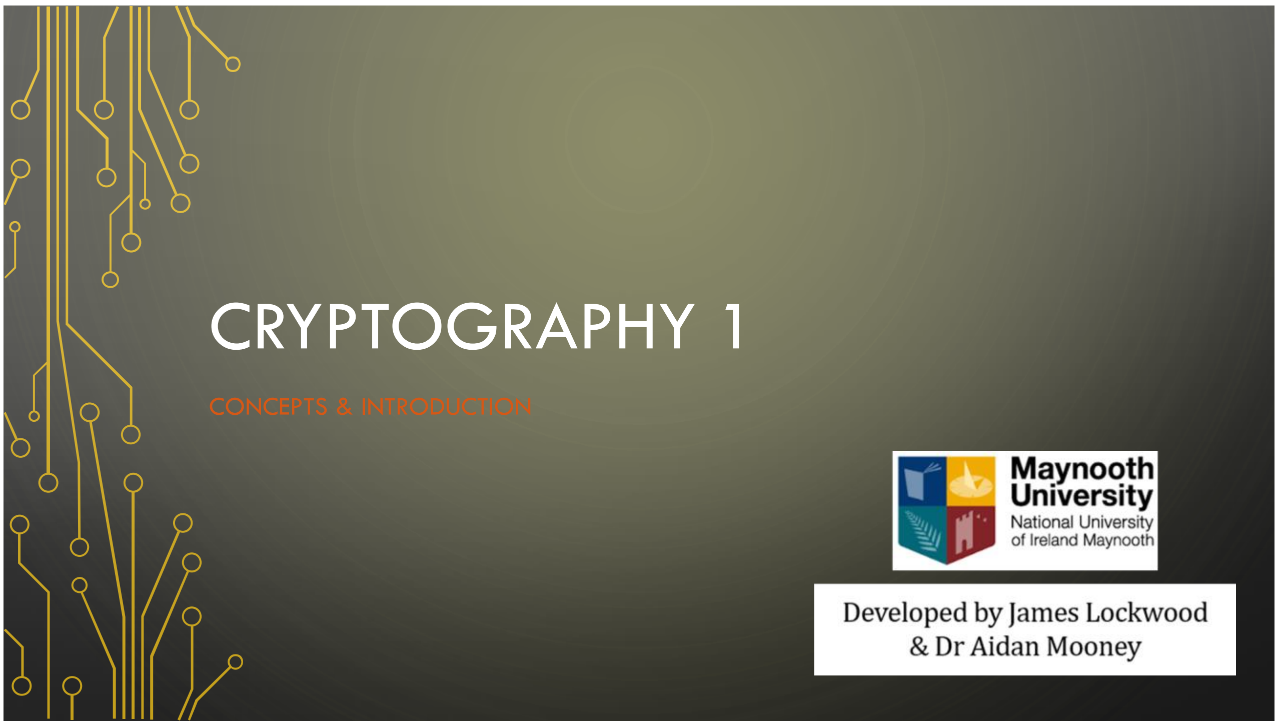
This is much harder to break as there is no common key for each letter.

Again, brute force attacks are an option, but as they’ll have seen it could take a really long time!

Another way could have been something called a **letter frequency attack.** Letters (such as ‘e’, ‘s’) are more common in English than others (such as ‘z’, ‘x’) and so you could count the number of times each letter turns up in the message and assign that to the most common English letter (‘e’) and so on. This works best for larger texts. Also, things such as double letters (“ss”, “tt” etc.) are more common with certain letters.

Feedback – Conclusion – 5 minutes

Please get your students to fill out the Cryptography feedback form.

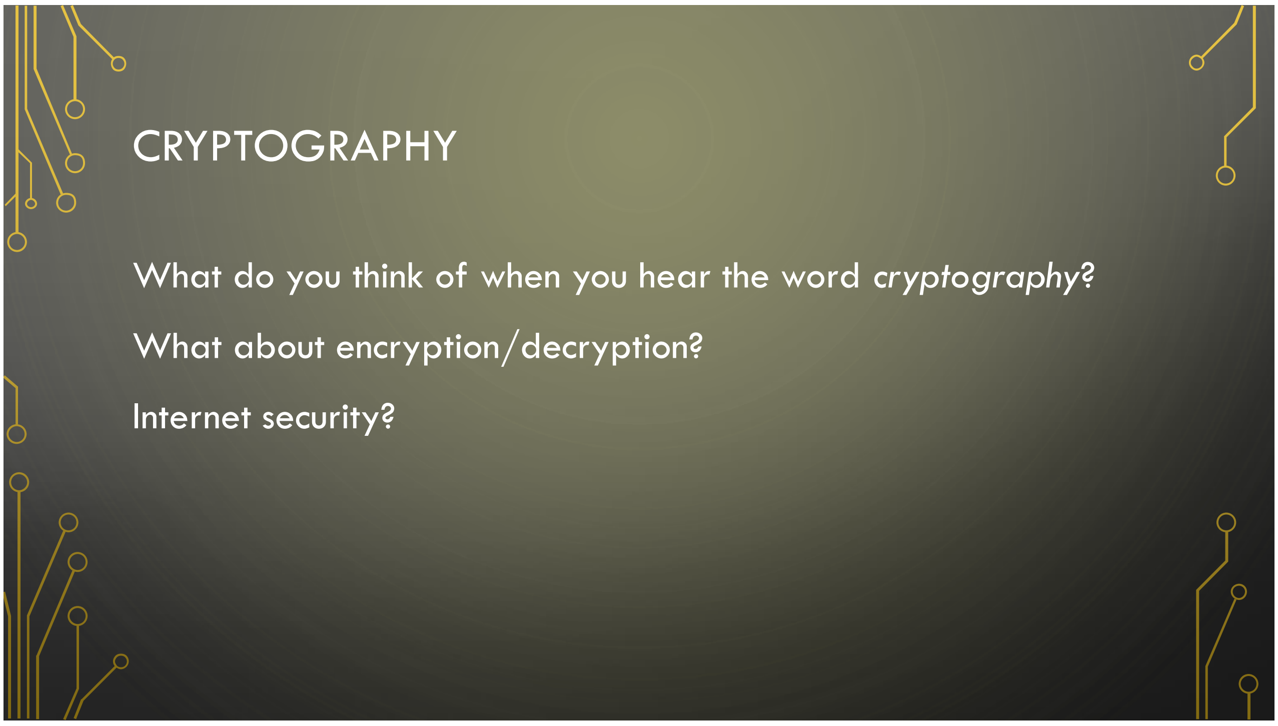


CRYPTOGRAPHY 1

CONCEPTS & INTRODUCTION

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

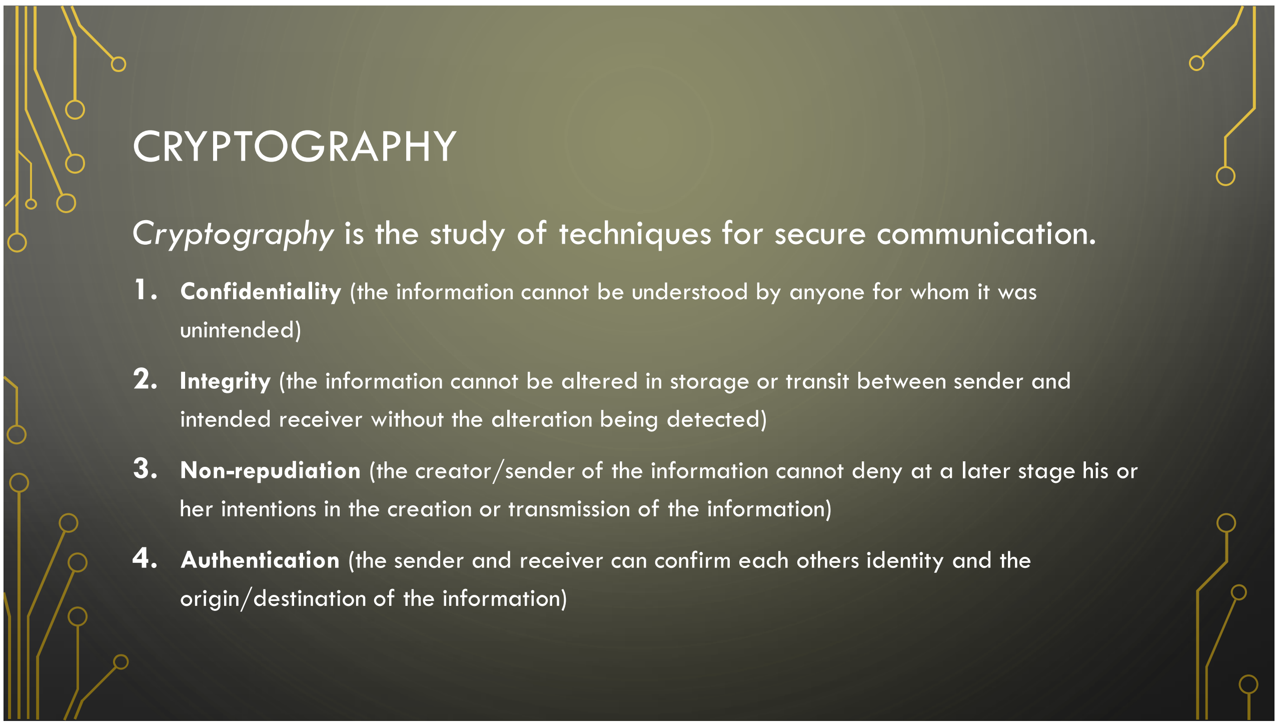


CRYPTOGRAPHY

What do you think of when you hear the word *cryptography*?

What about encryption/decryption?

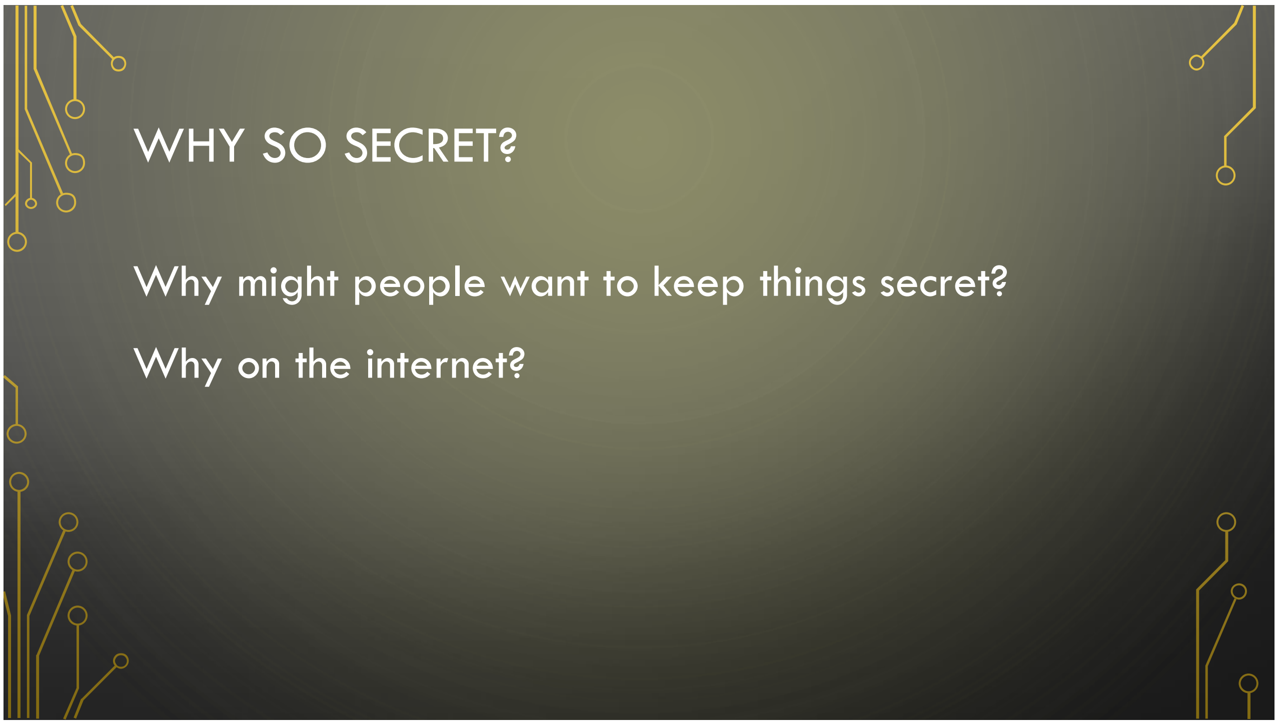
Internet security?



CRYPTOGRAPHY

Cryptography is the study of techniques for secure communication.

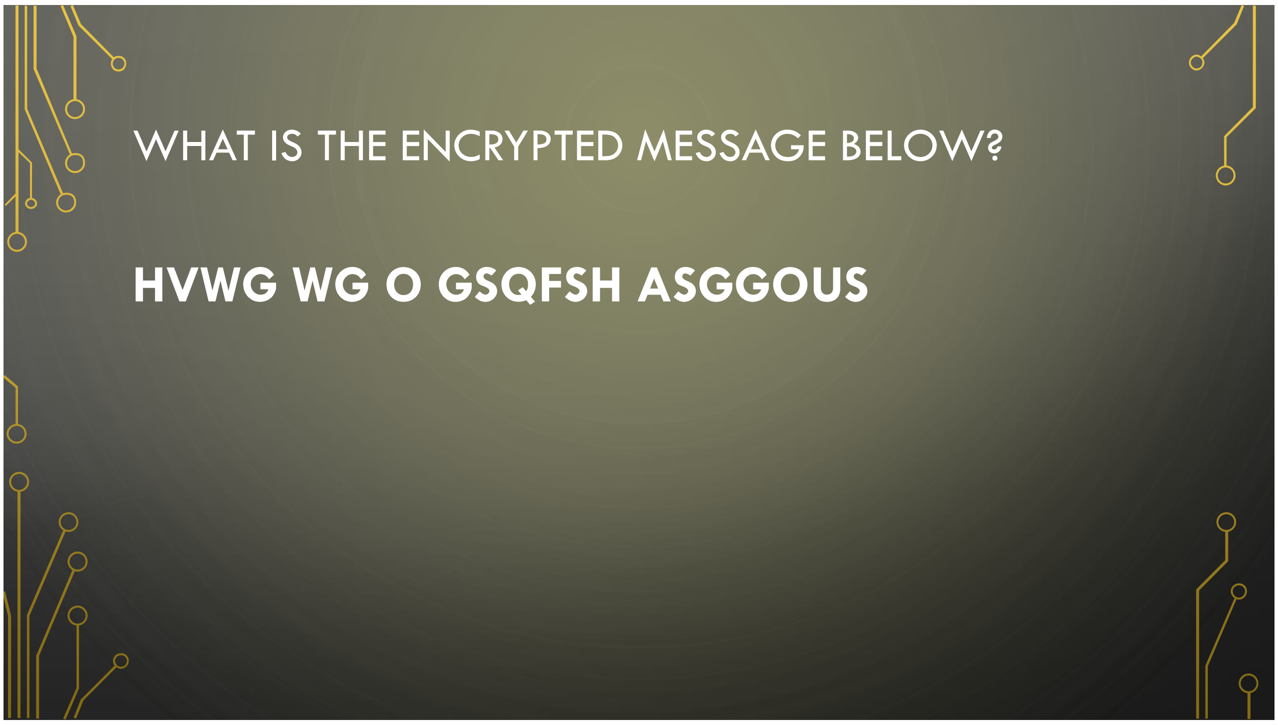
1. **Confidentiality** (the information cannot be understood by anyone for whom it was unintended)
2. **Integrity** (the information cannot be altered in storage or transit between sender and intended receiver without the alteration being detected)
3. **Non-repudiation** (the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information)
4. **Authentication** (the sender and receiver can confirm each others identity and the origin/destination of the information)



WHY SO SECRET?

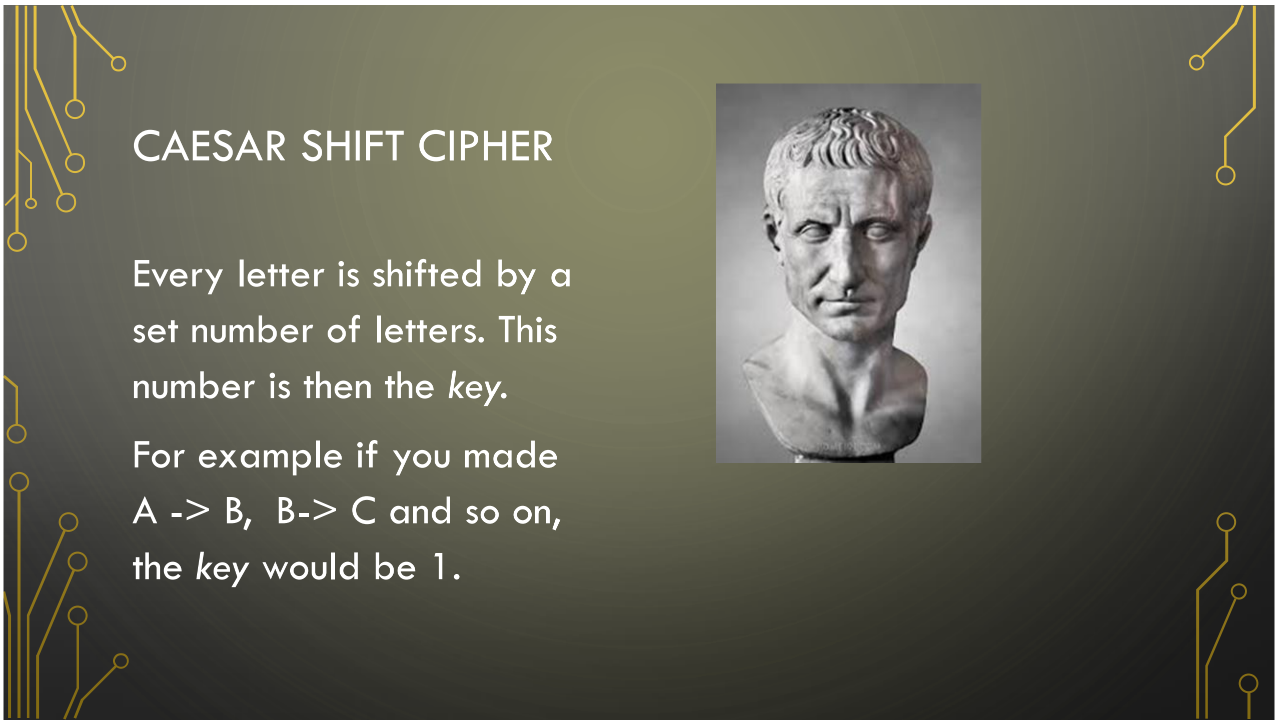
Why might people want to keep things secret?

Why on the internet?



WHAT IS THE ENCRYPTED MESSAGE BELOW?

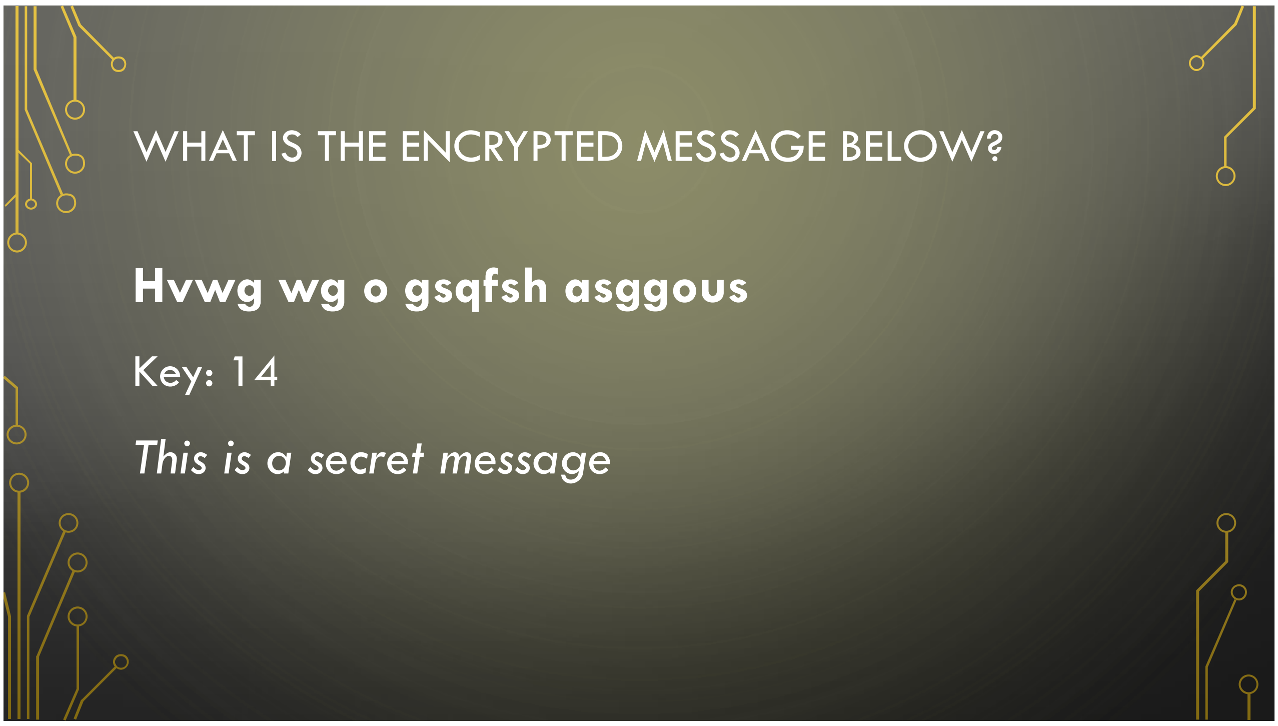
HVWG WG O GSQFSH ASGGOUS



CAESAR SHIFT CIPHER

Every letter is shifted by a set number of letters. This number is then the *key*.

For example if you made A -> B, B-> C and so on, the *key* would be 1.

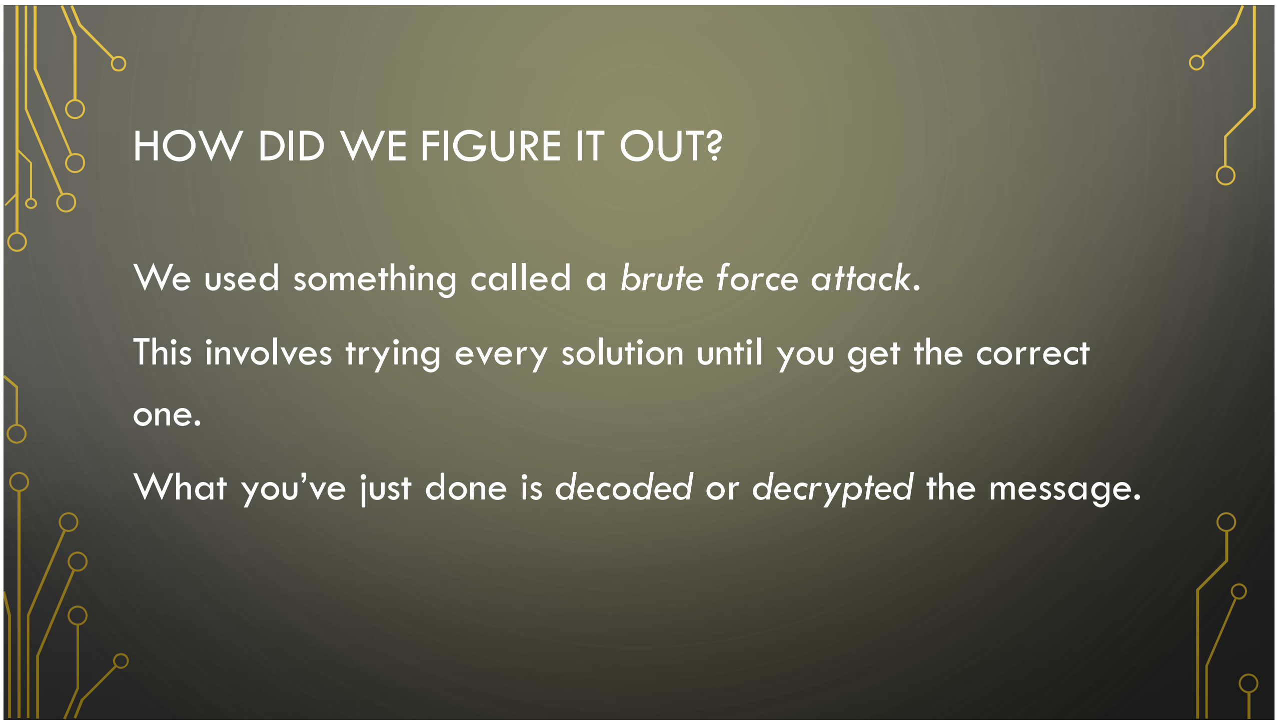


WHAT IS THE ENCRYPTED MESSAGE BELOW?

Hvwg wg o gsqfsh asggous

Key: 14

This is a secret message

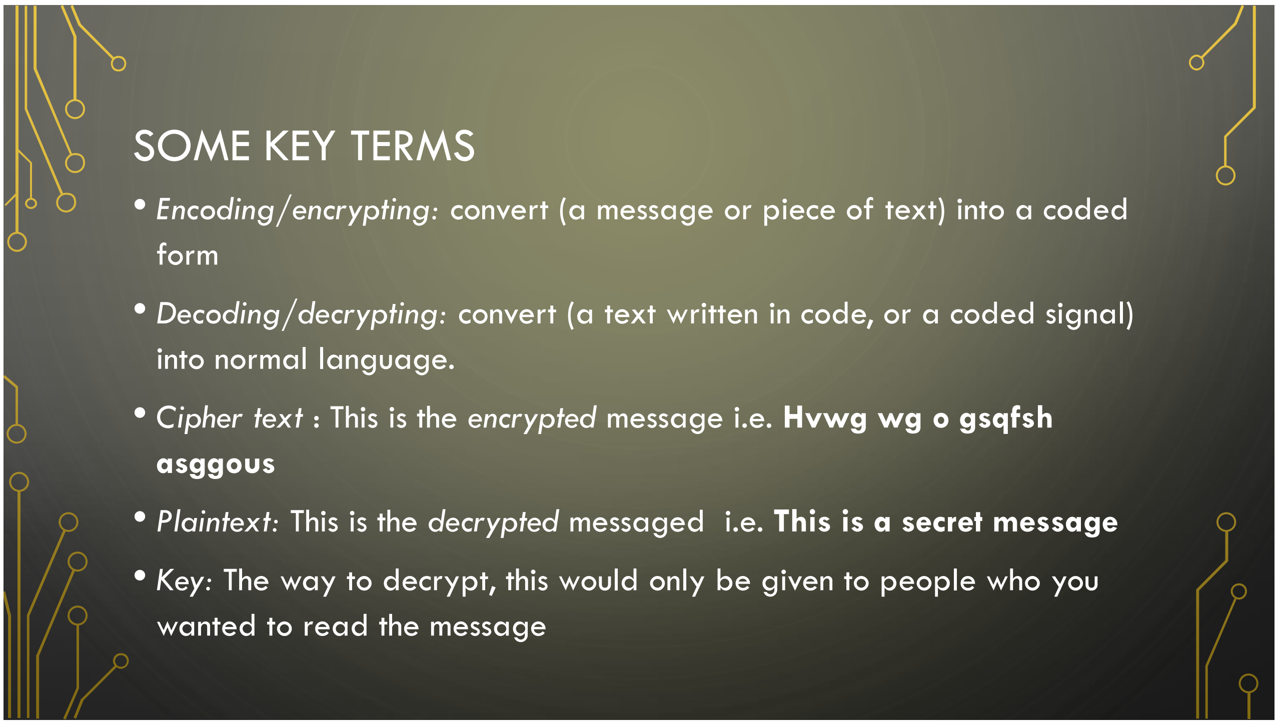


HOW DID WE FIGURE IT OUT?

We used something called a *brute force attack*.

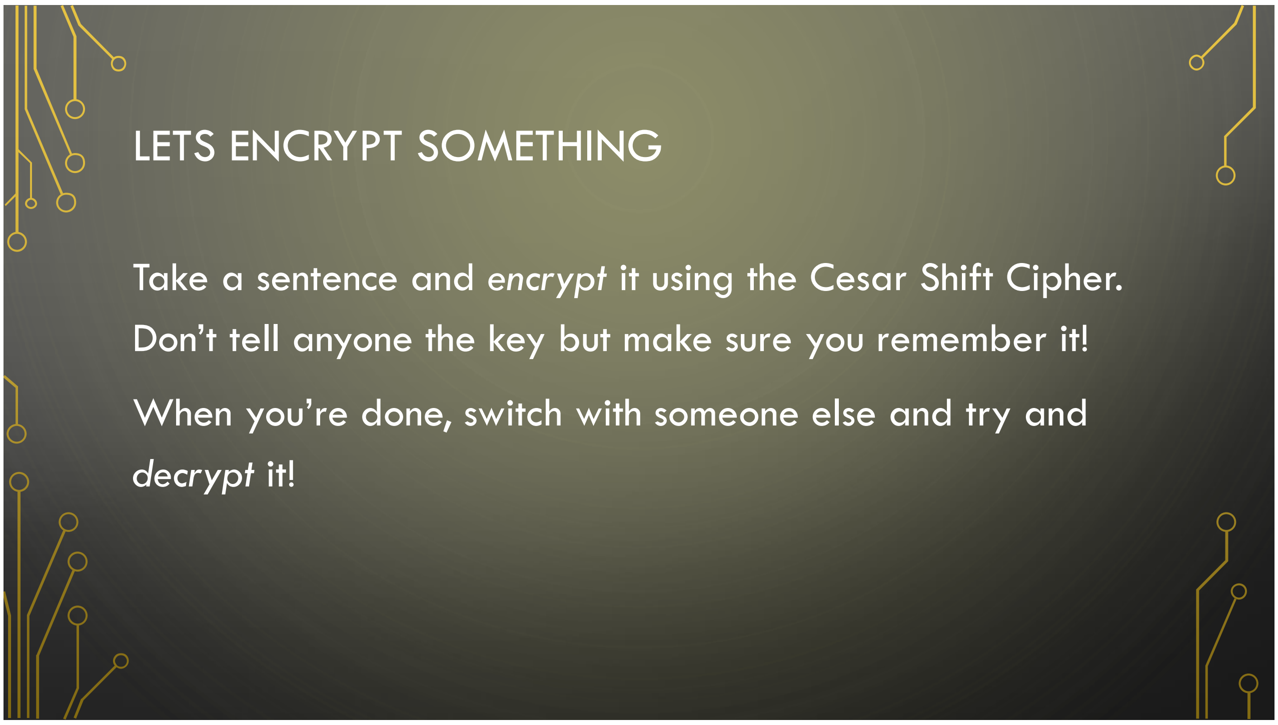
This involves trying every solution until you get the correct one.

What you've just done is *decoded* or *decrypted* the message.



SOME KEY TERMS

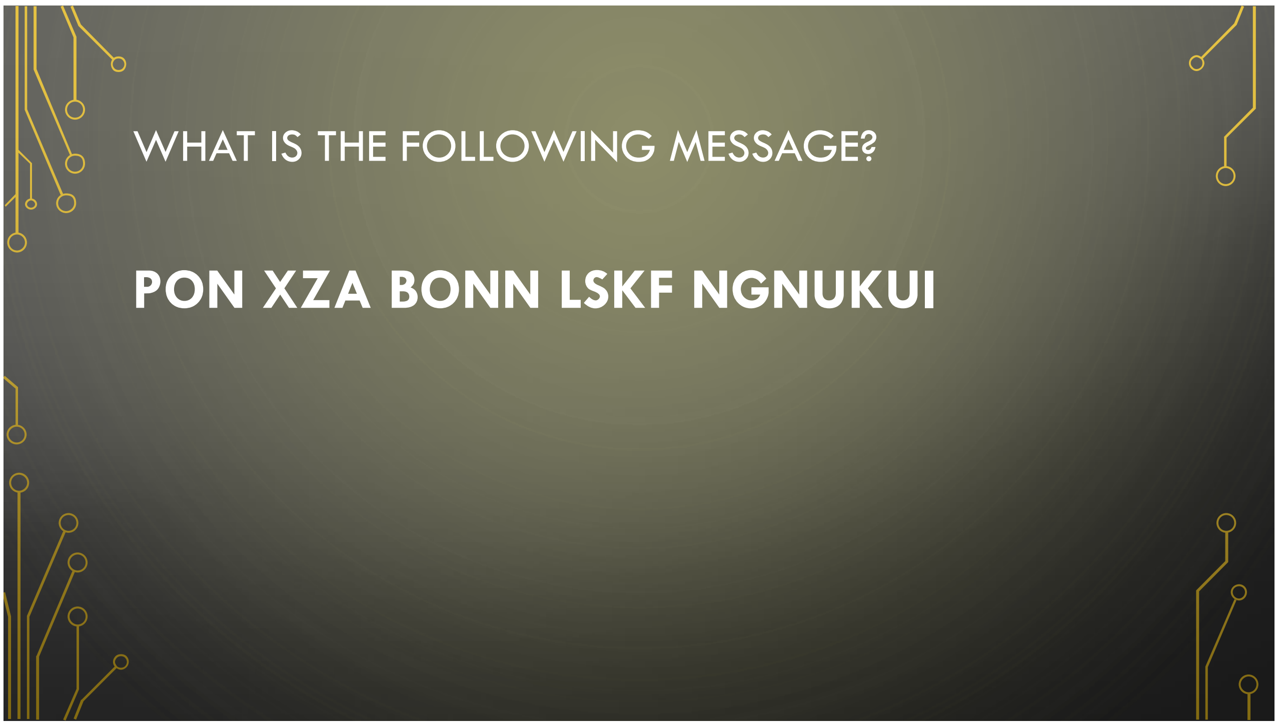
- *Encoding/encrypting:* convert (a message or piece of text) into a coded form
- *Decoding/decrypting:* convert (a text written in code, or a coded signal) into normal language.
- *Cipher text* : This is the *encrypted* message i.e. **Hvwg wg o gsqfsh asggous**
- *Plaintext:* This is the *decrypted* messaged i.e. **This is a secret message**
- *Key:* The way to decrypt, this would only be given to people who you wanted to read the message



LETS ENCRYPT SOMETHING

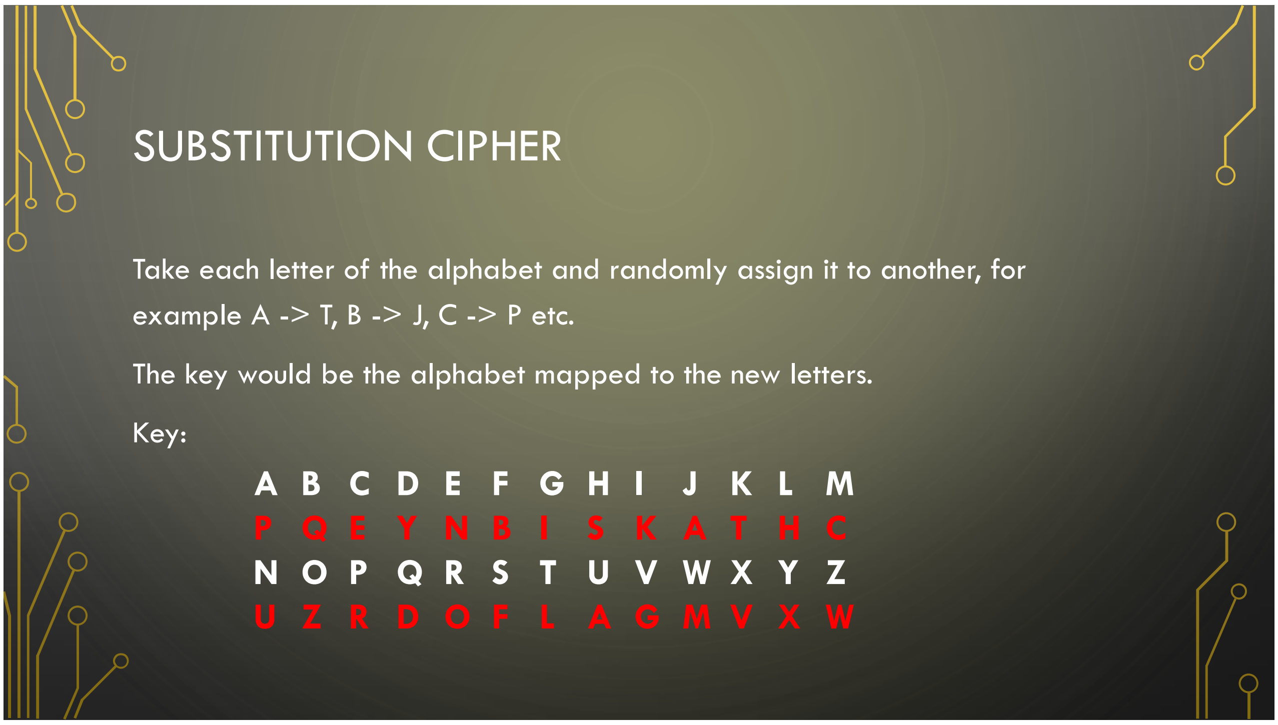
Take a sentence and *encrypt* it using the Cesar Shift Cipher. Don't tell anyone the key but make sure you remember it!

When you're done, switch with someone else and try and *decrypt* it!



WHAT IS THE FOLLOWING MESSAGE?

PON XZA BONN LSKF NGNUKUI



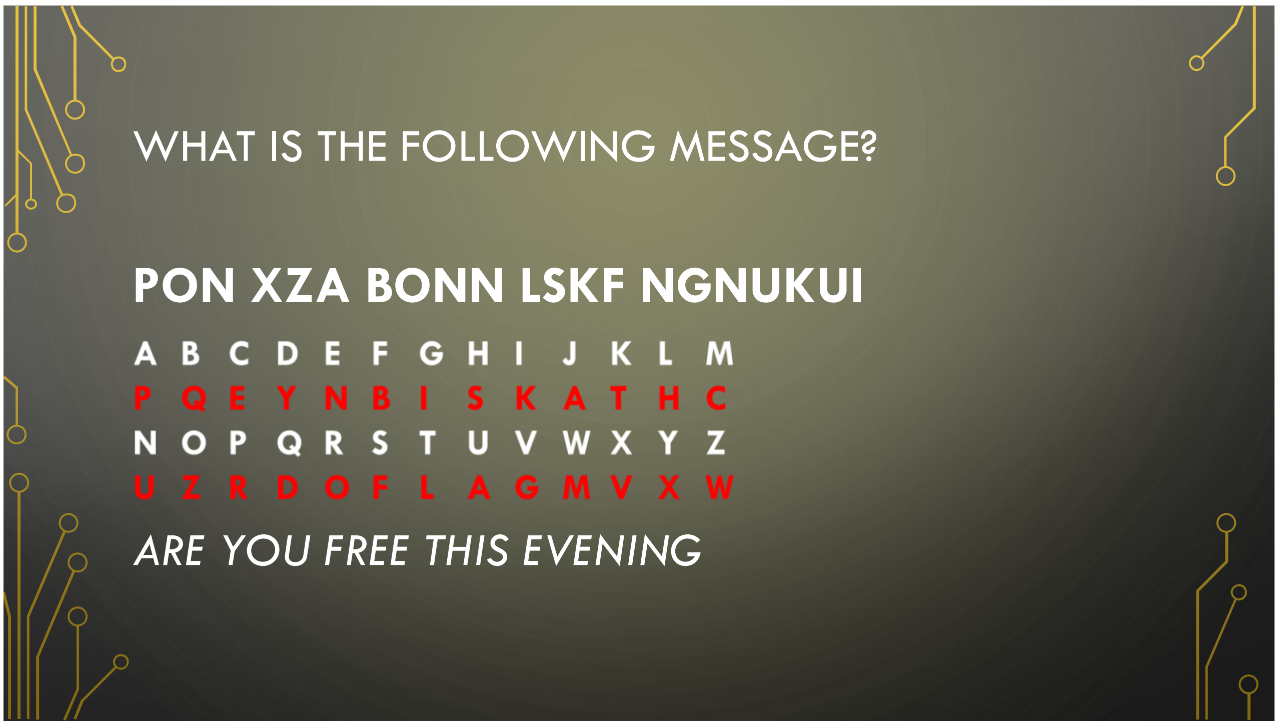
SUBSTITUTION CIPHER

Take each letter of the alphabet and randomly assign it to another, for example A -> T, B -> J, C -> P etc.

The key would be the alphabet mapped to the new letters.

Key:

A	B	C	D	E	F	G	H	I	J	K	L	M
P	Q	E	Y	N	B	I	S	K	A	T	H	C
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	Z	R	D	O	F	L	A	G	M	V	X	W

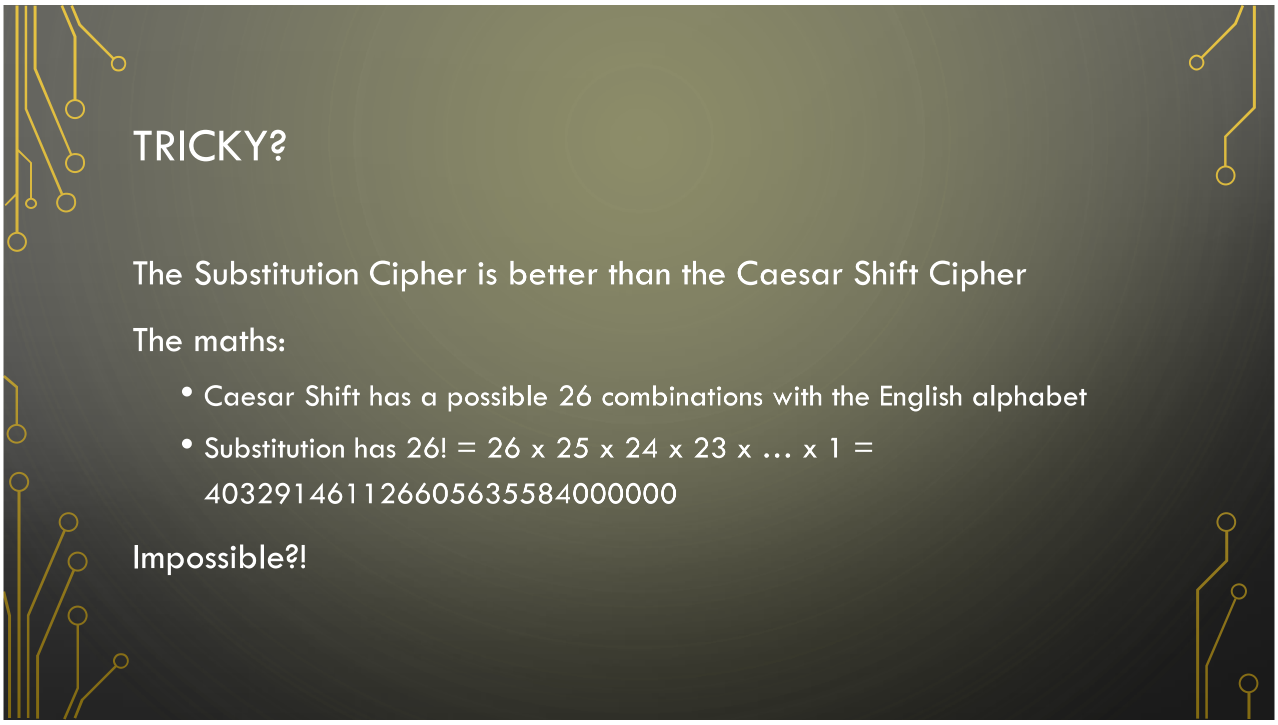


WHAT IS THE FOLLOWING MESSAGE?

PON XZA BONN LSKF NGNUKUI

A	B	C	D	E	F	G	H	I	J	K	L	M
P	Q	E	Y	N	B	I	S	K	A	T	H	C
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	Z	R	D	O	F	L	A	G	M	V	X	W

ARE YOU FREE THIS EVENING



TRICKY?

The Substitution Cipher is better than the Caesar Shift Cipher

The maths:

- Caesar Shift has a possible 26 combinations with the English alphabet
- Substitution has 26! = 26 x 25 x 24 x 23 x … x 1 = 403291461126605635584000000

Impossible?!

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Cryptography 2 Lesson Plan

Strand: Computer Science Concepts   *Duration of Lesson: 40 mins – 1 hours*

Learning Outcomes

Students will be familiar with what cryptography and cryptanalysis are.
Students will able to use a number of classical and other ciphers.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.2 explain how the power of computing enables different solutions to difficult problems

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.7 develop algorithms to implement chosen solutions

1.8 evaluate the costs and benefits of the use of computing technology in automating processes

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

1.21 identify alternative perspectives, considering different disciplines, stakeholders and end users

2.2 use a range of methods for identifying patterns and abstract common features

Junior Certificate Coding Short Course

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

Computational Thinking Skills used

Algorithm

Computer Science concepts/topics involved

Cryptography
Encrypting

Decrypting

Resources / Materials

Cryptography manual
Pre-made clues, these may need to be stuck up in places around the school/classroom

The Class

Discussion – Introduction – 5-10 mins

Recap what you covered in the previous Cryptography lesson. Explain that today they'll be going on a treasure hunt where they'll have to decrypt clues.

Activity – Development – 30 mins – 1.5 hours

Split the class into teams of 3-5 students. Make sure each team has a copy of the Cryptography manual.

There are a number of ways you can run the treasure hunt. The best way, if possible, is to get the students to move around the school where clues have been stuck on places of significance. For example, if students decrypt a clue saying something like "Hungry people love it here" it might lead them to the cafeteria where the next clue would be, which might send them to the gym, to the computer room etc. This might not be practical or feasible so below are a number of alternatives:

Have things hidden around the room, for example coloured tokens, and they must collect the token and bring it to you to get the next clue. For example, a clue might say something like "A bit of light reading" and this would take students to the book shelf. One issue with this is teams cheating by watching other teams going up to the hiding spots. You could avoid this by saying they must show you the decrypted message.

You could just get teams to race through the clues as quickly as possible, not a real hunt but more a decryption race! You don't need to come up with specific clues then it could just be sentences. This might be useful as you could link it into other lessons, topics or current events.

You could extend this by having the clues give them a problem to solve or a piece of information to find out. For example, a clue might say this once decrypted "I am a sorting algorithm" or "Ireland got independence in this year" and students will have to give you a correct answer.

The treasure hunt will take a bit of work on your part but below will be both an example but also a series of links to websites which will help you encrypt your own clues. Depending on your timeframe it's a good idea to use each encryption method in the book.

Generic school treasure hunt (if moving around):

Clue 1 – In your classroom – Caesar Shift Cipher

Obunyf wlvwsl svcl pa olyl

Hungry people love it here

Clue 2 – In cafeteria/on its door – Phone code

555 33 8 7777   555 33 2 777 66   2 22 666 88 8   8 44 33   9 666 777 555 3

Lets learn about the world

Clue 3 – Geography room/map – Reversed words

?dnuora nur a fo tib a ekil leef

Feel like a bit of a run around?

Clue 4 – Gym/running track/field – Pig latin

Ou-yay ight-may eed-nay o-tay ash-way ow-nay

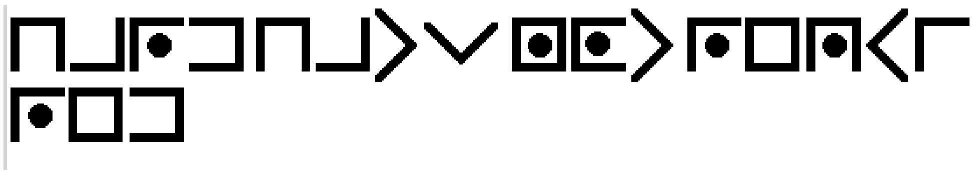
You might need to wash now!

Clue 5 – Bathrooms – Number grid

45 24 33 15   45 35   11 44 44 15 33 12 32 15

Time to assemble!

Clue 6 – Assembly hall/room - Pigpen



Hard hats not required

Clue 7 – Construction/woodwork room – Letters to numbers

8 15 23 19   25 15 21 18   19 8 1 11 5 19 16 5 1 18 5

How's your Shakespeare?

Clue 8 – English class/theatre/library –Winding way

Tobosxcahimeclxxakaetdtsx

Time to head back to class

Clue 9 – Back in your classroom - Scytale

This one you'll have to make yourself as it will change depending on the size of the object you use, see the manual for the method. An example clue might be: "I guess we made it!"

Online tools

Numerous (including PigPen, Caesar shift, Morse code, numbers, reversed word and more!)
https://cryptii.com/text/select

Caesar shift: https://www.xarg.org/tools/caesar-cipher/

Pigpen: http://crypto.interactive-maths.com/pigpen-cipher.html

Tips:

- To make them harder you could just make the clues longer.
- To make all of the number ciphers harder, just remove the spaces between the words.
- To make it longer or shorter remove clues or add as required.
- It's a good idea to put multiple of each clue (as many as teams you have) at the locations, perhaps in a polypocket. This means the team can take the clue and either return to your classroom or at least move away from the location to make it harder for the next teams.

Feedback – Conclusion – 5 minutes

Please get students to fill out the feedback form for this lesson.

CRYPTOGRAPY MANUAL

Pig Latin

ONE LANGUAGE NOT UNDERSTOOD BY MOST VEGETARIANS
ORK-PAY OP-CHAY ?
?
PIG LATIN
© Original Artist
Reproduction rights obtainable from www.CartoonStock.com
search ID: gfon348

In words that begin with consonant sounds, the initial consonant or consonant cluster is moved to the end of the word, and "ay" is added, as in the following examples:
Happy → appy-hay
Pleasant → easant-play

For words that begin with a vowel, just add “way” at the end. For example:
Apple → apple-way
Oink → oink-way

Scytale (Pronounced SKEE-tay-LEE)

Invented by the Spartans Around the 500 BC so that they could send secret messages

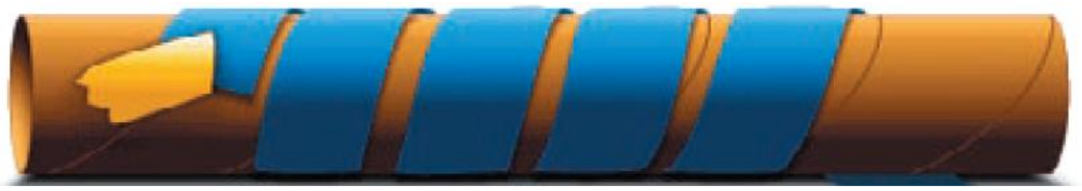
They used a stick and parchment – but we can do a simpler version

We will need

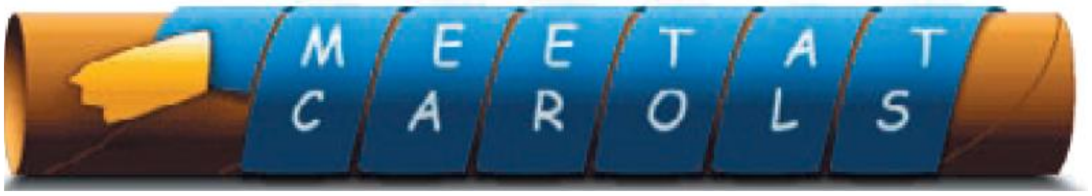
- Cylindrical object (like the cardboard tube found inside of paper towel/toilet paper rolls or even a pen)
    - Note: The person who you are sending the message to must have the same cylindrical object
- Long strip of paper, about 2 cm wide
- Something to write with
- Tape or blue-tack to fasten the paper to the cylinder

How to make A Scytale and encrypt your message

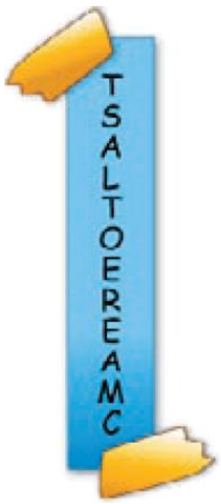
1. Tape one strip of paper to the end of the cardboard tube.
2. Wrap the remaining length of the paper around the tube in a spiral fashion. Make sure that the edges meet without overlapping. Secure the second end of paper to the tube with another piece of tape.



3. After the paper is secure, write the message across the strip of paper – just as if you were writing on a normal piece of paper from left to right. Make sure you write one letter per pass of paper. When you reach the end of a line, rotate the scytale slightly away from you and begin the next line of your message.



4. Once the message has been written, unwrap it and you will simply see a bunch of scrambled letters on the paper.



Your message has been encrypted!

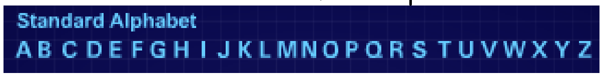
To Decrypt the message
Wrap the strip of paper around a scytale of the same size until the letters align and reveal the secret message.

The Caesar Shift

The Caesar Shift is a substitution cipher and is one of the oldest ciphers spies have ever used. Believe it or not, it was invented by Julius Caesar. In this cipher, one letter is substituted for another letter.

To encrypt and decrypt we first need to decide what letters we will be substituting. To do this we take the following steps:

Step 1
Write down each letter of the alphabet. This is called a Standard Alphabet.

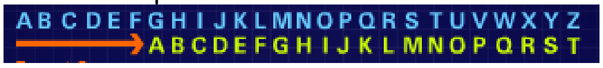


Standard Alphabet
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Step 2
Then select a number that will be used as the KEY.
For our example, we will use 7 as our Key Number.

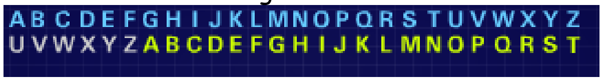
Step 3
Starting with the letter A in your Standard Alphabet, count seven letters to the right and write the letter A underneath the G.

Continue writing the remaining letters of the alphabet until you come to the end of the Standard Alphabet.



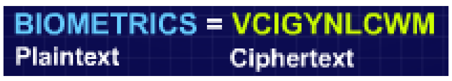
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
$\longrightarrow$ A B C D E F G H I J K L M N O P Q R S T

Step 4
Once you have reached the end of the Standard Alphabet, go back to the beginning and write in the remaining six letters.



A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
U V W X Y Z A B C D E F G H I J K L M N O P Q R S T

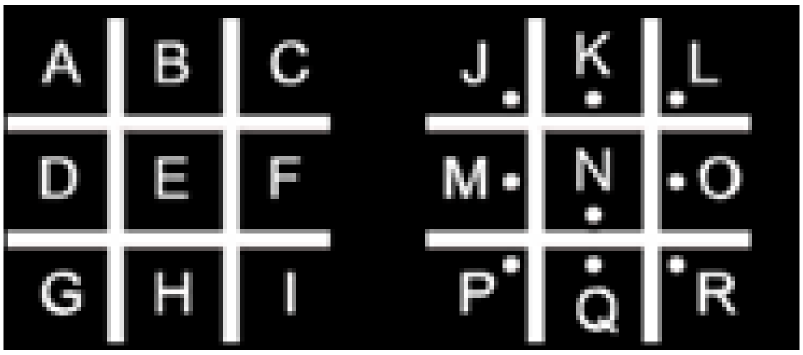
Now you are ready to encrypt or decrypt
Remember, the top alphabet is the Plaintext (The real text) and the bottom is the ciphertext (the message encrypted)

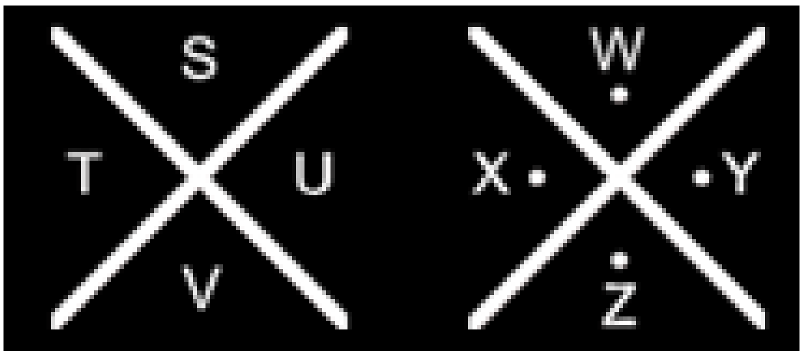


BIOMETRICS = VCIGYNLCWM
Plaintext Ciphertext

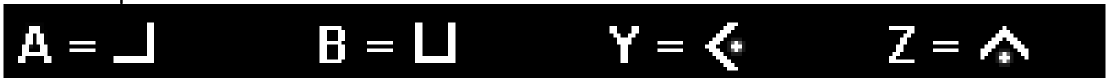
The Pigpen cipher

The Pigpen Cipher was used by Freemasons in the 18th Century to keep their records private. The cipher does not substitute one letter for another; rather it substitutes each letter for a symbol. The alphabet is written in the grids shown, and then each letter is enciphered by replacing it with a symbol that corresponds to the portion of the pigpen grid that contains the letter.

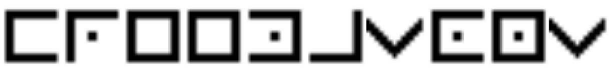




For example:



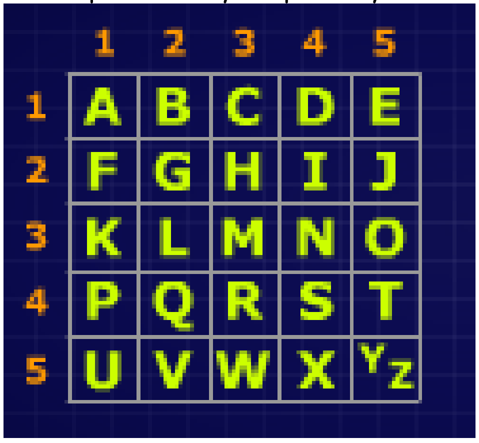
So "Freemasons" would be:



6

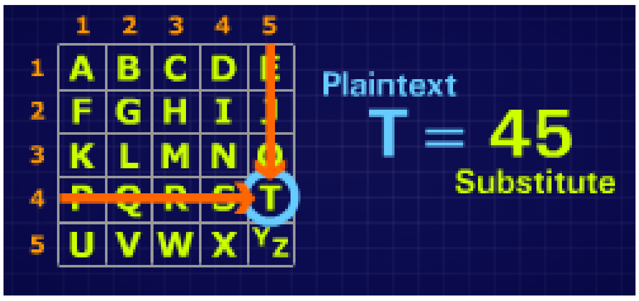
The Number Grid

This Cipher is very simple; all you need is this grid:

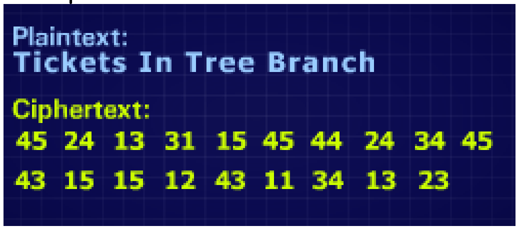


	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	K	L	M	N	O
4	P	Q	R	S	T
5	U	V	W	X	YZ

To encrypt
Begin encrypting your message by locating the first letter of your plaintext message in the grid.
Write down the row number followed by the column number.
In our example, the T will be encrypted as the number 45.



To make your message more secure, leave out the spaces between each word. For example:



Plaintext:
Tickets In Tree Branch

Ciphertext:
45 24 13 31 15 45 44 24 34 45
43 15 15 12 43 11 34 13 23

To Decrypt
To begin decrypting the ciphertext, take the first number from the message and examine the number's two digits. The first digit is the row number and the second digit is the column number.

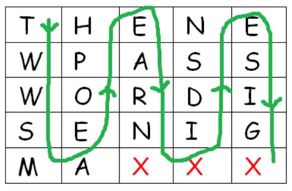
The winding way cipher

To Encrypt

- Count the number of letters used in the message.
    - Eg: "The new password is enigma" has 22 letters
- Create a matrix, or grid, that is large enough to contain your message that has the same number of rows and columns.
    - Eg our message has 22 letters so we use a grid that is 5x5
- Writing from left to right, fill in the boxes of the matrix. Use an "X" to fill in any remaining boxes in the matrix.

T	H	E	N	E
W	P	A	S	S
W	O	R	D	I
S	E	N	I	G
M	A	X	X	X

- Draw the pattern below lightly over the matrix and then copy the letters as they appear on the path.



T	H	E	N	E
W	P	A	S	S
W	O	R	D	I
S	E	N	I	G
M	A	X	X	X

TWWSMAEOPHEARNXXIDSNESIGX

To Decrypt

Using the pattern we used above, put the ciphertext into the grid

T				

T				
W				

T				
W				
W				

Etc…

Until all the grid has been filled in and the message is visible again.

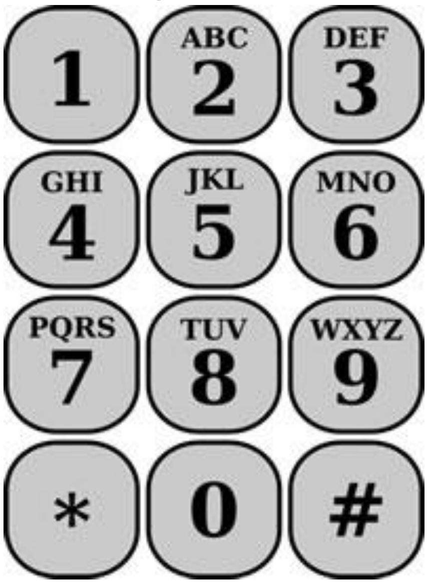
Others

There are many other things you can do to disguise your message. For cryptography to be efficient, a person you do not wish to read the message should not be able to understand the message. However, the person who the message is intended for should be able to decode the message easily. Given these rules you can try and come up with some cryptography yourself! Here are some more examples to get you started.

Example 1
Why not have numbers be letters, for example:
A=1
B=2
C=3
...
Z=26

So hello would be 8 5 12 12 15

Example 2
You could use the letters on a phone keypad to come up with a cipher. So you would get the following table:



A=2
B=22
C=222
D=3
...
Z=9999

So hello would be:
44 33 555 555 666

Example 3
You could reverse the message in number of ways for example:

Reverse in is message this.

Is actually: This message is in reverse

You don't have to just reverse the order you can exchange word order or the letter order.

May be useful

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Cryptography 3 Lesson Plan

Strand: Computer Science Concepts *Duration of Lesson: 40 minutes - 2 hours*

Learning Outcomes

Students will be familiar with what cryptography and cryptanalysis are.
Students will be familiar with different types of encryption algorithms and how they work.
Students will learn about keys and key spaces regarding how effective an encryption algorithm is.
Students will learn how to attack encryptions through methods other than brute force.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.2 explain how the power of computing enables different solutions to difficult problems

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.7 develop algorithms to implement chosen solutions

1.8 evaluate the costs and benefits of the use of computing technology in automating processes

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

1.21 identify alternative perspectives, considering different disciplines, stakeholders and end users

2.2 use a range of methods for identifying patterns and abstract common features

Junior Certificate Coding Short Course

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

Computational Thinking Skills used

Algorithm

Computer Science concepts/topics involved

Cryptography
Encrypting
Decrypting
Frequency analysis

Resources / Materials

The Class

Discussion – Introduction – 5-10 minutes

This lesson is designed to be done after doing at least one of the previous Cryptography lessons. Depending on how much you did you may have to refer back to some of the definitions given in Cryptography 1. To begin recap some of the phrases and ideas that have been introduced so far. The main ideas to focus are on why we need to encrypt data or messages, that there are lots of methods to do that, but some are better than others. In both previous lessons they'll have been introduced to at least two different encrption methods (ciphers), discuss which one/s were better (harder to decrypt) and why that is.

In todays world it is more important than ever that we have secure ways of communicating and sending information to one another. The ciphers we've looked at previously are very easy to decrypt once you know the method (and some even without knowing!) and a computer could decrypt even long messages very quickly. This lesson will be broadly divided into two sections, to begin with we'll look at some ways we can speed up attacks on text to show that the above statement is true. The second part will then look at how our online data is encrypted, thankfully in a more secure way than the Ceaser shift cipher!

Activity – Development – 15 minutes +

In the first lesson we were introduced to two Ciphers, the Ceaser shift cipher and the Substituion cipher. These are both basic ciphers but there is clearly one that is stronger than the other. The reason for this is what's called the key size. For the Ceaser shift cipher the key was a number, the number of letters that you have shifted the alphabet by. For the Substituion cipher the key would be the alphabet mapped to it's new arrangement. See if the students can think about how many possible keys there would be for these two ciphers.

The Ceaser shift cipher has a keyspace of 26, this means there's 26 possible keys. This is because we can only shift the alphabet 26 times before it ends up repeating. This is very small and easy for even our human brains to try all of them, a computer will fly through them! The Substitution cipher is a slightly stronger encryption method, it's keyspace is very large, $26! \approx 2^{88.4} \approx 403291461126605635584000000$, which is a big number! However, there is some very smart ways of breaking a substitution cipher which make it much quicker, see if the students can think of any. One way is something called a **letter frequency attack.** Letters (such as 'e', 's') are more common in English than others (such as 'z', 'x') and so you could count the number of times each letter turns up in the message and assign that to the most common English letter ('e') and so on. This works best for larger texts. Also, things such as double letters ("ss", "tt" etc.) are more common with certain letters.

Give students the below text and the letter frequency information below. In pairs, they must decrypt the text into readable English.

```
LIVITCSWPIYVEWHEVSRIQMXLEYVEOIEWHRXEXIPFEMVEWHKVSTYLXZIXLIKIIXPI
JVSZEYPERRGERIMWQLMGLMXQERIWGPSRIHMXQEREKIETXMJTPRGEVEKEITREWHEX
XLEXXMZITWAWSQWXSWEXTVEPMRXRSJGSTVRIEYVIEXCVMUIMWERGMIWXMJMGCSMW
XSJOMIQXLIVIQIVIXQSVSTWHKPEGARCSXRWIEVSWIIBXVIZMXFSJXLIKEGAEWHEP
SWYSWIWIEVXLISXLIVXLIRGEPIRQIVIIBGIIHMWYPFLEVHEWHYPSRRFQMXLEPPXL
IECCIEVEWGISJKTVWMRLIHYSPHXLIQIMYLXSJXLIMWRIGXQEROIVFVIZEVAEKPIE
WHXEAMWYEPPXLMWYRMWXSGSWRMHIVEXMSWMGSTPHLEVHPFKPEZINTCMXIVJSVLMR
SCMWMSWVIRCIGXMWYMX
```

It's decrypted form:

```
Hereupon Legrand arose, with a grave and stately air, and
brought me the beetle from a glass case in which it was
enclosed. It was a beautiful scarabaeus, and, at that time,
unknown to naturalists—of course a great prize in a
scientific point of view. There were two round black spots
near one extremity of the back, and a long one near the
other. The scales were exceedingly hard and glossy, with
all the appearance of burnished gold. The weight of the
insect was very remarkable, and, taking all things into
consideration, I could hardly blame Jupiter for his opinion
respecting it.
```

Letter	Frequency
e	12.7
t	9.1
a	8.2
o	7.5
i	7.0
n	6.7
s	6.3
h	6.1
r	6.0
d	4.3
l	4.0
c	2.8
u	2.8
m	2.4
w	2.4
f	2.2
g	2.0
y	2.0
p	1.9
b	1.5
v	1.0
k	0.8
j	0.15
x	0.15
q	0.10
z	0.07

The best way to do this, which you can tell students, is count the frequencies of all the letters. Then assign the most common letter to be e, as this is the most commonly found in the English language. You then have two options, continue changing one letter at a time with the most common, and then making changes as you go (for example if you had the word xju originally and assigned u -> e and j -> h and x ->i, you could see that x is probably actually t as the word is probably the).

The other option is to then find the most commonly occurring 2 letters together and substitute th for them, and the 3 letters is the so that should hopefully match with your chosen e.

An exercise sheet is provided which has the text and the frequency chart on it.

Frequency chart and text extract credit: https://en.wikipedia.org/wiki/Frequency_analysis also see this link for a step by step decryption.

Activity – Development – 15 minutes +

We're going to look at one final classical cipher. This is known as the Vigenère Cipher. The main difference between this cipher and the ones we've previously looked at is that it uses it's key to encrypt the messages. This means that the encryption will change depending on the key, below is an example (credit Wikipedia: https://en.wikipedia.org/wiki/Vigen%C3%A8re_cipher) of this. Basically the letters are shifted a number of positions (like the Ceasar shift cipher) but this time they are all shifted a different amount depending on the chosen key.

For example, suppose that the plaintext to be encrypted is:

ATTACKATDAWN

The person sending the message chooses a keyword and repeats it until it matches the length of the plaintext, for example, the keyword "LEMON":

LEMONLEMONLE

Each letter of the alphabet is then assigned a numerical value, this is usually A = 0, B =1, …, Z = 25.

The first letter of the plaintext, A, is paired with L, the first letter of the key. Taking the numerical value of A (0) we add it to the numerical value of L (11) and we get the new letter L (0 + 11 = 11). We can do the same with T (19) and add it to E (4) and get X (23), and so on. The only tricky part is when we get a value that is greater than 25 as this is the number of letters in our alphabet. To do this we wrap around, so if for example you had the letter T (19) in the plaintext and the letter M (12) in the key you would end up with the value 33. We then modulus this number by 26 (get the remainder or wrap around) and this would give us 7 which gives us the letter H.

Plaintext:	ATTACKATDAWN
Key:	LEMONLEMONLE
Ciphertext:	LXFOPVEFRNHR

Decryption is performed by taking the key away from the cipher text e.g. L (11) – L (11) = 0 which corresponds to A.

Give the students the below plaintext and keyword and get them to encrypt it. The below alphabet table might be useful for them as well.

Plaintext:	WEWILLMEETATTWOOCLOCK
Key:	METEORMETEORMETEORMET
Ciphertext:	IIPMZCYIXXOKFAHSQCAGD

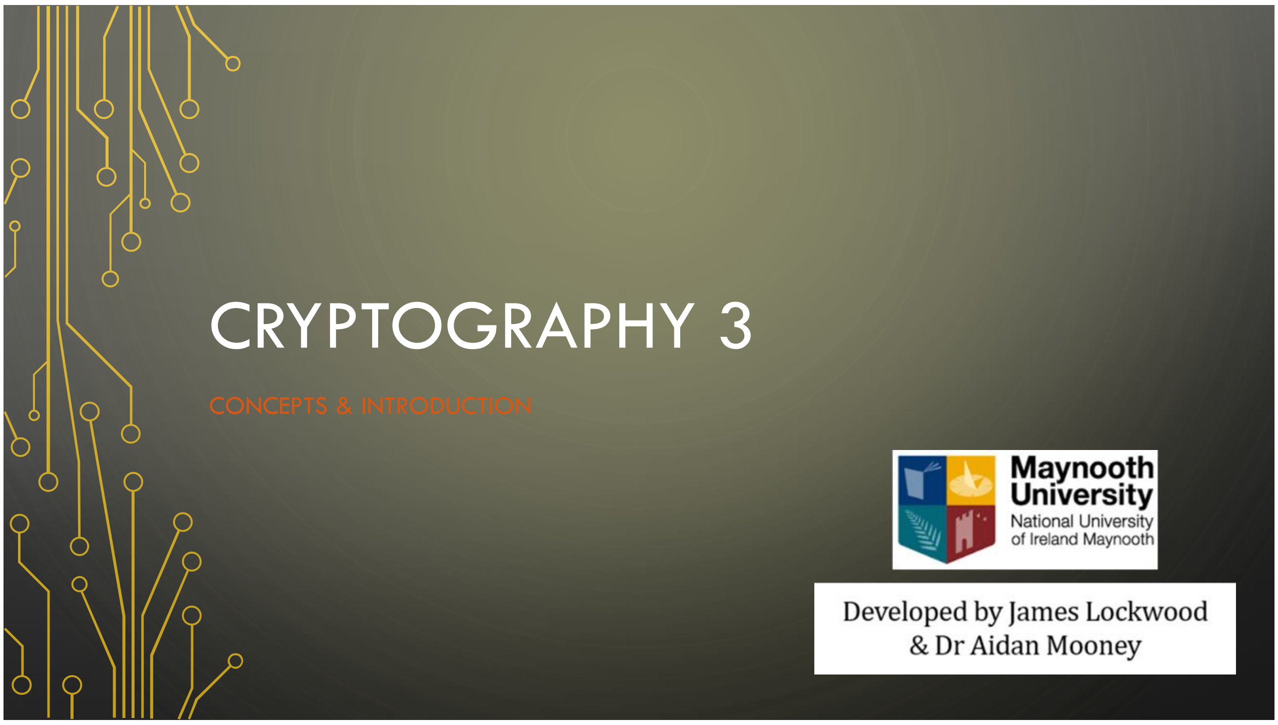
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

If you want to let them have a go and decrypting a message you can easily make your own at the following website http://www.dcode.fr/vigenere-cipher. This site allows you to encrpyt and decrypt when you know the key (and when you know less but we won't get into that!).

One key difference between the Vigenere cipher and the previous one's we've looked at is that the encryption changes depending on the key. This means (like in our first example) that the letter A can be encoded to L one time and the next time it could be encoded to O. This is known as a polyalphabetic cipher.

Feedback – Conclusion – 5 minutes

Please get students to fill out the feedback form for this lesson.

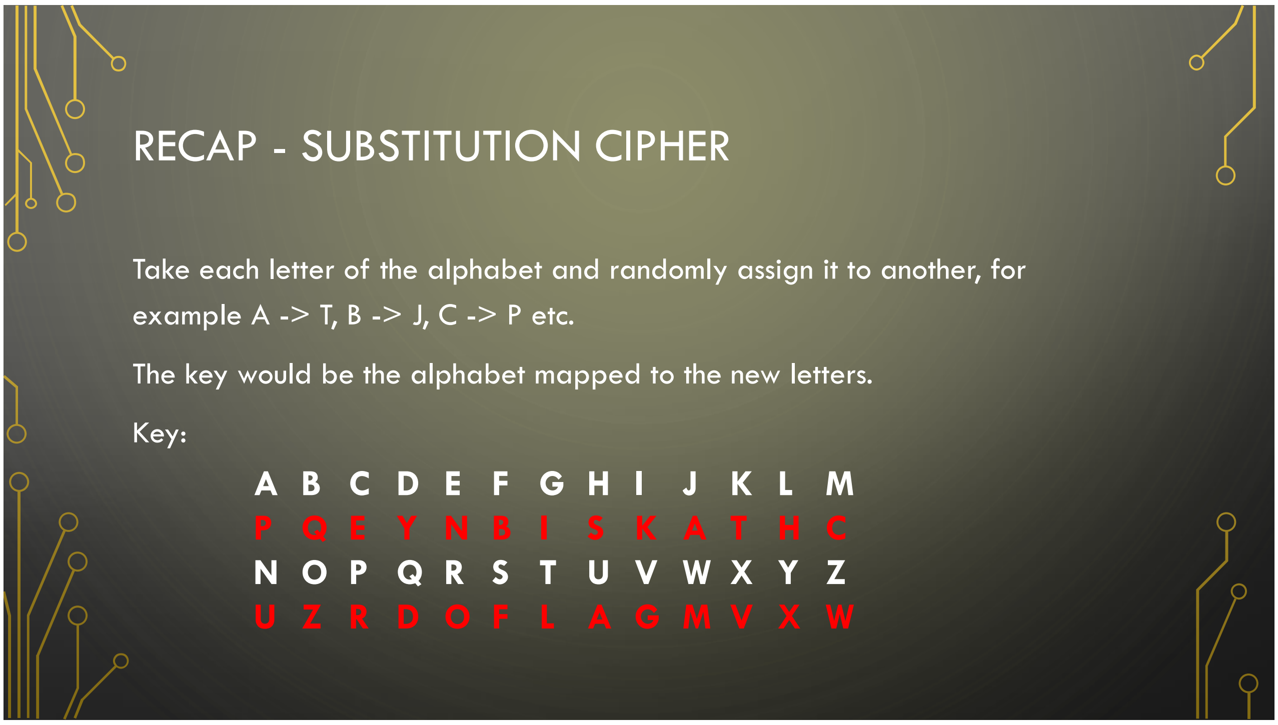


CRYPTOGRAPHY 3

CONCEPTS & INTRODUCTION

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney



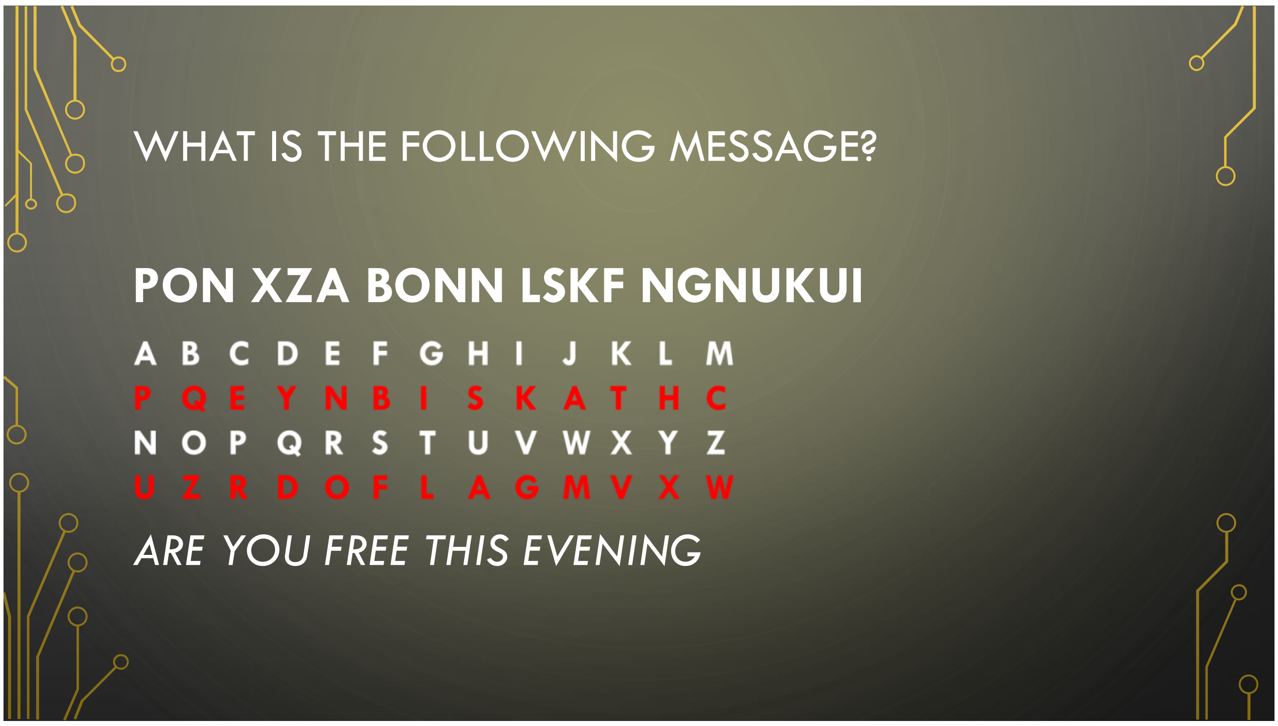
RECAP - SUBSTITUTION CIPHER

Take each letter of the alphabet and randomly assign it to another, for example A -> T, B -> J, C -> P etc.

The key would be the alphabet mapped to the new letters.

Key:

A	B	C	D	E	F	G	H	I	J	K	L	M
P	Q	E	Y	N	B	I	S	K	A	T	H	C
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	Z	R	D	O	F	L	A	G	M	V	X	W

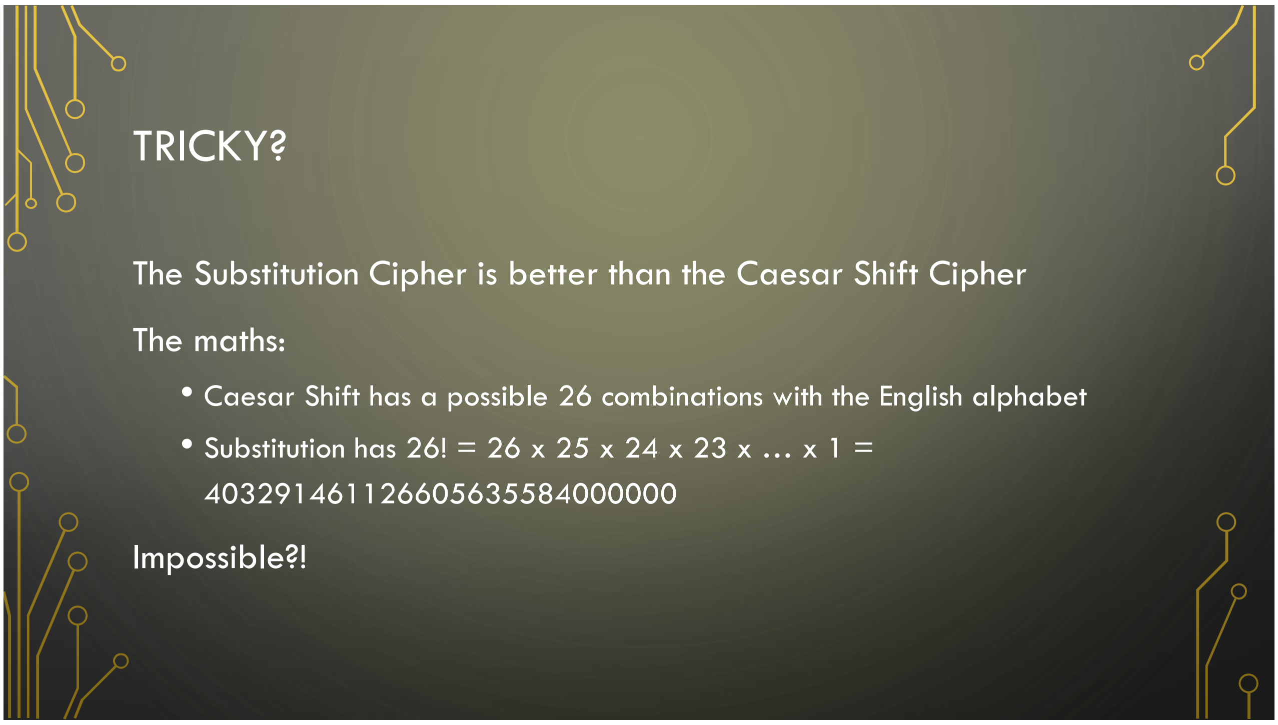


WHAT IS THE FOLLOWING MESSAGE?

PON XZA BONN LSKF NGNUKUI

A	B	C	D	E	F	G	H	I	J	K	L	M
P	Q	E	Y	N	B	I	S	K	A	T	H	C
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	Z	R	D	O	F	L	A	G	M	V	X	W

ARE YOU FREE THIS EVENING



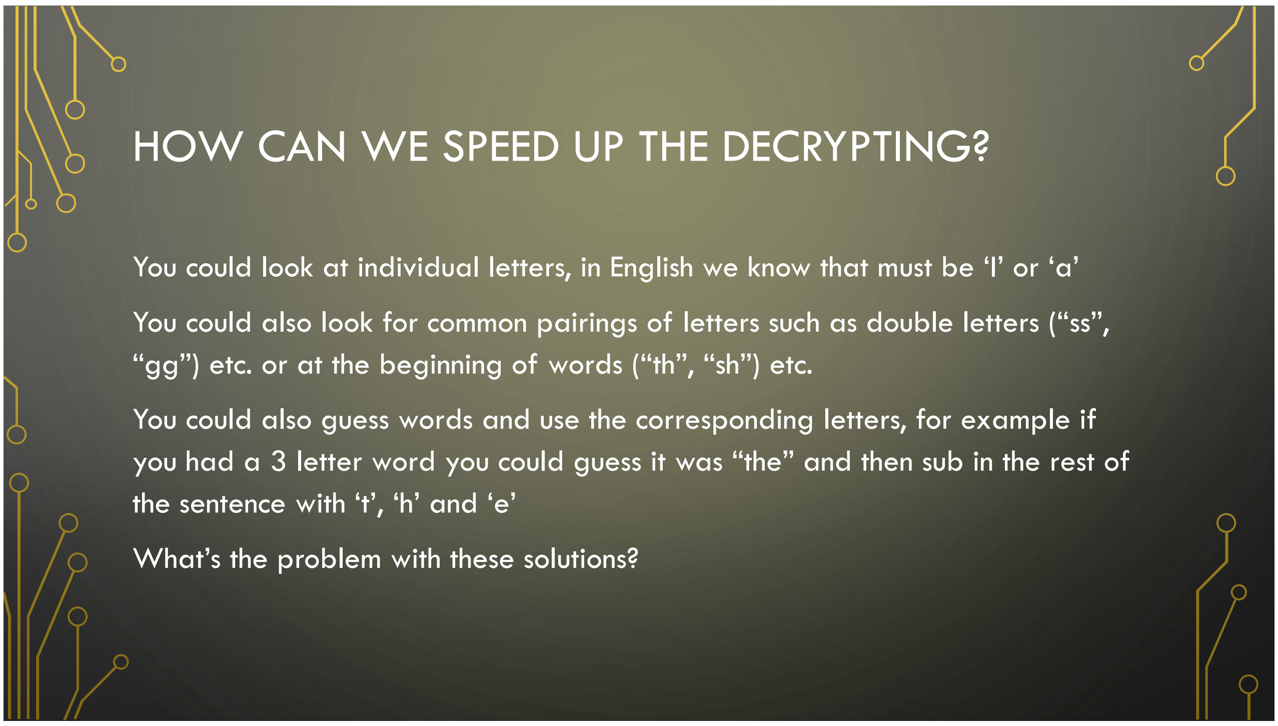
TRICKY?

The Substitution Cipher is better than the Caesar Shift Cipher

The maths:

- Caesar Shift has a possible 26 combinations with the English alphabet
- Substitution has 26! = 26 x 25 x 24 x 23 x … x 1 = 403291461126605635584000000

Impossible?!



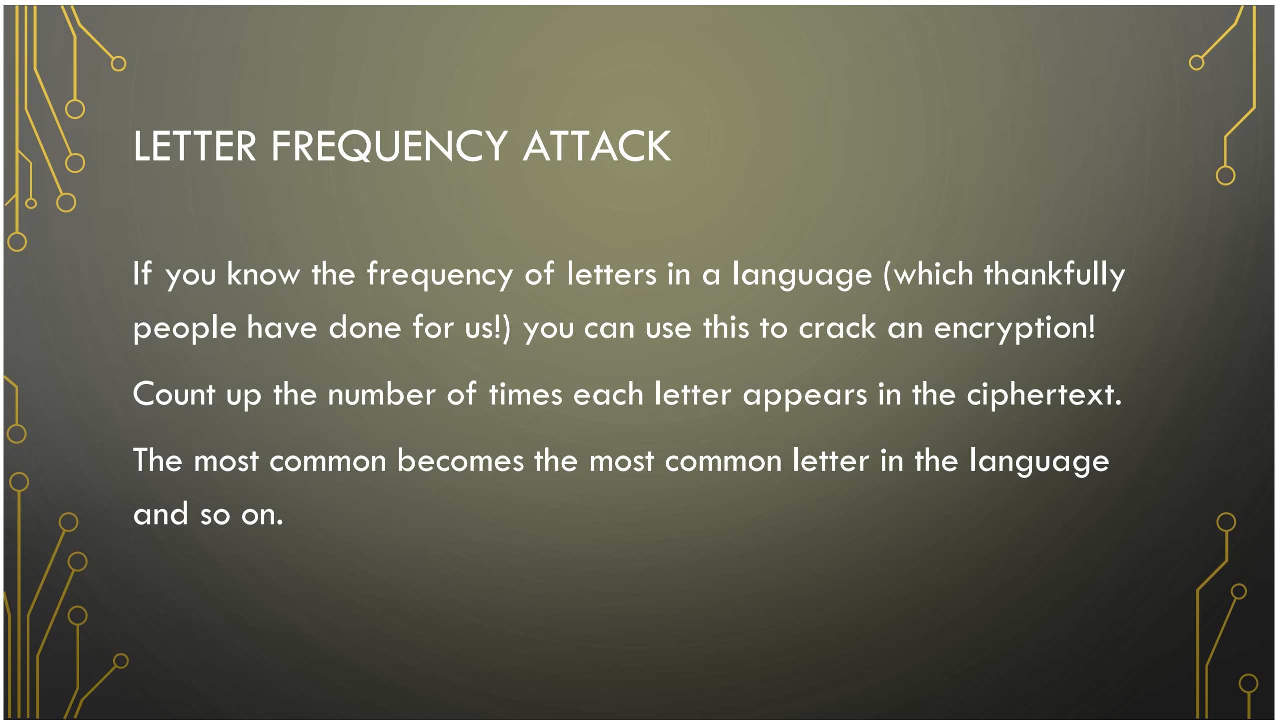
HOW CAN WE SPEED UP THE DECRYPTING?

You could look at individual letters, in English we know that must be ‘I’ or ‘a’

You could also look for common pairings of letters such as double letters (“ss”, “gg”) etc. or at the beginning of words (“th”, “sh”) etc.

You could also guess words and use the corresponding letters, for example if you had a 3 letter word you could guess it was “the” and then sub in the rest of the sentence with ‘t’, ‘h’ and ‘e’

What’s the problem with these solutions?

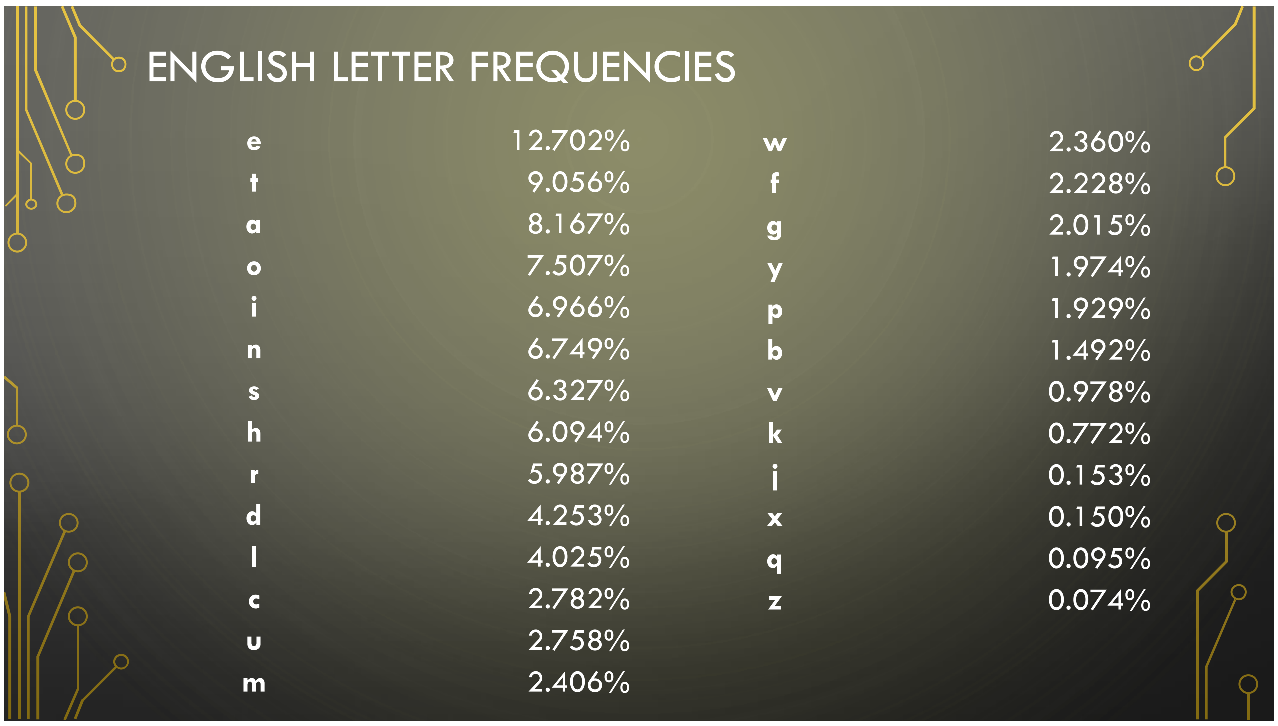


LETTER FREQUENCY ATTACK

If you know the frequency of letters in a language (which thankfully people have done for us!) you can use this to crack an encryption!

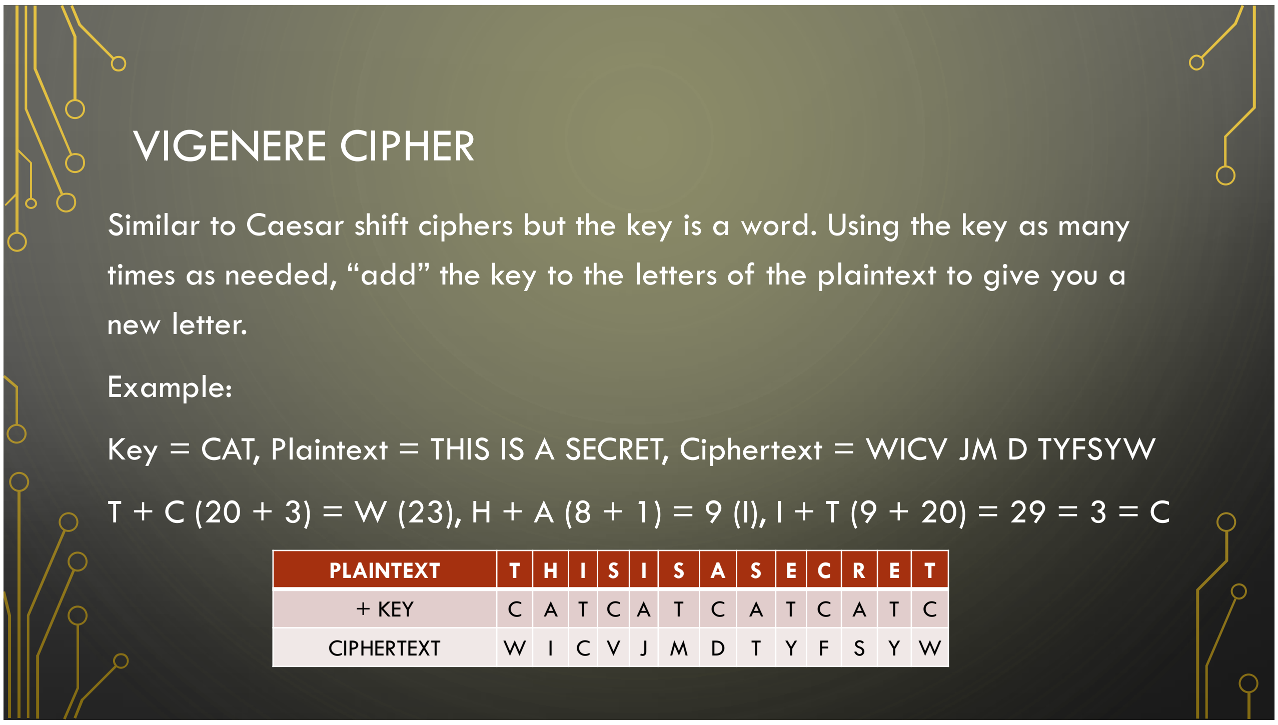
Count up the number of times each letter appears in the ciphertext.

The most common becomes the most common letter in the language and so on.



ENGLISH LETTER FREQUENCIES

e	12.702%	**w**	2.360%
t	9.056%	**f**	2.228%
a	8.167%	**g**	2.015%
o	7.507%	**y**	1.974%
i	6.966%	**p**	1.929%
n	6.749%	**b**	1.492%
s	6.327%	**v**	0.978%
h	6.094%	**k**	0.772%
r	5.987%	**j**	0.153%
d	4.253%	**x**	0.150%
l	4.025%	**q**	0.095%
c	2.782%	**z**	0.074%
u	2.758%		
m	2.406%		



VIGENERE CIPHER

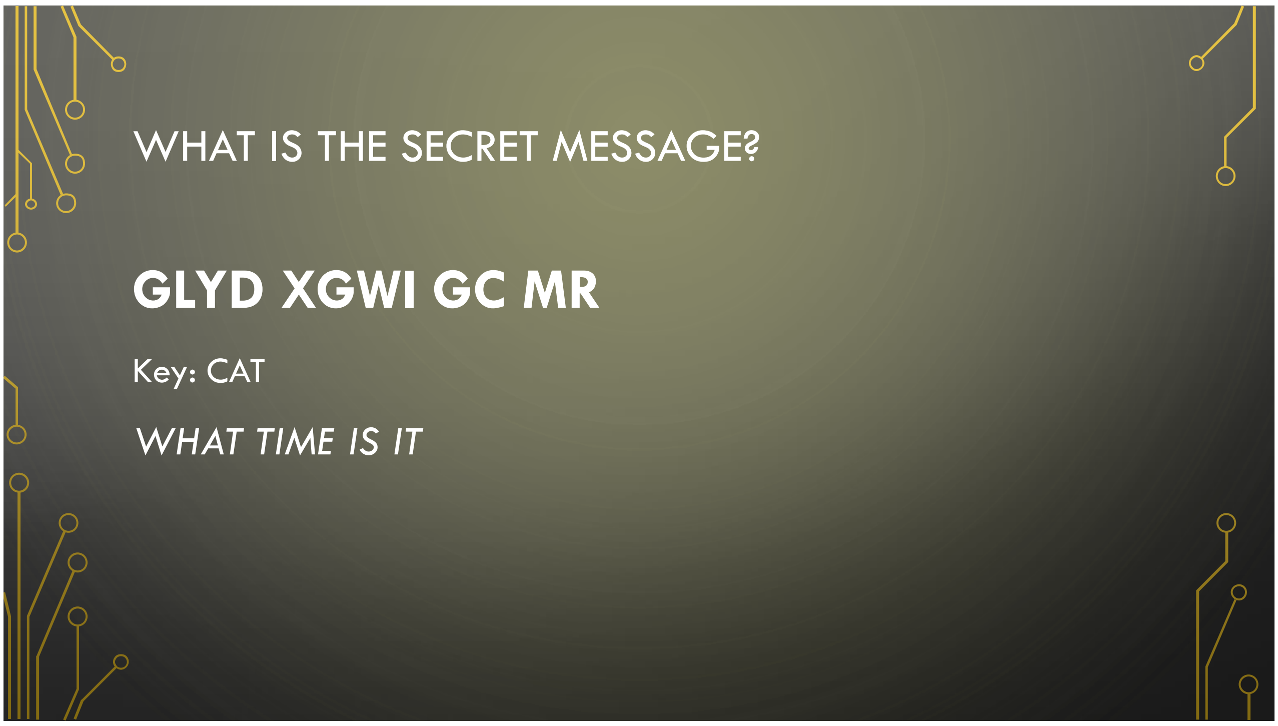
Similar to Caesar shift ciphers but the key is a word. Using the key as many times as needed, "add" the key to the letters of the plaintext to give you a new letter.

Example:

Key = CAT, Plaintext = THIS IS A SECRET, Ciphertext = WICV JM D TYFSYW

T + C (20 + 3) = W (23), H + A (8 + 1) = 9 (I), I + T (9 + 20) = 29 = 3 = C

PLAINTEXT	T	H	I	S	I	S	A	S	E	C	R	E	T
+ KEY	C	A	T	C	A	T	C	A	T	C	A	T	C
CIPHERTEXT	W	I	C	V	J	M	D	T	Y	F	S	Y	W



WHAT IS THE SECRET MESSAGE?

GLYD XGWI GC MR

Key: CAT

WHAT TIME IS IT

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Cryptography 4 Lesson Plan

Strand: Computer Science Concepts *Duration of Lesson: 40 minutes - 1 hour*

Learning Outcomes

Students will be familiar with what cryptography and cryptanalysis are.
Students will be familiar with different types of encryption algorithms and how they work.
Students will learn about public key cryptography and how it works.
Students will learn about one-way functions, what they are and why they are useful.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.2 explain how the power of computing enables different solutions to difficult problems

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.7 develop algorithms to implement chosen solutions

1.8 evaluate the costs and benefits of the use of computing technology in automating processes

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

1.21 identify alternative perspectives, considering different disciplines, stakeholders and end users

2.2 use a range of methods for identifying patterns and abstract common features

Junior Certificate Coding Short Course

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

Computational Thinking Skills used

Algorithm

Computer Science concepts/topics involved

Cryptography
Encrypting
Decrypting
Frequency analysis

Resources / Materials

The Class

Discussion – Introduction – 5-10 minutes

Over the last couple of lessons we've looked at a number of different encryption algorithms, ciphers. Get the students to think of everything that was similar and different about all of these different algorithms. Some examples include: Some use letters whilst others use numbers or shapes, they all encrypt sentences etc. One thing that might come out is that they all needed a key.

The main issue with all the ciphers we've looked at (besides the issue that they're easy for computers to crack) is that you need to have the key to decrpyt it. Wouldn't it be great if we could do it without having to pass the key to them somehow? Discuss this, explaining the differences between asymmetric and symmetric ciphers (see the slides). We're going to look today at somme asymmetric ciphers.

Activity – Introduction – 10 minutes

Our modern encryption algorithms are built on what are called one-way functions. These are introduced brillaintly in the CS Unplugged activity Tidy towns. See the provided PDF exercise sheet for a full explantion. The images and explanation are also available on the slides.

Activity – Development – 20 minutes

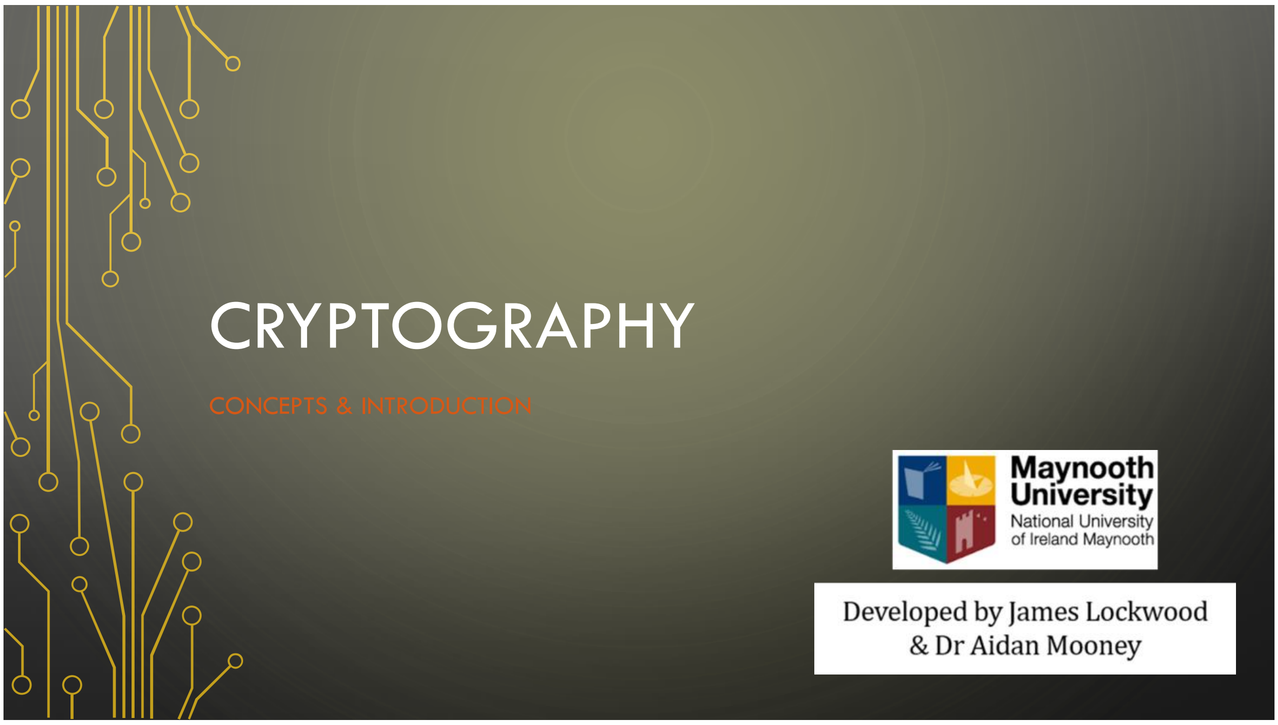
One-way functions are helpful in Cryptography as they allow things to be easy to encrypt but hard to encrypt, unless you know the key. This is further discussed in the CS Unplugged activity Private Key Cryptography. It is a great introduction to what is known as Public Key Cryptography – this allows us to send secret messages without passing a key. The images and explanation are also available on the slides.

Feedback – Conclusion – 5 minutes

Please get students to fill out the feedback form for this lesson.

Additional Activities

CS Unplugged provide two more cryptography based lessons on protocools called the Peruvian Coin Flip and another called Sharing Secrets.

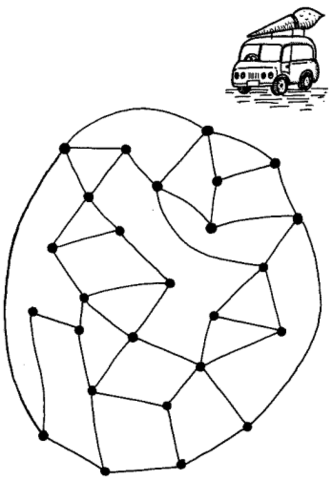


CRYPTOGRAPHY

CONCEPTS & INTRODUCTION

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney



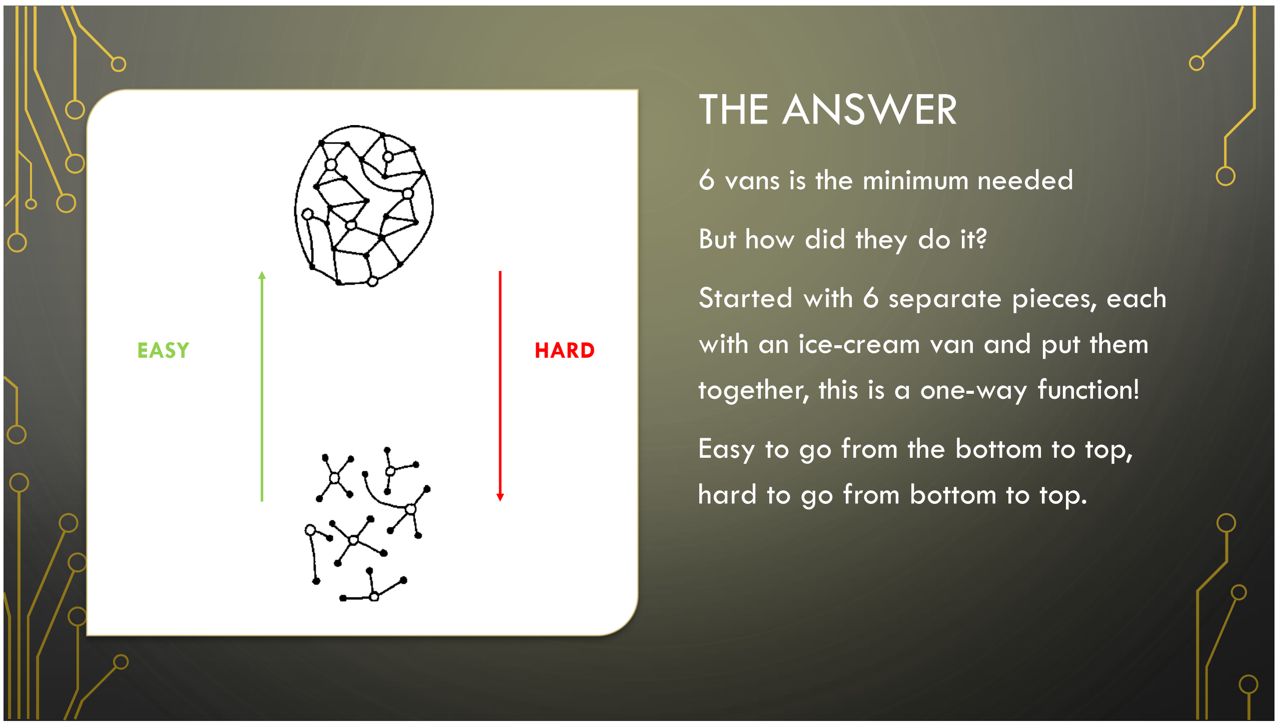
TOURIST TOWN

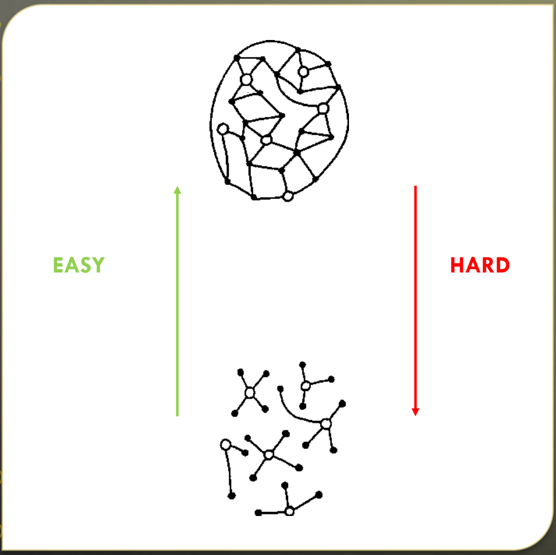
The lines are streets and the dots are street corners.

The town lies in a very hot country, and in the summer season ice-cream vans park at street corners and sell ice-creams to tourists.

We want to place the vans so that anyone can reach one by walking to the end of their street and then at most one block further.

The question is, how many vans are needed and on which intersections should they be placed?





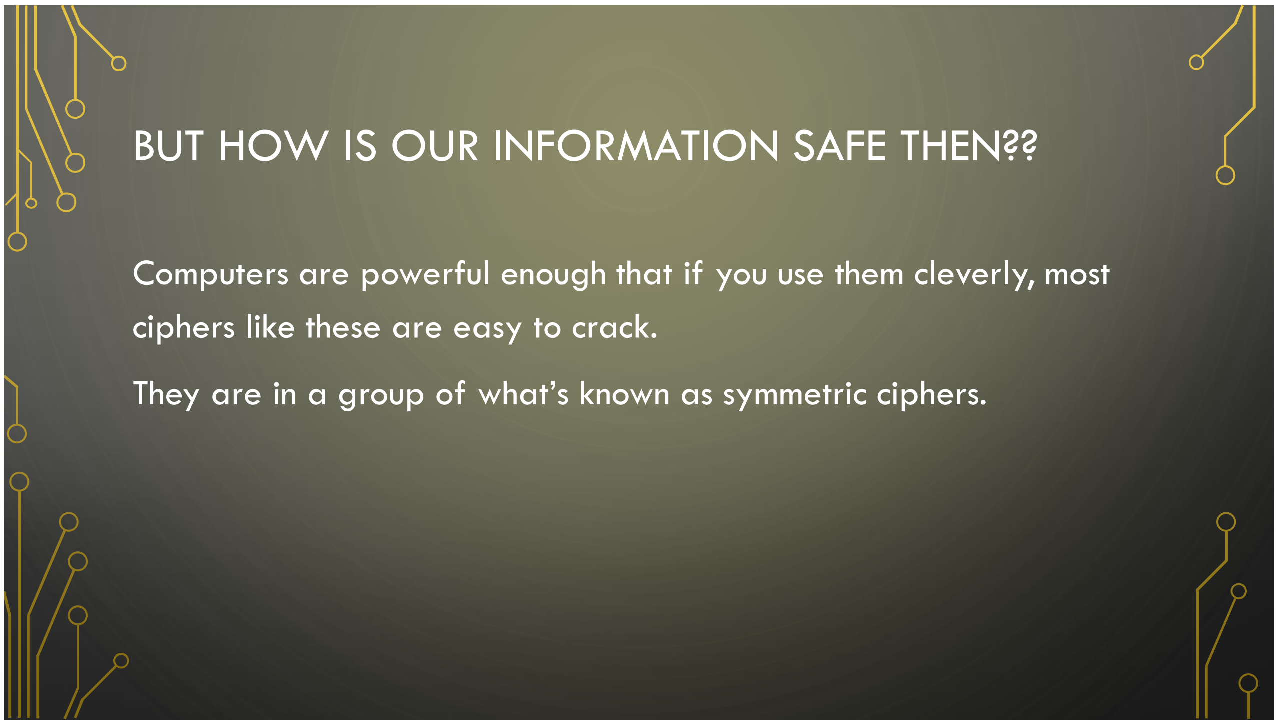
THE ANSWER

6 vans is the minimum needed

But how did they do it?

Started with 6 separate pieces, each with an ice-cream van and put them together, this is a one-way function!

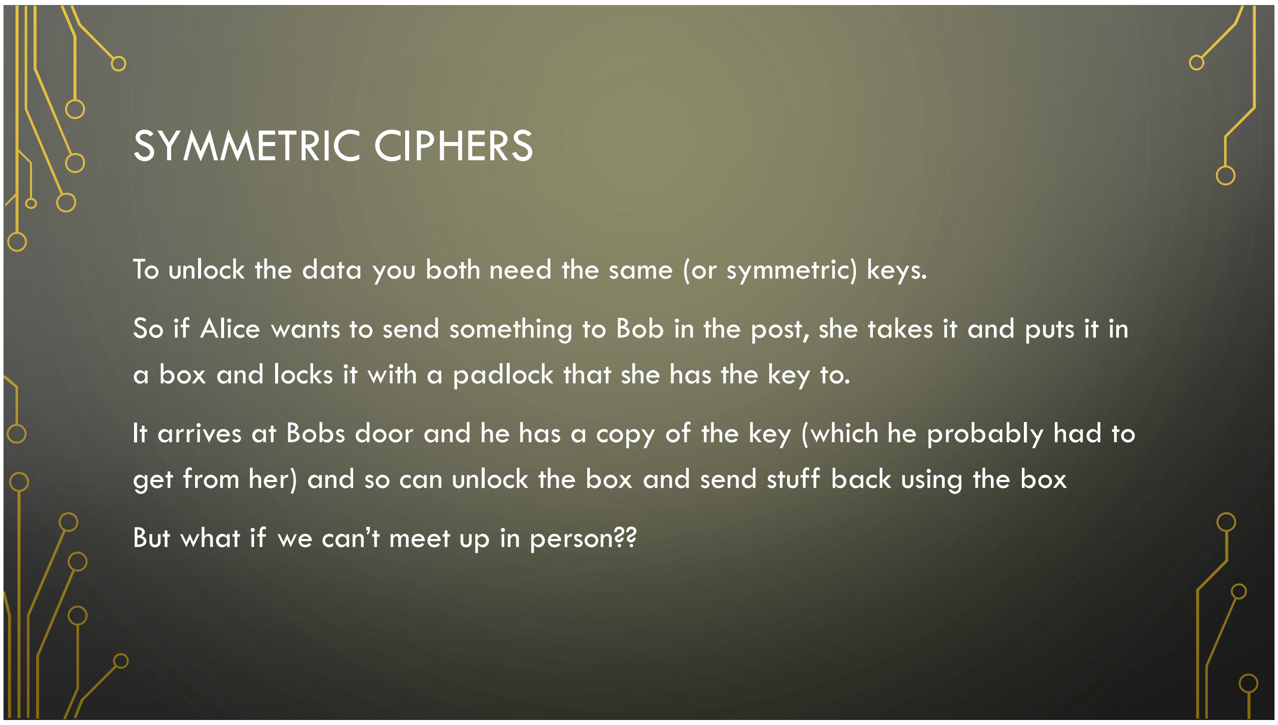
Easy to go from the bottom to top, hard to go from bottom to top.



BUT HOW IS OUR INFORMATION SAFE THEN??

Computers are powerful enough that if you use them cleverly, most ciphers like these are easy to crack.

They are in a group of what's known as symmetric ciphers.



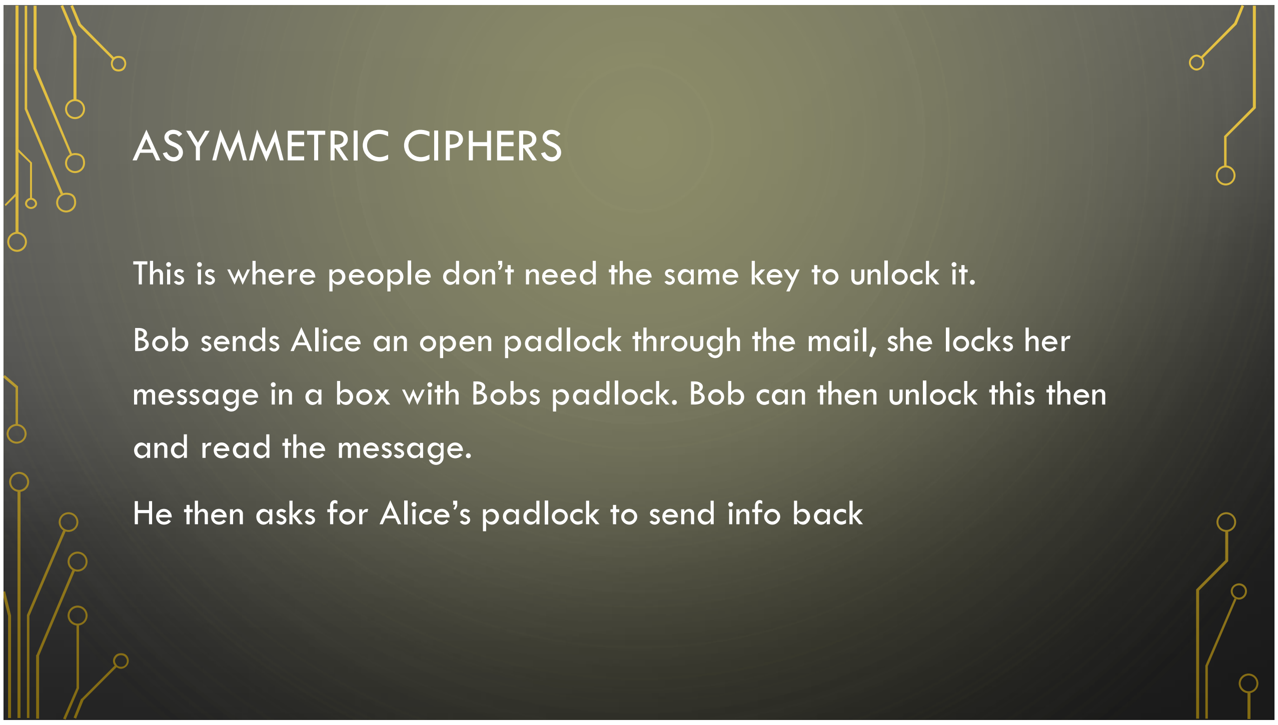
SYMMETRIC CIPHERS

To unlock the data you both need the same (or symmetric) keys.

So if Alice wants to send something to Bob in the post, she takes it and puts it in a box and locks it with a padlock that she has the key to.

It arrives at Bobs door and he has a copy of the key (which he probably had to get from her) and so can unlock the box and send stuff back using the box

But what if we can't meet up in person??

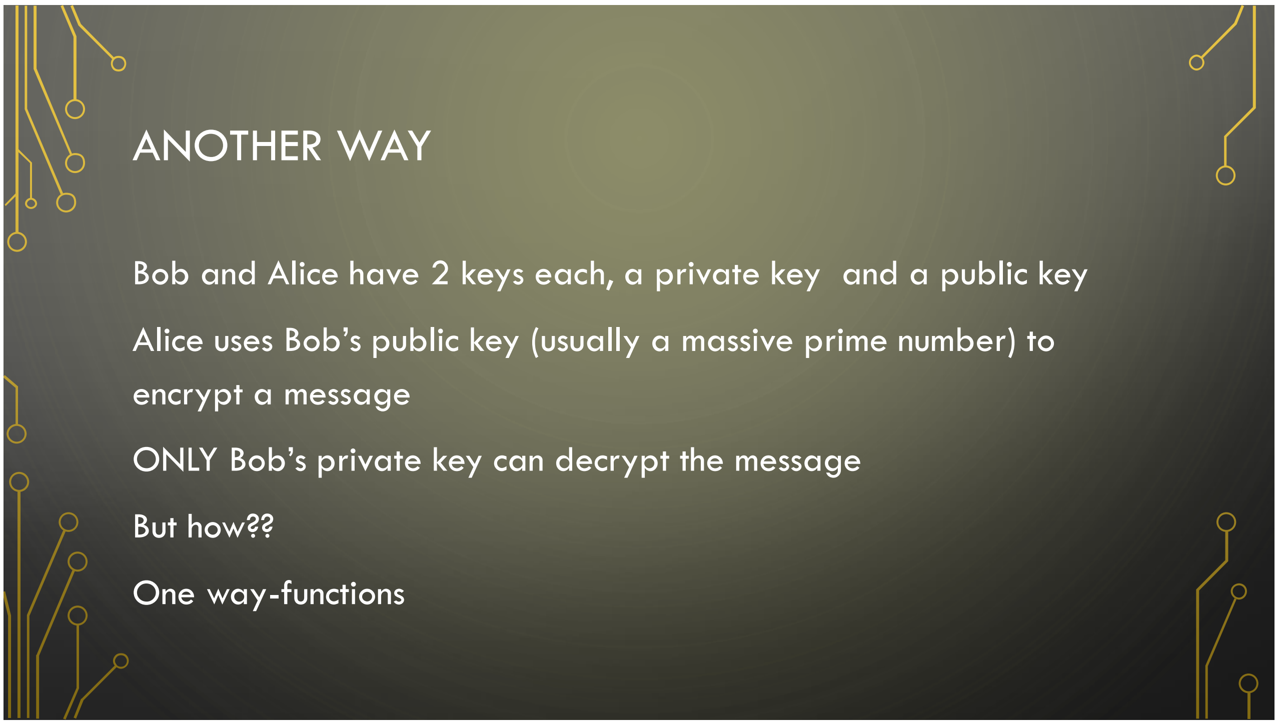


ASYMMETRIC CIPHERS

This is where people don't need the same key to unlock it.

Bob sends Alice an open padlock through the mail, she locks her message in a box with Bobs padlock. Bob can then unlock this then and read the message.

He then asks for Alice's padlock to send info back



ANOTHER WAY

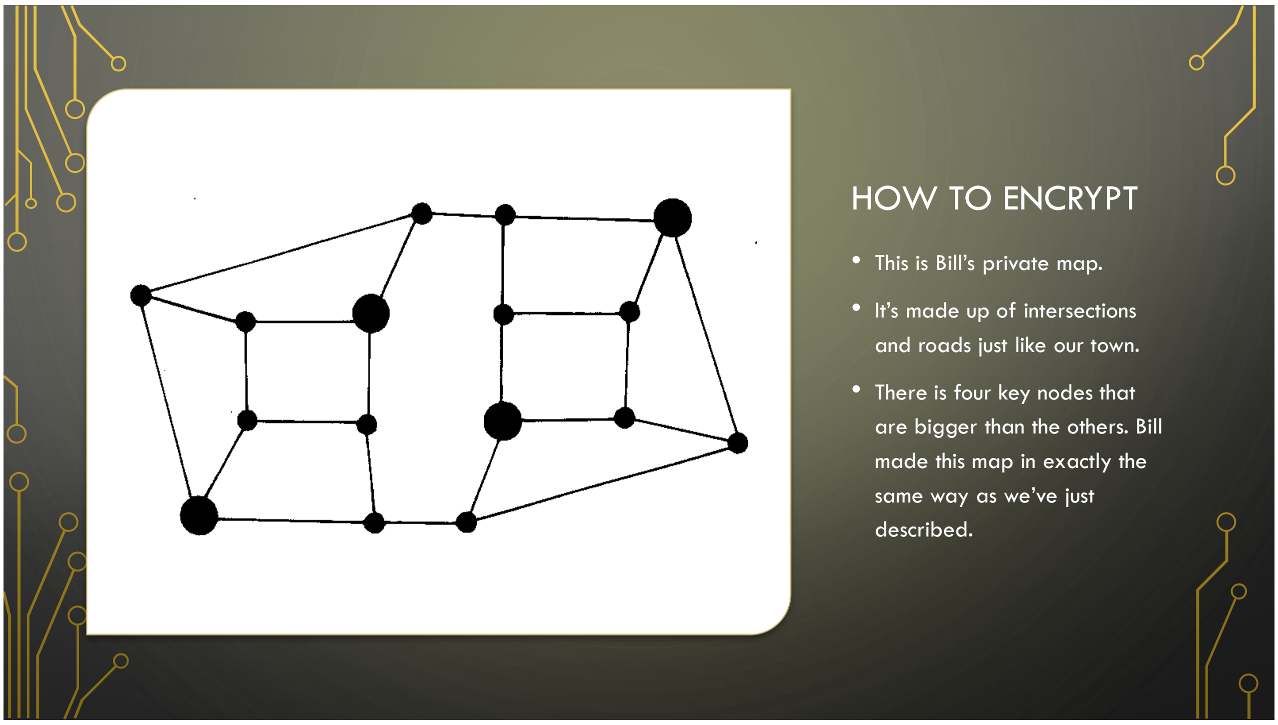
Bob and Alice have 2 keys each, a private key and a public key

Alice uses Bob's public key (usually a massive prime number) to encrypt a message

ONLY Bob's private key can decrypt the message

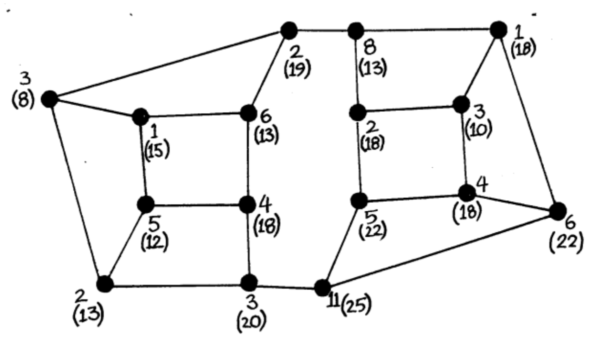
But how??

One way-functions



HOW TO ENCRYPT

- This is Bill's private map.
- It's made up of intersections and roads just like our town.
- There is four key nodes that are bigger than the others. Bill made this map in exactly the same way as we've just described.



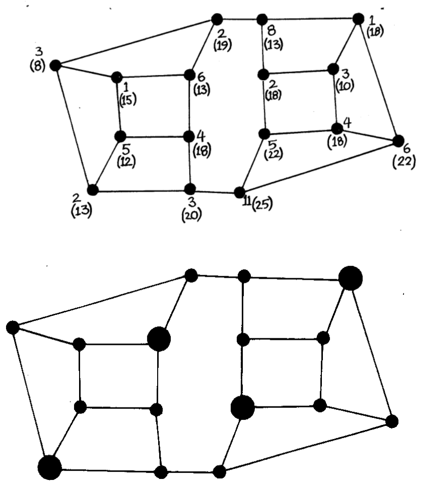
HOW TO ENCRYPT

- Choose the number you want to send (in this example 66)
- Write a number at each node (intersection) so that they total 66.
- Add all the numbers that are linked to each node, including itself, write this number in brackets.

e.g. bottom right

6 + 11 + 4 + 1 = 22

- Rub out all of the non-bracketed numbers. You're done!



HOW TO DECRYPT

- Bill take's his private map and takes all the numbers that are at the larger nodes.
- So 18 + 22 + 13 + 13 = 66
- Now give it a go!

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Data & Binary Lesson Plan

Strand: Computer Science Concepts *Duration of Lesson: 45 mins – 1 hour 15 mins*

Learning Outcomes

Students will understand what data is and how it differs from information.
Students will understand why storing and being able to access data is good and beneficial.
Students will begin to understand how computers store data.
Students will understand that labelling data through units is important.
Students will be able to convert decimal numbers to binary and back.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

2.13 describe the rationale for using the binary number system in digital computing and how to convert between binary, hexadecimal and decimal

3.7 use algorithms to analyse and interpret data in a way that informs decision-making

Junior Certificate Coding Short Course

2.5 explain how computers represent data using 1's and 0's

Computational Thinking Skills used

Algorithm
Pattern matching
Abstraction
Decomposition

Computer Science concepts/topics involved

Binary numbers & conversion to/from decimal
Data

Resources / Materials

Pens, paper etc.
Bakuro problems 1 & 2

The Class

Discussion – Introduction – 10 mins

Data is all around us in this technology era and can be easily accessed through the internet from almost anywhere in the world on devices ranging from supercomputers to watches. How do computers store so much information and how is it that we can access it? This lesson will show students how computers store this data.

To begin with get students to suggest definitions for what data is. Do they think there's any difference between data and information?

Data is a collection of facts, such as numbers, words, measurements, observations or even just descriptions of things. The key difference between data and information is that data is unorganized, data "becomes" information when it is organised, structured or presented in each context to make it useful, it is called information. For example, a student's test score is one piece of data but the average of the class/school etc. is information that can be derived from the data.

Get them to think of a few examples of data and information.

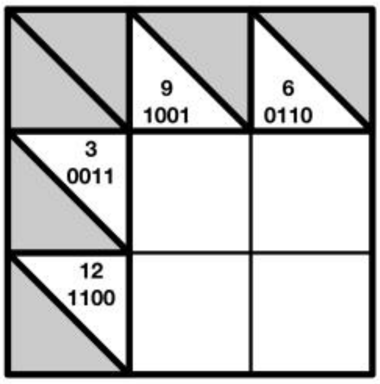
Activity – Introduction – 10-20 mins

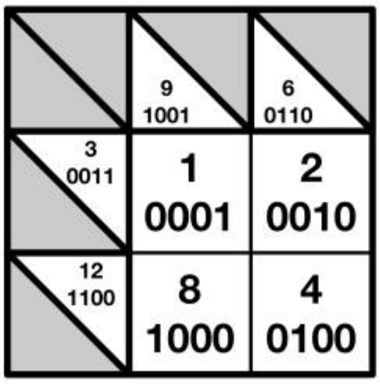
Computer processors are made up of a huge number of switches that can either be on or off. It is in this way that computers store, read and write data. This is done using binary. We live most of our lives using the decimal number system but computers don't work that way.

However, we do use others, ask students if they can think of any other number systems we use regularly in day-to-day life?

12 – months
7 – days
24 – hours
60 – minutes

Binary is the number system of base 2, so all numbers are represented using only 1's and 0's. In this way computers can use binary to do calculations and store data using the switches, 1 = on, 0 = off. During this lesson, we'll have a look more closely at binary numbers.
To introduce this, we'll have a look at Bakuro problems. Bakuro are binary versions of the popular Kakuro puzzle. The empty cells of the grid must be filled with the numbers 1, 2, 4 and 8 (i.e., only powers of 2). The numbers in each block in a column or row must add up to the number given in the clue above or to the left, respectively. No number can be used twice within any sum. The clues are given in both binary and decimal. The answers must also be written in both binary and decimal. Below is an example.





There is two Bakuro problems which you can download separately. They explain the rules and how to complete them, they also have the above examples. *(credit Queen Mary University of London, Teaching Computing London)*

Activity – Development – 10-20 minutes

Now we've introduced binary numbers, we'll have a look at binary to decimal conversion. To convert a binary number, say 1010, to decimal you do the following. Starting with the least-significant digit (the one on the far right), multiply it by 2^0. The next least-significant digit you multiply by 2^1, and so on, raising the power by one each time until you have all the digits accounted for. You then add the resulting numbers up, below is two worked examples:

$(1010)_2 = 0x2^0 + 1x2^1 + 0x2^2 + 1x2^3$
$= 0 + 2 + 0 + 8$
$= (10)_{10}$
$(1100111)_2 = 1x2^0 + 1x2^1 + 1x2^2 + 0x2^3 + 0x2^4 + 1x2^5 + 1x2^6$
$= 1 + 2 + 4 + 0 + 0 + 32 + 64$
$= (103)_{10}$

Having done these examples, make sure you explain the meaning and importance of the subscripts. You can think of the subscript as like giving the unit in maths (kg, g, m, km etc.). If we don't put $_2$ or $_{10}$ after numbers, then it would be impossible to know what number system we're using. This can lead to massive problems, for example if you agreed (for some reason!) to buy a car from a dealer who used binary for pricings, then offering 1000 would be quite an insult (as it would be €8)!

Or take NASA. Full of incredibly smart and capable people in all fields but especially engineering and mathematics. Not once, but twice they've lost hugely expensive ($100 mill plus!) equipment due to miscalculations. One was a Mars orbiter as one team use metric units for calculations whilst the other didn't. Another near-disaster was in 1983 when an Air Canada plane ran out of fuel mid-flight when there first metric measurement plane was flying. Luckily only two people received minor injuries. Units matter!

Give the students the following binary numbers and get them to convert them into decimal.

$(1101)_2 = (13)_{10}$

$(10010)_2 = (34)_{10}$

$(111100011)_2 = (483)_{10}$

Activity – Development – 10-20 minutes

Next, we'll look at how to convert decimal numbers to binary. To convert a decimal number such as 35 to binary you do the following:

Take 35 and divide it by 2, but instead of getting 17.5 we're going to use remainders. So we get 17 remainder 1. The important part is the remainder so write 1 out and repeat this process. Then read the remainders from the last to the first. Two examples are done below:

35/2 = 17 remainder 1
17/2 = 8 remainder 1
8/2 = 4 remainder 0
4/2 = 2 remainder 0
2/2 = 1 remainder 0
1/2 = 0 remainder 1
$(35)_{10} = (100011)_2$

20/2 = 10 remainder 0
10/2 = 5 remainder 0
5/2 = 2 remainder 1
2/2 = 1 remainder 0
1/2 = 0 remainder 1
$(20)_{10} = (10100)_2$

Give the students the below numbers and get them to convert the decimal numbers to binary.

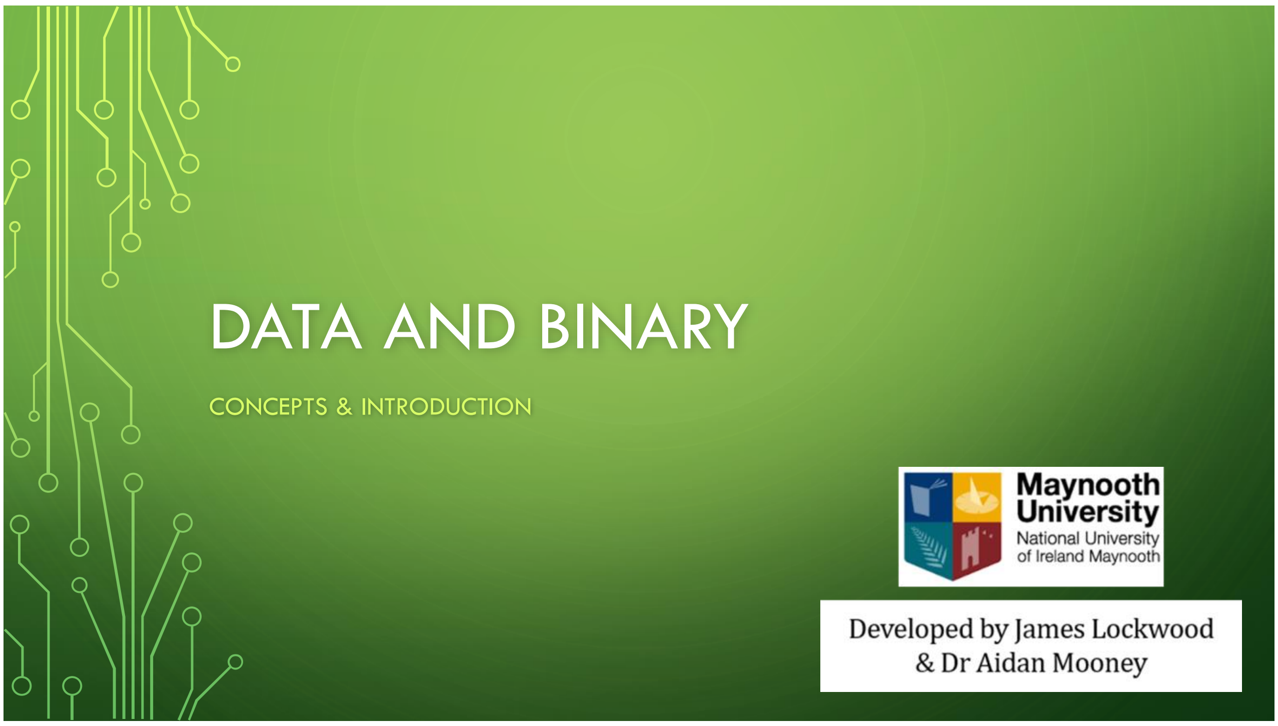
$(7)_{10} = (111)_2$

$(23)_{10} = (10111)_2$

$(49)_{10} = (110001)_2$

Feedback – Conclusion – 5 minutes

Please get your class to fill out the Data & Binary feedback forms.

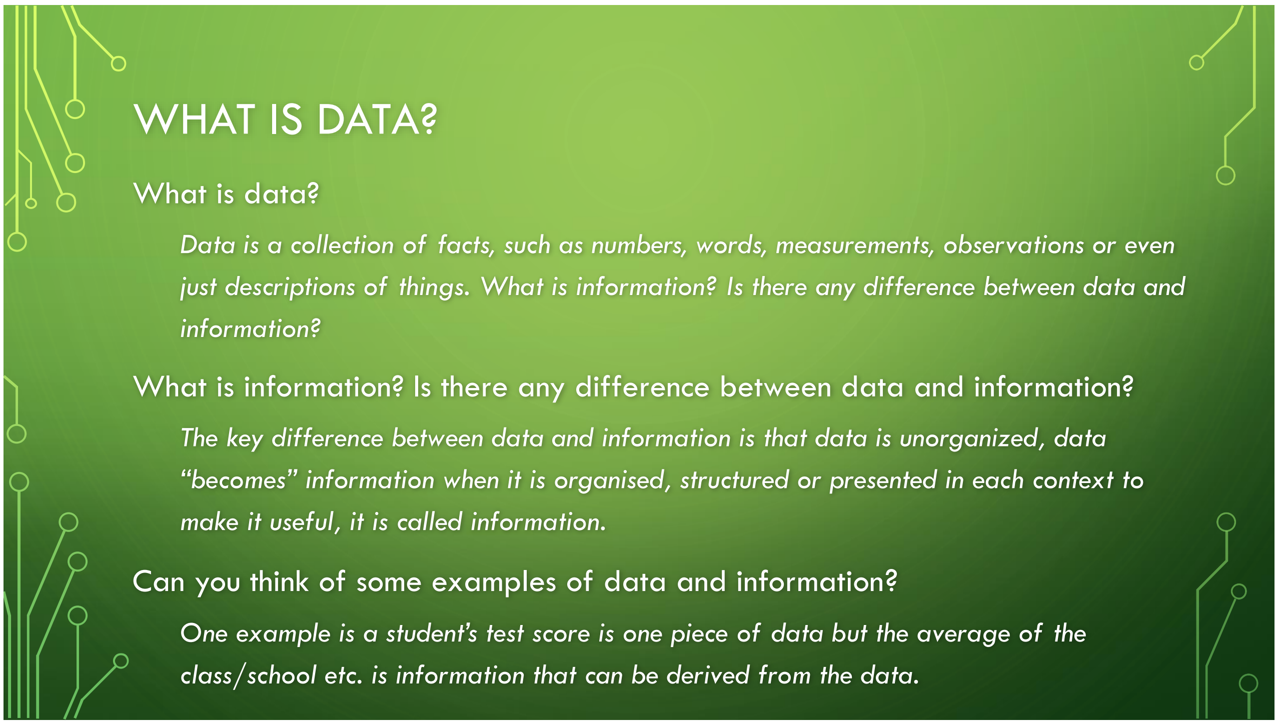


DATA AND BINARY

CONCEPTS & INTRODUCTION

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney



WHAT IS DATA?

What is data?

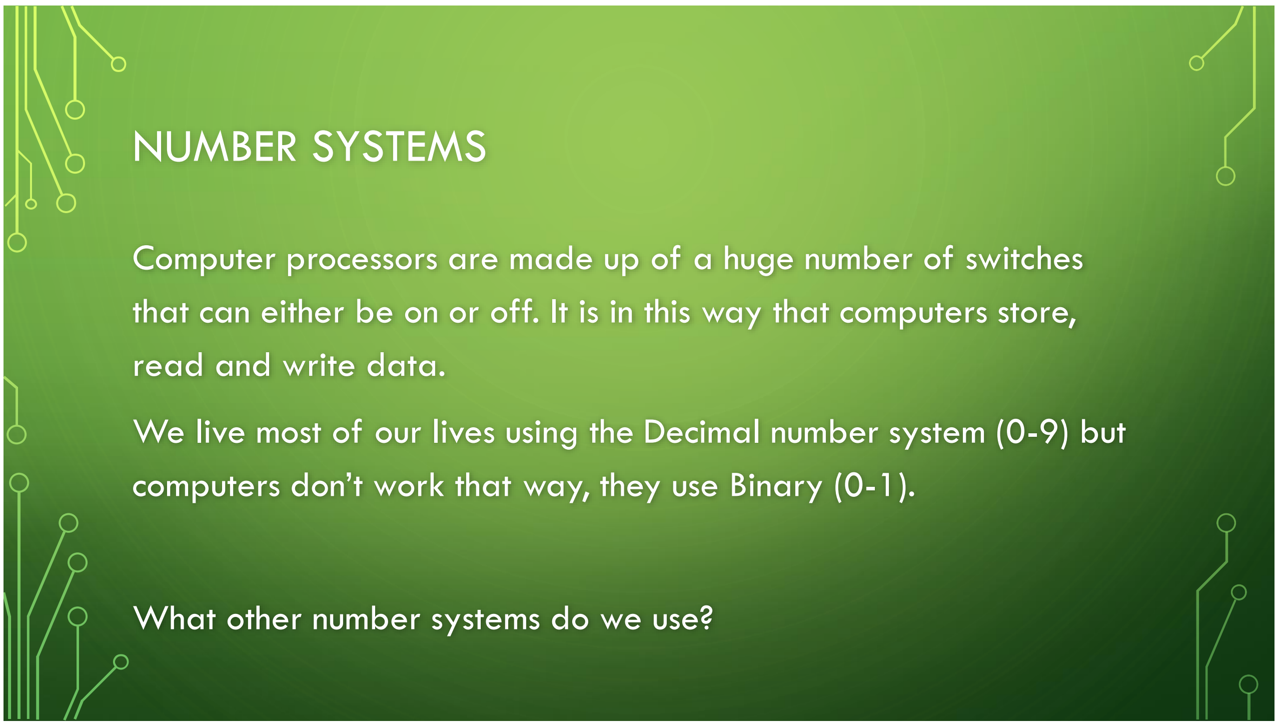
Data is a collection of facts, such as numbers, words, measurements, observations or even just descriptions of things. What is information? Is there any difference between data and information?

What is information? Is there any difference between data and information?

The key difference between data and information is that data is unorganized, data "becomes" information when it is organised, structured or presented in each context to make it useful, it is called information.

Can you think of some examples of data and information?

One example is a student's test score is one piece of data but the average of the class/school etc. is information that can be derived from the data.

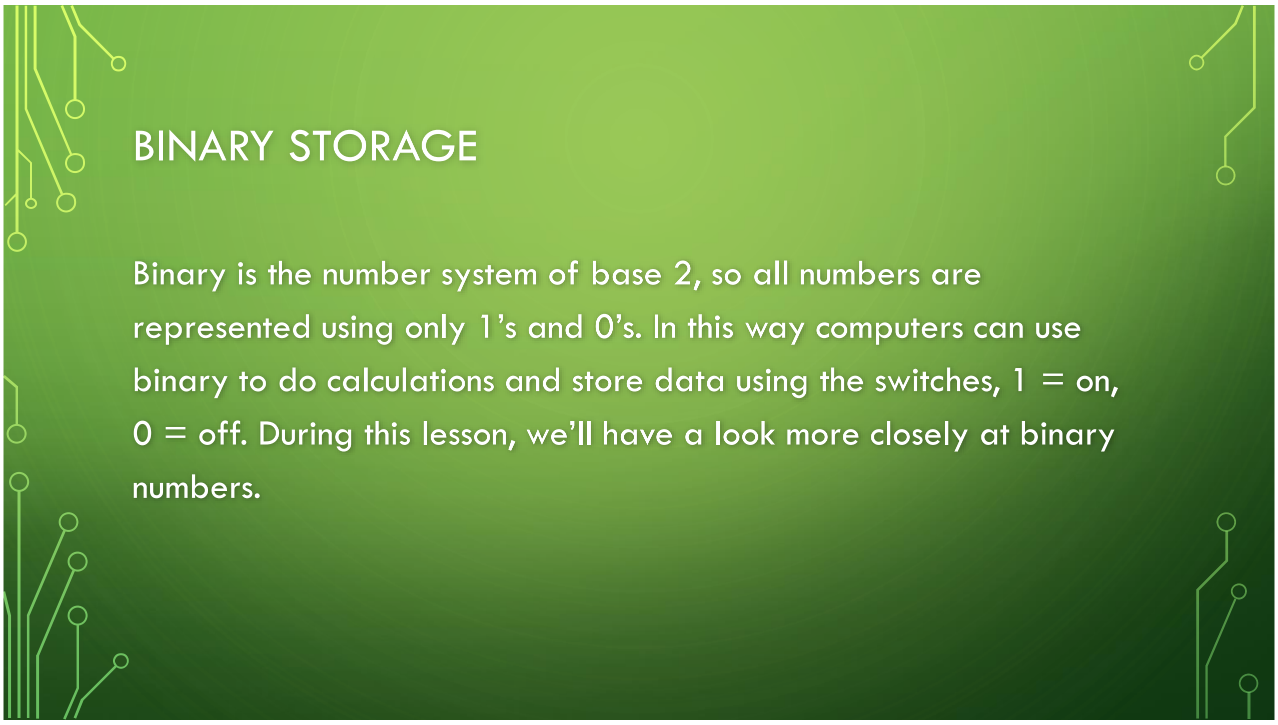


NUMBER SYSTEMS

Computer processors are made up of a huge number of switches that can either be on or off. It is in this way that computers store, read and write data.

We live most of our lives using the Decimal number system (0-9) but computers don't work that way, they use Binary (0-1).

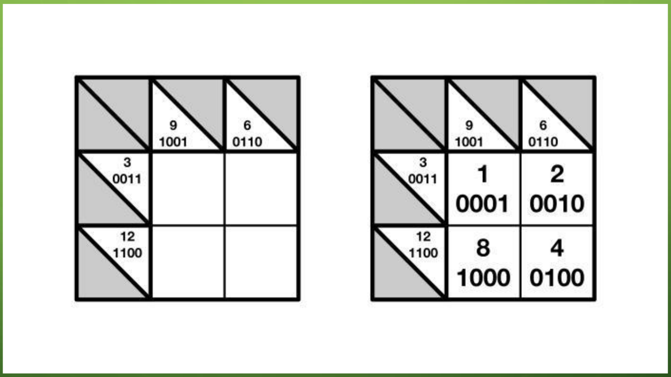
What other number systems do we use?



BINARY STORAGE

Binary is the number system of base 2, so all numbers are represented using only 1's and 0's. In this way computers can use binary to do calculations and store data using the switches, 1 = on, 0 = off. During this lesson, we'll have a look more closely at binary numbers.

BAKURO



BINARY NUMBERS – BINARY TO DECIMAL CONVERSION

$(1010)_2 = 0x2^0 + 1x2^1 + 0x2^2 + 1x2^3$

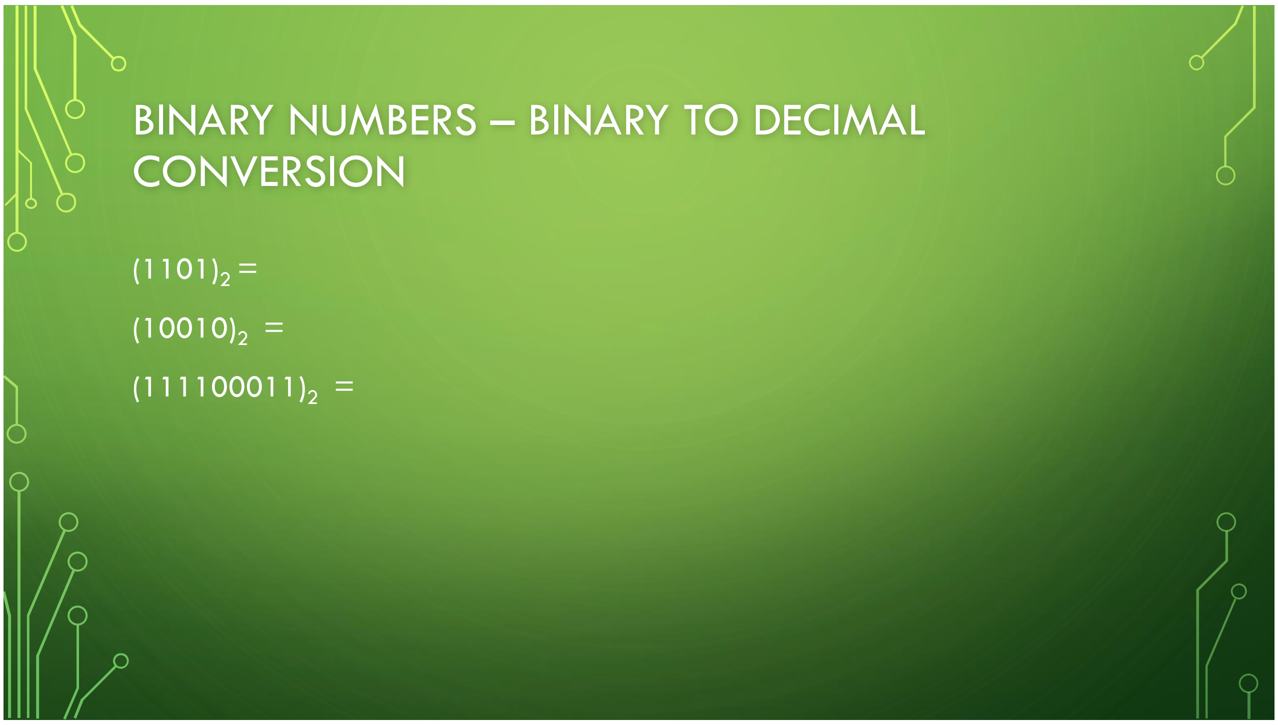
$= 0 + 2 + 0 + 8$

$= (10)_{10}$

$(1100111)_2 = 1x2^0 + 1x2^1 + 1x2^2 + 0x2^3 + 0x2^4 + 1x2^5 + 1x2^6$

$= 1 + 2 + 4 + 0 + 0 + 32 + 64$

$= (103)_{10}$

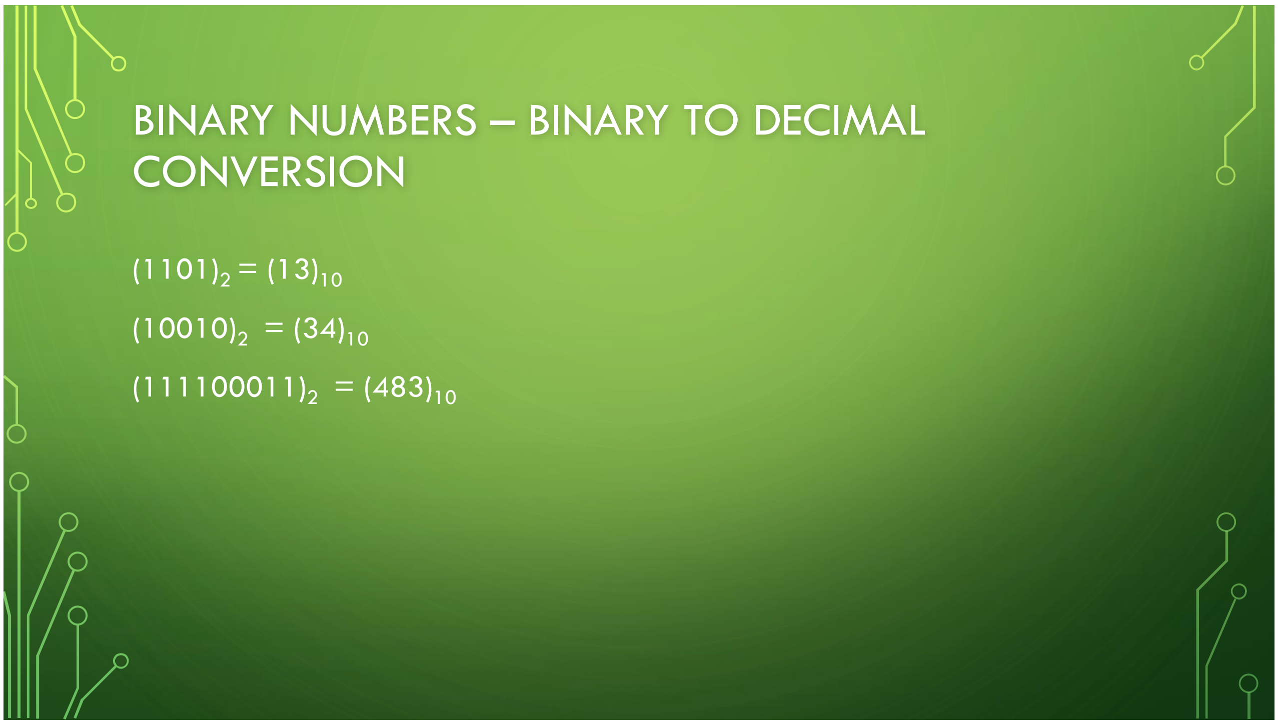


BINARY NUMBERS – BINARY TO DECIMAL CONVERSION

$(1101)_2 =$

$(10010)_2 =$

$(111100011)_2 =$

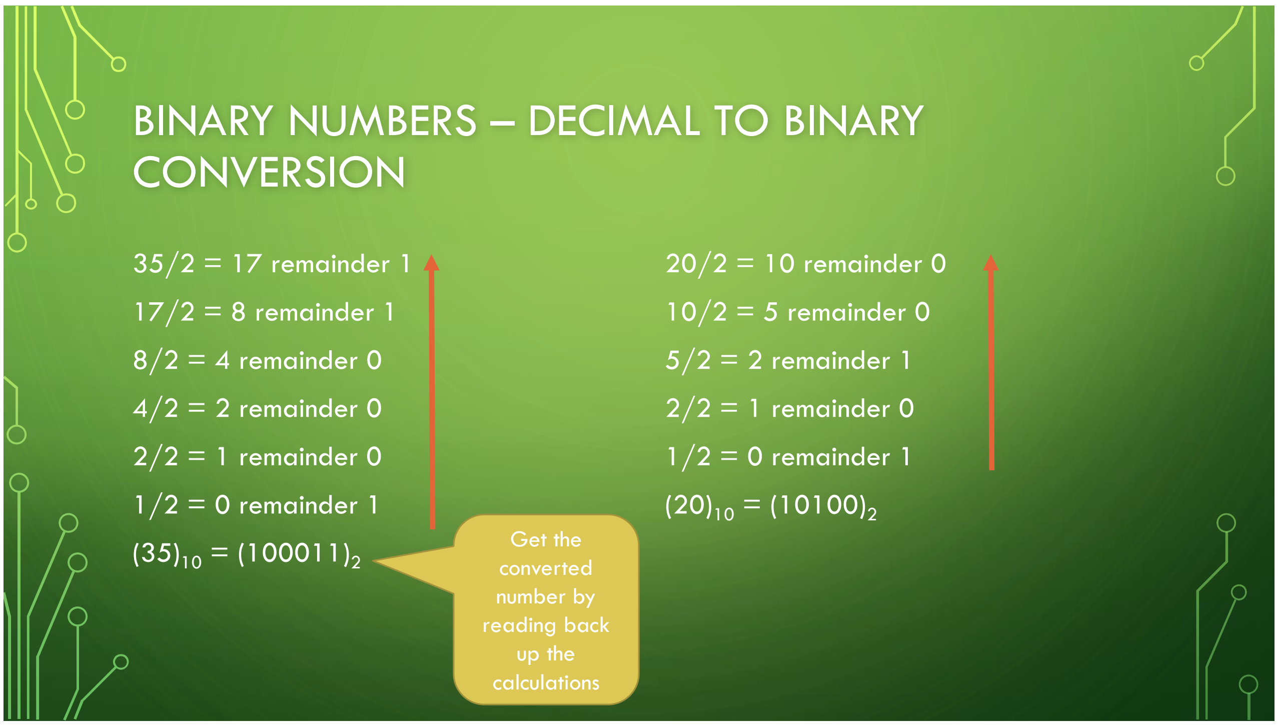


BINARY NUMBERS – BINARY TO DECIMAL CONVERSION

$(1101)_2 = (13)_{10}$

$(10010)_2 = (34)_{10}$

$(111100011)_2 = (483)_{10}$

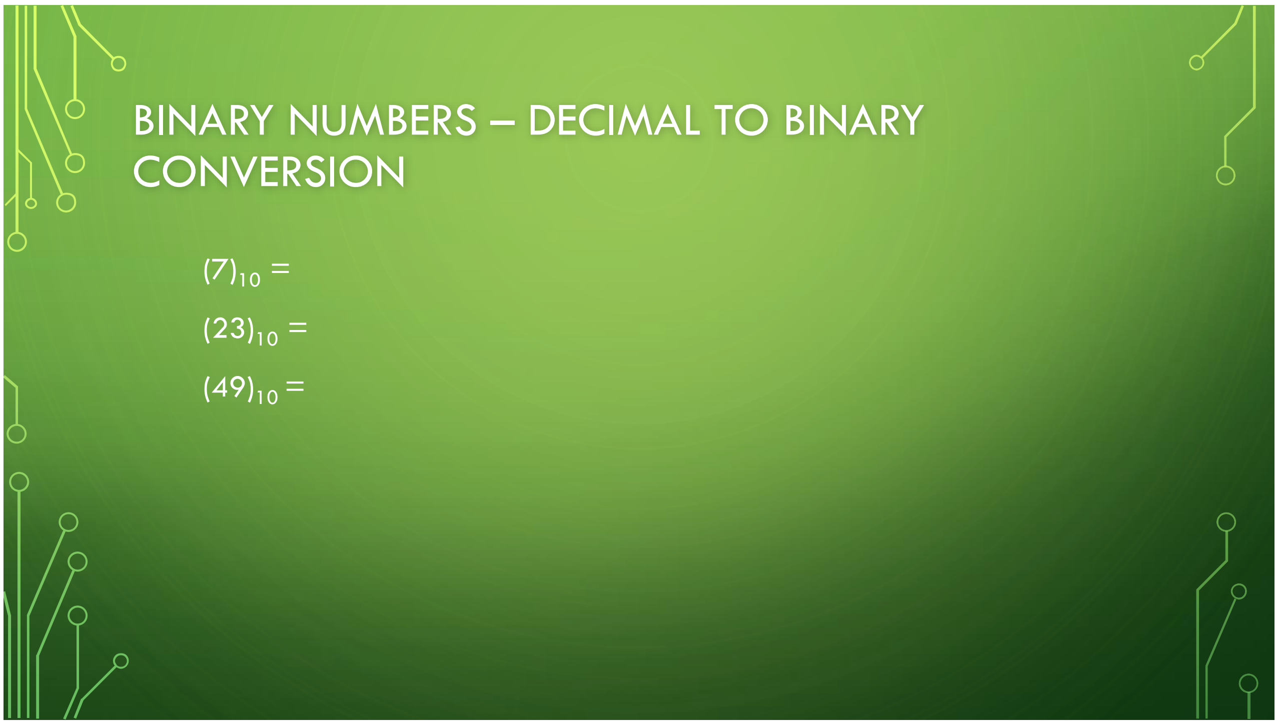


BINARY NUMBERS – DECIMAL TO BINARY CONVERSION

35/2 = 17 remainder 1
17/2 = 8 remainder 1
8/2 = 4 remainder 0
4/2 = 2 remainder 0
2/2 = 1 remainder 0
1/2 = 0 remainder 1
$(35)_{10} = (100011)_2$

Get the converted number by reading back up the calculations

20/2 = 10 remainder 0
10/2 = 5 remainder 0
5/2 = 2 remainder 1
2/2 = 1 remainder 0
1/2 = 0 remainder 1
$(20)_{10} = (10100)_2$

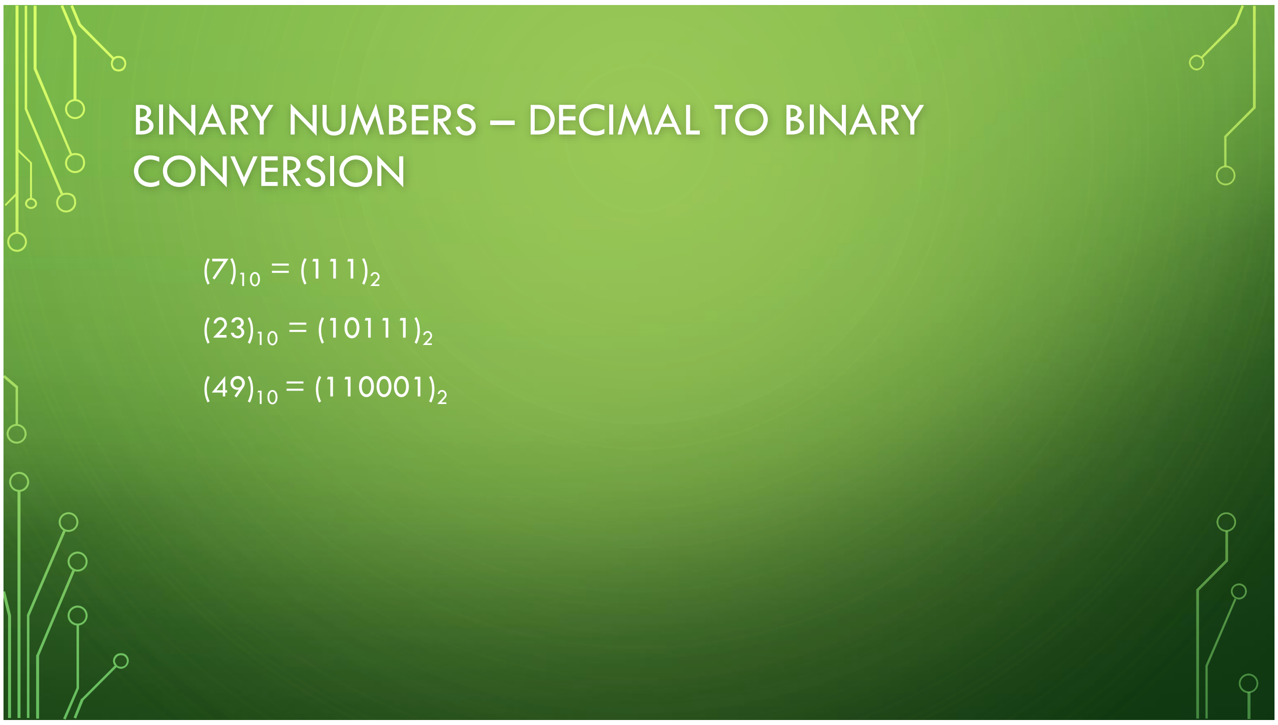


BINARY NUMBERS – DECIMAL TO BINARY CONVERSION

$(7)_{10} =$

$(23)_{10} =$

$(49)_{10} =$



BINARY NUMBERS – DECIMAL TO BINARY CONVERSION

$(7)_{10} = (111)_2$

$(23)_{10} = (10111)_2$

$(49)_{10} = (110001)_2$

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Finite State Machines Lesson Plan

Strand: Computer Science Concepts *Duration of Lesson: 1 hour 20 minutes*

Learning Outcomes

Students will learn what Finite State Machines are and how they can be used.
Students will learn what Regular Expressions are and how they can be used.
Students will learn natural language definitions (e.g. Alphabet, Strings etc.)
Students will be able to generate strings from given regular expressions as well as generate regular expressions for given strings.
Students will be able to design FSM's for given languages as well as generate languages from given FSM's.

Curriculum links

Leaving Certificate Computer Science Specification

1.2 explain how the power of computing enables different solutions to difficult problems

2.2 use a range of methods for identifying patterns and abstract common features

Computational Thinking Skills used

Algorithm

Computer Science concepts/topics involved

Finite State Machines
Regular Expressions
Pattern matching
Natural language theory

Resources / Materials

CS Unplugged Activity – Treasure Island

The Class

Activity – Introduction and Development - 20 minutes

The CS Unplugged activity, Treasure Island, is a fantastic introduction to Finite State Machines (FSM). The idea is that students must move from island to island using two ships, A and B. This mimics a FSM as the ships are possible inputs and the students must find the treasure which is at the final state. The worksheet provides all the necessary information and materials. I recommend doing the class activity and the worksheet activity Treasure Island as well (page 12). If required/wanted there is a second worksheet activity as well.

Discussion – Development – 10 minutes

After explaining that the maps can relate to FSM, we'll look briefly at FSM's in more detail.
Before this we need to look at something called regular expressions. Regular expressions are a special type of notation for specifying an input or output. They work mainly with what are known as Formal languages, the reason they're called Formal is because we're only worried about the form (or syntax) of the language, not what the symbols actually mean (the semantics).
There are a number of things we need to define before we get started. See if you students can come up with any other examples of the below definitions as you go through them.

Alphabet: A finite set of symbols. These are the fundamental units from which we build structures.

Examples of alphabets include:

- Our treasure maps alphabet was {A, B} as we only had two ships called A and B.
- All whole numbers can be made up of the alphabet {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- all binary numbers of the alphabet {0, 1}
- All the letters in the English alphabet {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}

String: A string over an alphabet is a finite sequence of symbols from the alphabet.

Examples of strings from the above alphabets:

- Whole Numbers: 109, 45678, 301, ...
- Binary numbers: 0, 1, 10, 11, 100, …, 11100010101010, …
- English: Any word in the dictionary

Language: Is a set of strings.

Example of languages from the previous examples:

- All even numbers: {2, 4, 6, 8, …, 222, …., 1034560, …}
- All odd numbers: {1, 11, 101, …, 110101, …, 1010101010101, …}
- Any word that begins with an a: {aaron, after, apple, august,…}

Word: Is a string in a language.

Any of the strings in the above languages would be a word. For example: 22, 101, after etc.

Activity – Development – 10-20 minutes

Hopefully now your students get the general pattern of how Formal Languages are defined. Get them to go through Exercise Sheet 1 either individually or in pairs.

Activity – Development – 20 minutes

FSM's are usually designed to accept certain types of inputs, for example names, email addresses, flight codes, even numbers. These are defined using a language, which is basically just a set of "words". All inputs that are accepted can usually be defined using Regular Expressions. Regular expressions have a few symbols that make generalising expressions easier, these are listed below:

a = means one a

. = means that is matches one character

* = means zero or more of the preceding character

[a-z], [0-9] = means any element in the given range

Some examples of regular expressions:

aaa = this produces the string "aaa"

ababa = this produces the string "ababa"

a* = this produces the strings {empty, a, aa, aaa, aaa, …, aaaaaaaaaaaaaa, …}

ba* = this produces the strings {b, ba, baa, baaa, …, baaaaaaaaa, …}

(ba)* = this produces the strings {empty, ba, baba, bababa, …, babababababababa, …}

a|b = this produces the string a or b

(a|b)* = this produces any string containing 0 or more of a or b in any order e.g. {empty, a, b, aa, bb, abababa, bababa, bbabaababa, …}

[a-z] = this produces the strings {a, b, c, d, …, z}

[a-z]* = this produces any word in the English language and any other combination of lower case letters for example: {cat, monkey, fundraising, huuaa, laouaoi, buytali, …}

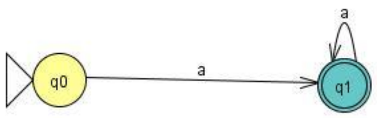
Get students to fill out Exercise Sheet 2 individually or in pairs.

Activity – Development – 20 minutes

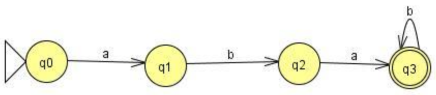
This task is to design FSM's that match the below regular expressions. Below are also some examples, make sure to remind students they've already seen FSM's in the treasure island activity!
FSM's are made up of states (our islands) which are drawn with circles and transitions (arrows). There is always a start state and an end state (where the treasure is). The start state has an arrow pointing into it

and the end state has two circles. A FSM is said to *accept* a string if it ends up at the final state at the end of the string.

Design a FSM that accepts all strings of the form a*



Design a FSM that accepts all strings of the form abab*



Get students to complete Exercise Sheet 3 either individually or in pairs.

Feedback – Conclusion – 5 minutes

Please get your students to fill out the Finite Automata feedback forms.

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

FSMs Exercise Sheet 1

Computational Thinking Course

Q1. Given the following languages write out 4 words that are in the language, an example for each language has been give to you.

Alphabet = {a, b, c, d, …, z}

L1 = {w | w is a word beginning with the letters "be"}

1. **bed**________________________
2. ____________________________
3. ____________________________
4. ____________________________
5. ____________________________

L2 = {w | w contains *only* a single vowel}

1. **thumb**_____________________
2. ____________________________
3. ____________________________
4. ____________________________
5. ____________________________

L3 = {w | words ending in "ing"}

1. **running**
2. ____________________
3. ____________________
4. ____________________
5. ____________________

Alphabet = {0, 1, 2, 3,…, 9}

L4 = {w | w is an odd number > 40 and < 100}

1. **41**
2. ____________________
3. ____________________
4. ____________________
5. ____________________

L5 = {w | w is a prime number}

1. **3**
2. ____________________
3. ____________________
4. ____________________
5. ____________________

L6 = {w | w is a multiple of 2, 3, 4 and 6}

1. __144__________________________
2. ______________________________
3. ______________________________
4. ______________________________
5. ______________________________

Q2. Given the following words, construct a language they belong to. Hint: Some of them may belong to multiple valid languages (see example below).

Example:
Words: hello, apple, hoops
Possible Languages:

- LEx1: {w | w is a word of length 5}
- LEx2: {w | w is a word that contains at least one vowel}
- LEx3: {w | w is a word that contains exactly two vowels}

Alphabet: {a, b, c, d, …, z}

Words: Tom, Sarah, Phillip, Mary

L7: ____

Words: banana, battery, banjo, bandana, baseball

L8: ____

Words: *abbreviate, facilities, skirmisher, yourselves, headstrong*

L9: ____

Alphabet: {0, 1, 2, …, 9}

Words: 2, 86, 246, 900

L10: ____

Words: 2, 3, 5, 7

L11: ____

Words: *10, 100, 1000, 10000, 100000, 1000000*

L12: ____

Developed by James Lockwood
& Dr Aidan Mooney

FSMs Exercise Sheet 1 Teachers Guide

Computational Thinking Course

Strand: Finite State Machines *Duration of Sheet: 10-20 minutes*

Q1. Given the following languages write out 4 words that are in the language, an example for each language has been give to you.

Alphabet = {a, b, c, d, …, z}

L1 = {w | w is a word beginning with the letters "be"}

1. **bed**
2. **because**
3. **bee**
4. **better**
5. **been**

L2 = {w | w contains *only* a single vowel}

1. **thumb**
2. **hot**
3. **cold**
4. **and**
5. **ask**

L3 = {w | words ending in "ing"}

1. **running**
2. **jumping**
3. **flying**
4. **swimming**
5. **falling**

Alphabet = {0, 1, 2, 3,…, 9}

L4 = {w | w is an odd number > 40 and < 100}

1. **41**
2. **57**
3. **89**
4. **63**
5. **99**

L5 = {w | w is a prime number}

1. **3**
2. **5**
3. **7**
4. **23**
5. **197**

L6 = {w | w is a multiple of 2, 3, 4 and 6}

1. **144**
2. **12**
3. **24**
4. **36**
5. **48**

Q2. Given the following words, construct a language they belong to. Hint: Some of them may belong to multiple valid languages (see example below).

Example:

Words: hello, apple, hoops

Possible Languages:

- LEx1: {w | w is a word of length 5}
- LEx2: {w | w is a word that contains at least one vowel}
- LEx3: {w | w is a word that contains exactly two vowels}

Alphabet: {a, b, c, d, …, z}

Words: Tom, Sarah, Phillip, Mary

L7: {w | w is an English langauge first name}

Words: banana, battery, banjo, bandana, baseball

L8: {w | w is a word that begins with the letters "ba"}

Words: *abbreviate, facilities, skirmisher, yourselves, headstrong*

L9: {w | w is a word of length 10}

Alphabet: {0, 1, 2, …, 9}

Words: 2, 86, 246, 900

L10: _{w | w is an even number}______________________________________

Words: 2, 3, 5, 7

L11: __{w | w is a prime number under 10}_______________________________

Words: *10, 100, 1000, 10000, 100000, 1000000*

L12: __{ w | w is a power of ten up to the power of 6}_________________________

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

FSMs Exercise Sheet 2

Computational Thinking Course

Q1. Write out 3 words that the following Regular Expressions produce. An example is given for some of them, remember we aren't worried about whether they are valid words/numbers/phrases etc.

Alphabet: {0, 1}

Reg Exp 1: 0110*

1. ______________________________
2. ______________________________
3. ______________________________

Reg Exp 2: 10.*

1. 101
2. ______________________________
3. ______________________________
4. ______________________________

Reg Exp 3: .*00.*

1. 0000000000000
2. ______________________________
3. ______________________________
4. ______________________________

Alphabet: {a, b, c,…, z}

Reg Exp 4: ab*

1. ____________________________
2. ____________________________
3. ____________________________

Reg Exp 5: (a|b)*b

1. abbbbaaab
2. ____________________________
3. ____________________________
4. ____________________________

Reg Exp 6: [a-z]*ing

1. aing
2. ____________________________
3. ____________________________
4. ____________________________

Q2. Given the following words, construct a regular expression that fits all the given words (don't worry about matching other words that aren't included). There may be multiple correct answers for some of them., the trick is to see what is the same in each word.

Alphabet: {0,1}

Example:

Words: {1, 01, 1010101001, 101, 0000001} *all words end in a one*

Regular exp: _{0 | 1}*1__________________

Words: {0, 00, 000, 0000000000, 0000000000000000}

Reg Exp 1: ___________________________________

Words: {101001101, 101, 100000011111101, 10101, 10001}

Reg Exp 2: ____________________________________

Alphabet: {a, b, c, …, z}

Words: {abbbz, acgfhez, alopz, aaaazzzzz}

Reg Exp 3: ____________________________________

Words: {upupupupupupu, pu, up, pup, upppupupu}

Reg Exp 4: ____________________________________

Words: {abababababababzjauhuwbal, abitlksip, ab, abpoplpin, abc, abcd}

Reg Exp 5: ____________________________________

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

FSMs Exercise Sheet 2 Teachers Guide

Computational Thinking Course

Strand: Finite State Machines *Duration of Sheet: 20 minutes*

Q1. Write out 3 words that the following Regular Expressions produce. An example is given for some of them, remember we aren't worried about whether they are valid words/numbers/phrases etc.

Alphabet: {0, 1}

Reg Exp 1: 0110*

1. 0110
2. 011
3. 01100000000

Reg Exp 2: 10.*

1. 101
2. 10111
3. 100
4. 101010101010101010

Reg Exp 3: .*00.*

1. 0000000000000
2. 1001

3. 0000

4. 111110000000000

Alphabet: {a, b, c,…, z}

Reg Exp 4: ab*

1. abbbbbb
2. ab
3. a

Reg Exp 5: (a|b)*b

1. abbbbaaab
2. b
3. aaaaaaaab
4. bbbbbbbbbb

Reg Exp 6: [a-z]*ing

1. aing
2. running
3. zzzijaing
4. swimming

Q2. Given the following words, construct a regular expression that fits all the given words (don't worry about matching other words that aren't included). There may be multiple correct answers for some of them, the trick is to see what is the same in each word.

Alphabet: {0,1}

Example:

Words: {1, 01, 1010101001, 101, 0000001} *all words end in a one*

Regular exp: _(0 | 1)*1_________________

Words: {empty, 0, 00, 000, 0000000000, 0000000000000000} *any amount of 0s*

Reg Exp 1: _0*________________________

Words: {101001101, 101, 100000011111101, 10101, 10001} *all words start and end in a 1*

Reg Exp 2: _1 (0|1)* 1__________________

Alphabet: {a, b, c, …, z}

Words: {abbbz, acgfhez, alopz, aaaazzzzz, az} *all words start with an a and end with a z*

Reg Exp 3: _a [a-z]* z___________________

Words: {upupupupupupu, pu, up, pup, upppupupu}

Reg Exp 4: __(u|p)*_____________________

Words: {ababababababababzjauhuwbal, abitlksip, ab, abpoplpin, abc, abcd}

Reg Exp 5: ___(ab)*[a-z]*_______________

Maynooth University
National University of Ireland Maynooth

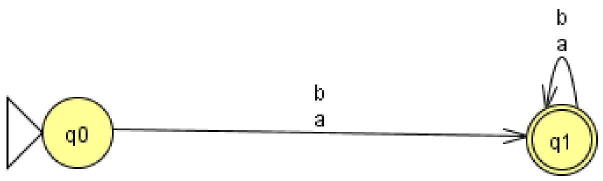
Developed by James Lockwood
& Dr Aidan Mooney

FSM Exercise Sheet 3

Computational Thinking Course

Q1. Are the given strings/regular expressions accepted by their corresponding Finite State Machine? Remember for a string to be accepted it must end on the accept state of the machine. Also see if you can figure out the "rule" i.e. what type of strings does the machine accept?

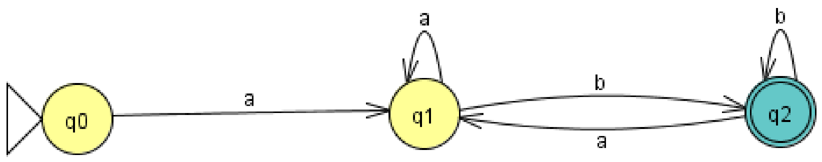
Machine 1:



String	Accept/Reject
a	
b	
c	
abab	
bbbbbbbb	
abaffbab	

The rule:

Machine 2:

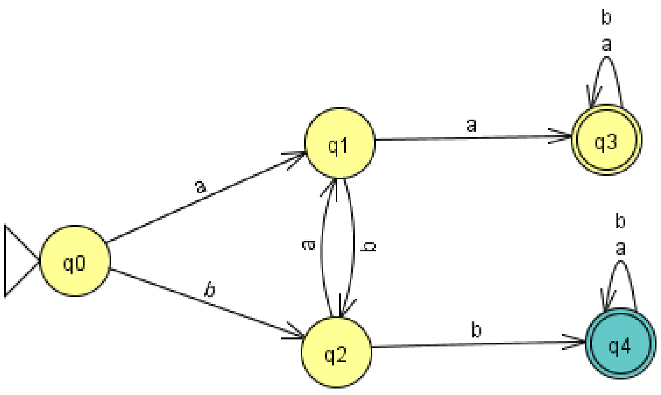


String	Accept/Reject
a	
b	

abc	
abab	
bbbbbbbb	
abababbbbb	
abababab a	

The rule:

Machine 3:



String	Accept/Reject
a	
b	
abc	
abab	
bbbbbbbb	
abababbbbb	
ababababa	
abababaababab	

The rule:

Q2. Design a FSM that accepts the following strings/regular expressions.

Alphabet {0, 1}

Accepted strings: (0 | 1)*0 *any string ending in an 0 e.g. 0, 10, 00, 11010100*

Accepted strings: 1(0|1)*0 *any string that starts with a 1 and ends with a 0 e.g. 10, 1010110, 10000, 101*

Accepted strings: .*(010).* *any string that contains the pattern 010 in it, it can be anywhere in the string e.g. 010, 111101000, 1010, 011101011*

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

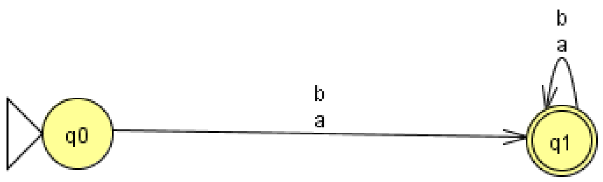
FSMs Exercise Sheet 3 Teachers Guide

Computational Thinking Course

Strand: Finite State Machines *Duration of Sheet: 20 minutes*

Q1. Are the given strings/regular expressions accepted by their corresponding Finite State Machine? Remember for a string to be accepted it must end on the accept state of the machine. Also see if you can figure out the "rule" i.e. what type of strings does the machine accept?

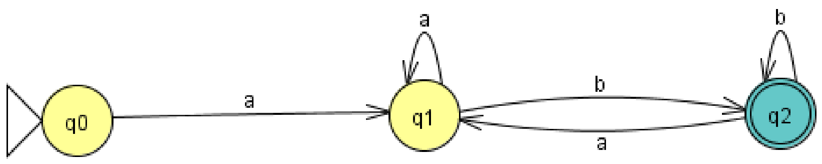
Machine 1:



String	Accept/Reject
a	Accept
b	Accept
c	Reject
abab	Accept
bbbbbbbb	Accept
abaffbab	Reject

The rule: Any string with at least one a or b, only contains a's and b's

Machine 2:

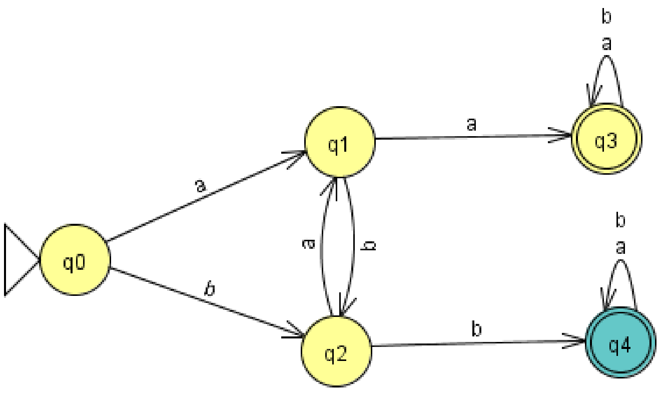


String	Accept/Reject
a	Reject

b	Reject
abc	Reject
abab	Accept
bbbbbbbb	Reject
abababbbbb	Accept
ababababa	Reject

The rule: Any word that starts with an a and ends with a b.

Machine 3:



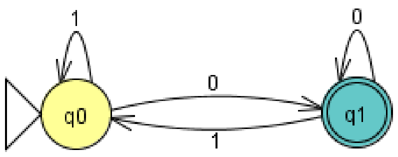
String	**Accept/Reject**
a	Reject
b	Reject
abc	Reject
abab	Reject
bbbbbbbb	Accept
abababbbbb	Accept
ababababa	Reject
abababaababab	Accept

The rule: Any word that contains a double aa or double bb i.e. two a's or b's in a row

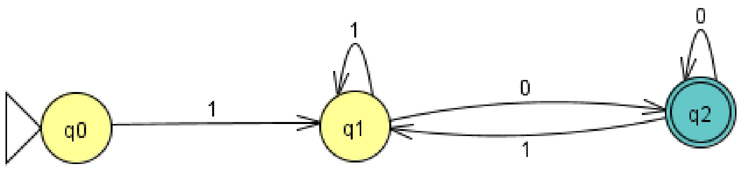
Q2. Design a FSM that accepts the following strings/regular expressions.

Alphabet {0, 1}

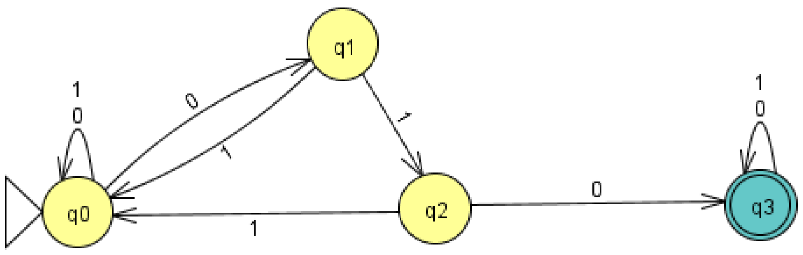
Accepted strings: (0 | 1)*0 *any string ending in an 0 e.g. 0, 10, 00, 11010100*



Accepted strings: 1(0|1)*0 *any string that starts with a 1 and ends with a 0 e.g. 10, 1010110, 10000, 101*



Accepted strings: .*(010).* *any string that contains the pattern 010 in it, it can be anywhere in the string e.g. 010, 111101000, 1010, 011101011*



Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Searching & Sorting

Strand: Computer Science Concepts *Duration of Lesson: 40 mins – 2 hours*

Learning Outcomes

Students will understand the idea of searching and how it is used in real-life and in computing.
Students will be familiar with several search algorithms including linear and binary.
Students will understand the idea of sorting and how it is used in real-life and in computing.
Students will be familiar with several sorting algorithms including bubble sort.
Students will understand that data can be sorted in different ways.

Curriculum links

Leaving Certificate Computer Science Specification

1.1 describe a systematic process for solving problems and making decisions

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.7 develop algorithms to implement chosen solutions

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

2.2 use a range of methods for identifying patterns and abstract common features

2.8 apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm

2.9 assemble existing algorithms or create new ones that use functions (including recursive), procedures, and modules

2.10 explain the common measures of algorithmic efficiency using any algorithms studied

3.7 use algorithms to analyse and interpret data in a way that informs decision-making

Junior Certificate Coding Short Course

1.4 develop appropriate algorithms using pseudo-code and/or flow charts

Computational Thinking Skills used

Algorithm

Computer Science concepts/topics involved

Linear search
Binary search
Bubble sort
Selection sort
Insertion sort
Merge sort

Resources / Materials

Pen, paper etc.
CS Unplugged Activity 6 - Battleships
A selection of random numbers between 1 and 100 written on cards, you'll need as many as people in your class minus 2.

The Class

Part 1 – Sorting

Activity – Introduction – 5-10 minutes

Give the students the Binary search Bebras problem. This introduces idea of binary search without using any data.

Activity – Development – 10 minutes

Use the introductory activity from the Battleships, it is described in detail but below is an overview.

- Choose about 15 students and give each a card with a number on it (in random order). Keep the numbers hidden.
- Give another student a container with four or five sweets/matchsticks/tokens in it. Their job is to find a given number. You could have 2 students compete to find it first.
- They can "pay" to look at a card. If they find the correct number before using all their sweets, they get to keep the rest.
- Now shuffle the cards and give them out again. This time, have the children sort themselves into ascending order. The searching process is repeated.

When the numbers aren't sorted, the only way of finding it is guessing, you could do this at random or start on the left/right and move along the line. This type of searching is called **linear search.** On average, this will take n/2 guesses, with n being the number of items.

If the numbers are sorted, we can do better than linear search. a sensible strategy is to use just one "payment" to eliminate half the children by having the middle child reveal their card. By repeating this process, they should be able to find the number using only three sweets. The increased efficiency will be obvious. This is called **binary search**.

Discussion – Conclusion – 5 minutes

So, to be able to search through data efficiently we need to sort it. *Why else might we want to sort data? How might we want to sort it?*

Part 2 – Sorting

Activity – Introduction – 5 – 10 minutes (can be dropped/shortened if time is an issue)

Split the class into two even groups. Explain they're going to take part in a race, first group to finish each task wins, most winning rounds wins the race. The idea is for the group to order themselves in terms of certain categories listed below. Get them to line up and then call out the following one at a time. After doing 1 or 2 categories, you could add in a rule that only one student can move at a time, switching with the person to either the right or left, this leads into **bubble sort**.

Height (tallest-> smallest)
Age (youngest-> oldest)
Name (alphabetically)
Birthday (earliest in the year -> last)

Activity – Introduction – 5-10 minutes (can be dropped/shortened if time is an issue)

Now show the students a series of datasets, see below & the slideshow for some examples. Get the students, in pairs if suits, to sort each one (write it down or just call out the method). Then show them it sorted (again see below & the slideshow). Hopefully some of them will get different answers from you. This shows that it's important to know what you're sorting by and how.

Data sets:

Original data set	Sorted Example 1	Sorted Example 2	Sorted Example 3
1,7,6,2,5,8	1,2,5,6,7,8 (numerically)	8, 5, 1, 7, 6, 2 (alphabetically - words)	
Tiger, mosquito, dog, oak tree	Dog, mosquito, oak tree, tiger (alphabetically)	Mosquito, dog, tiger, oak tree (size)	Mosquito, tiger, dog, oak tree (most likely to do you harm)
Maths, Geography, French, English, Business	Geography (9) Business (8) English (7) French (6) Maths (5) (number of letters)	Business, English, French, Geography, Maths (alphabetically)	Maths, Geography, English, Music, French (how much I like them as a subject!)

Coca Cola, Tropicana Orange Juice, Ballgowan Water, Nescafé Original, Barrys Tea	Coca Cola (1886) Barrys Tea (1901) Nescafé (1938) Tropicana Orange Juice (1947) Ballgowan Water (1981) (year first made)	Coca Cola (10.6g) Tropicana Orange Juice (9.1g) Nescafé (3.1g) Barrys Tea (< 0.1g) Ballgowan Water (0g) (sugar content per 100ml)	Coca Cola (1.8 billion bottles a day -> approx. 35 billion L a year if 500ml bottles) Tropicana Orange Juice (500 million L +) Ballgowan Water (30 million L +) Barrys Tea (data unavailable) Nescafé (data unavailable) (most sold/produced)
Mt Everest, Carrauntoohil, Spire, Table Mountain SA, Statue of Liberty	Everest (8848m) Table Mountain (1,084.6 m) Carrauntoohil (1038m) Spire (120m) Statue of Liberty (93m) (height)	Spire (750m) Carrauntoohil (311km) Statue of Liberty (5125km) Everest (7756km) Table Mountain (10007km) (furthest away from Connolly Station)	Spire Carrauntoohil Statue of Liberty Everest Table Mountain (furthest south)

Activity – Introduction – 5-10 minutes

Today the students will learn different sorting methods and talk about how and why some methods are quicker than others. To introduce the idea that sorting can be done in a specific way, using an algorithm, get students into groups of 3-4 and have them attempt the Bebras problem "Sorting Tree Trunks".

Activity – Development – 10 minutes

There are many widely used sorting algorithms and others which are commonly taught. In the next activity, we'll look at some of these but for now students can come up with one of their own! Give them a set of numbers (you could use playing cards or else have numbers printed off). The aim is then to come up with rules as to how to sort them with the following constraints:

- You can only move one number at a time.
- You want to try and limit the amount of numbers you're "remembering". By this we mean that if you take a number out of the list and move others around before placing it back in, try not to take more than a couple of numbers at a time.

Have the students trial their algorithms (the rules) and see how quick they think they are.

Discussion – Development – 5 minutes

Why would we want to have quicker ways of sorting things?

Activity – Development 10-20 minutes

One sorting algorithm at a time, explain the method to the students. Start with bubble sort, followed by selection sort and then insertion sort. If you have time and think the students will be up for it then you can also show them merge sort.

As you do each sorting algorithm you shoud give them all a set of 6-12 numbers and get them to write out each step of the sorting algorithms and show how they'd sort the set.

Discussion – Conclusion – 5 minutes

Recap what you've covered in this class, see if they agree that searching and sorting data is an important thing to get right and to be able to do quickly.

Feedback – Conclusion – 5 minutes

Please get your students to fill out the feedback form.

Additional activities

Activity – 10-20 minutes

To explain a variety of different sorting algorithms you can race each one against another. All the algorithms are described in detail in the file named Sorting Algorithms.

First, we're going to start with bubble sort v selection sort. Split the class in half, for example two groups of 14. Select one student from each group, they're going to be the "controller". Give the rest a random number card and tell them not to look at it. Get each half to line up in a random order and then turn their cards around. The controller then must go through and use the sorting algorithm technique assigned to them. For this first round one team will be bubble sort, the other will be selection.

In theory selection sort, should win, to make sure this happens you can give the controller an assistant who counts how many swaps are made. Selection sort is more efficient when it comes to the number of swaps made versus bubble sort. This is important because swapping is the most time/resource consuming part of sorting for a computer.

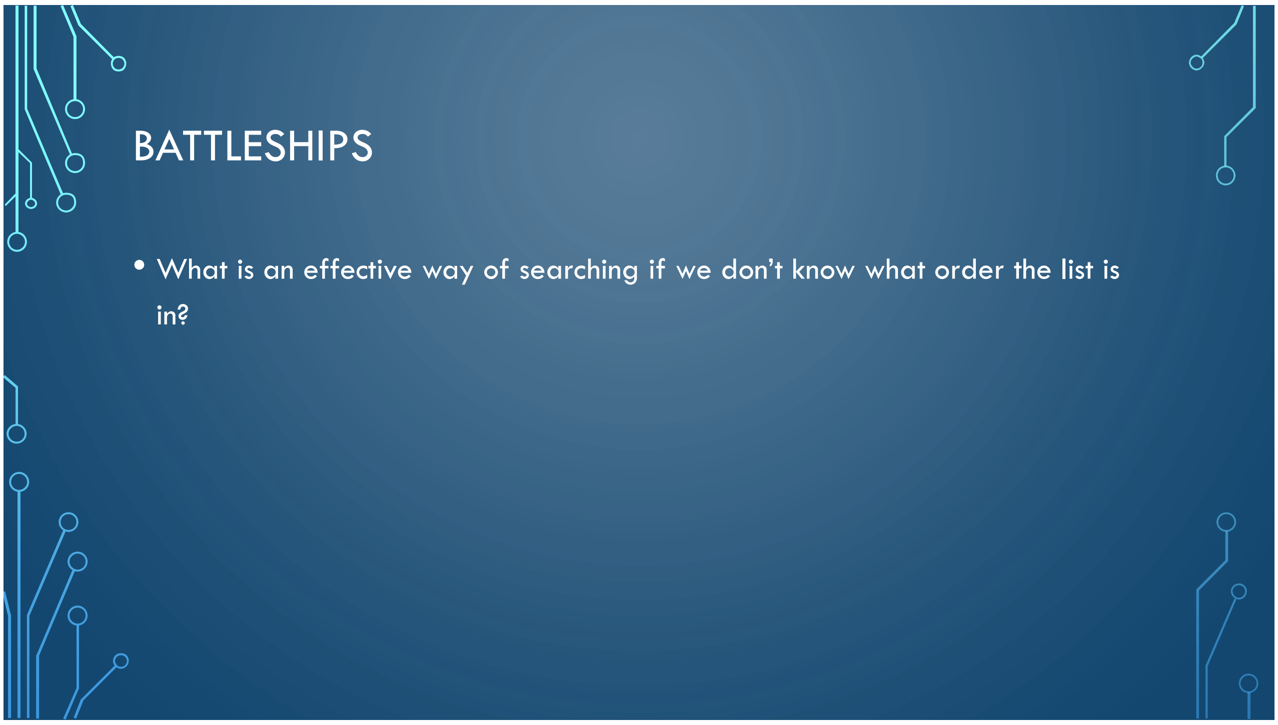
The next race should be selection vs. insertion sort, swap the controller and the assistant with another pair to give more students a chance.

SEARCHING & SORTING

CONCEPTS & INTRODUCTION

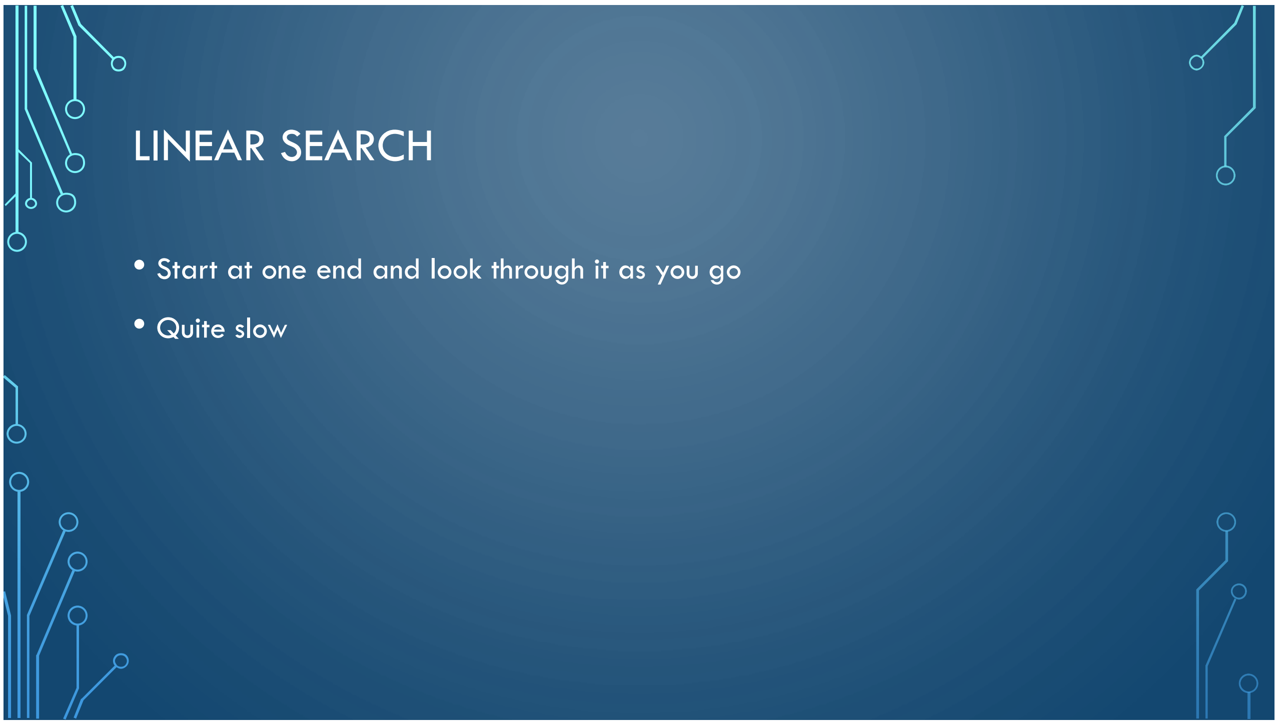
Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



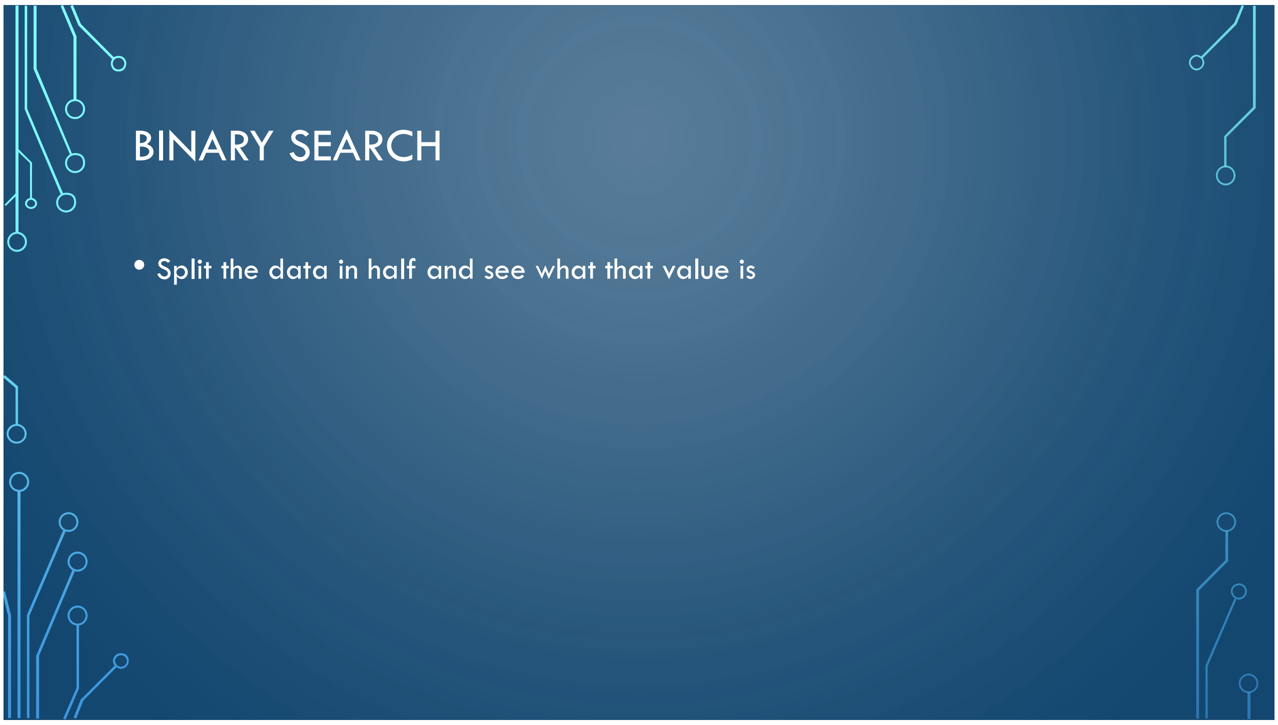
BATTLESHIPS

- What is an effective way of searching if we don't know what order the list is in?



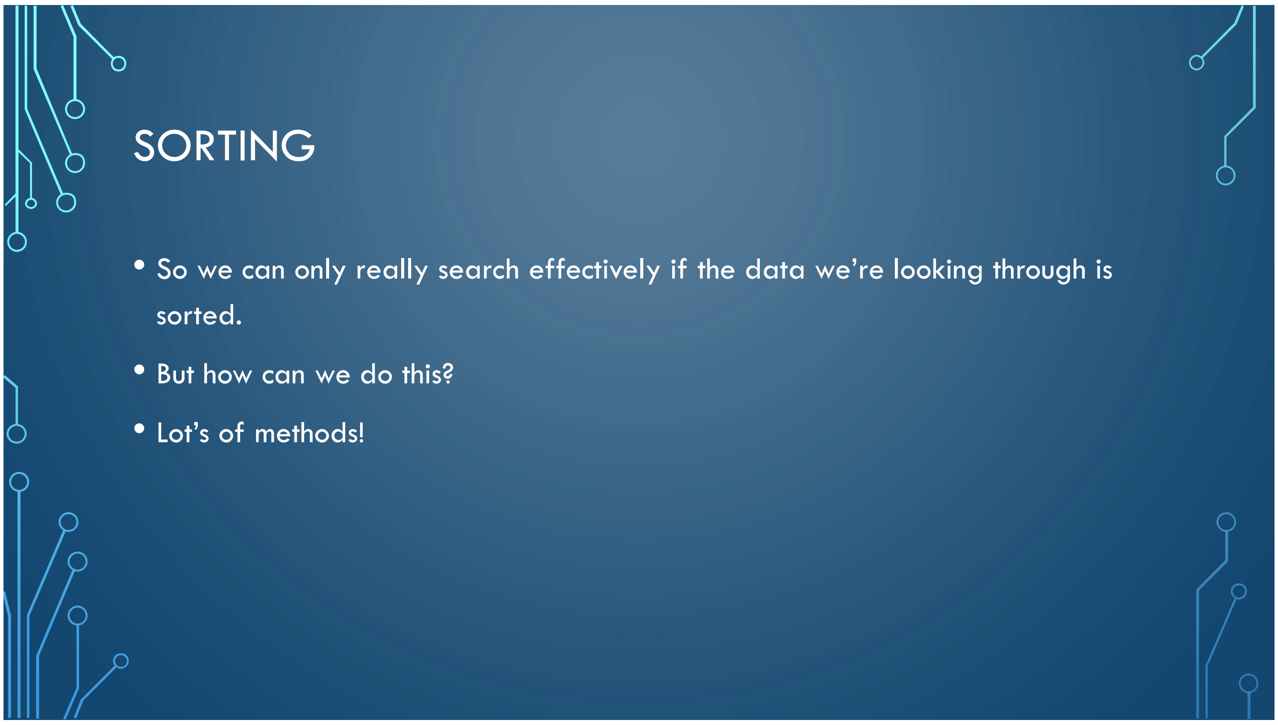
LINEAR SEARCH

- Start at one end and look through it as you go
- Quite slow



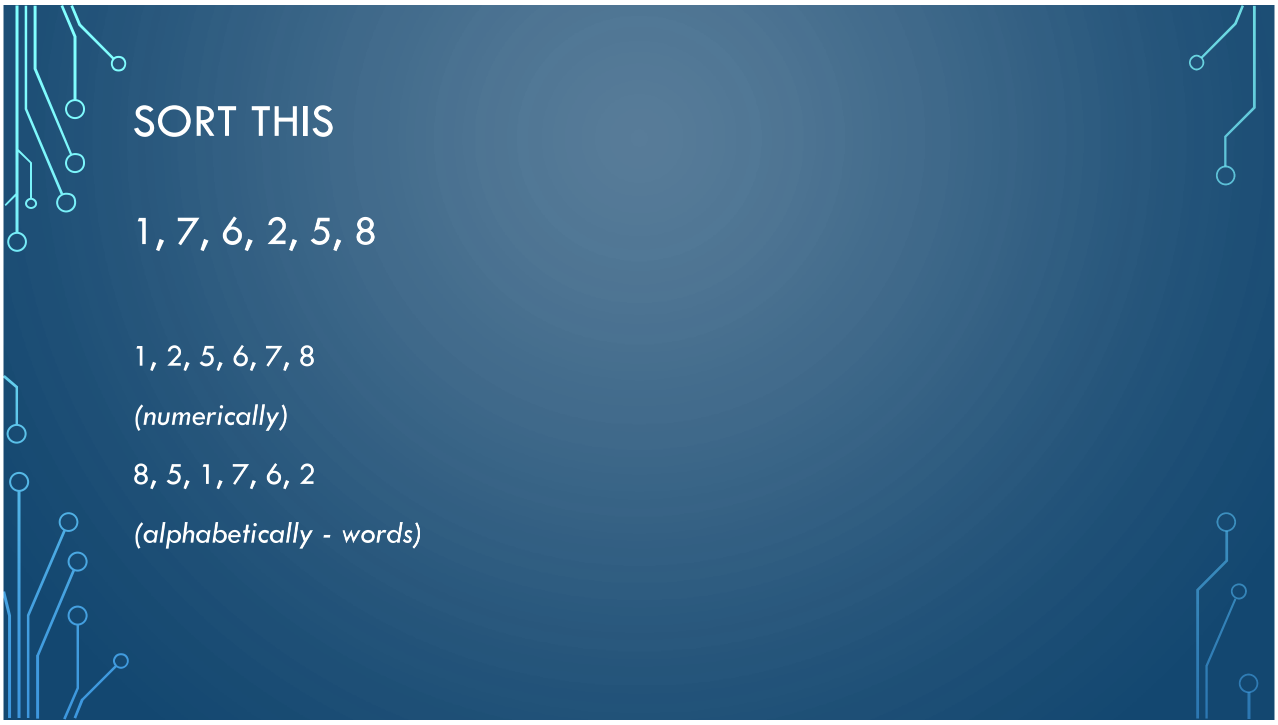
BINARY SEARCH

- Split the data in half and see what that value is



SORTING

- So we can only really search effectively if the data we're looking through is sorted.
- But how can we do this?
- Lot's of methods!



SORT THIS

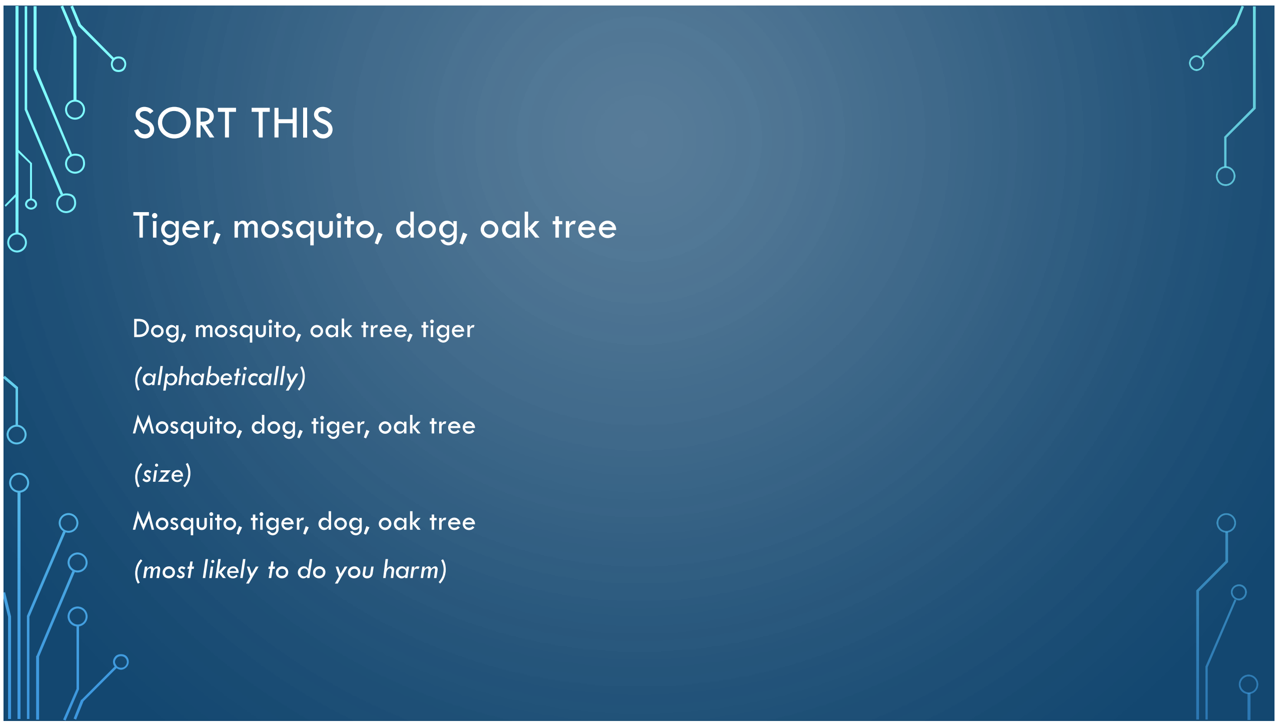
1, 7, 6, 2, 5, 8

1, 2, 5, 6, 7, 8

(numerically)

8, 5, 1, 7, 6, 2

(alphabetically - words)



SORT THIS

Tiger, mosquito, dog, oak tree

Dog, mosquito, oak tree, tiger

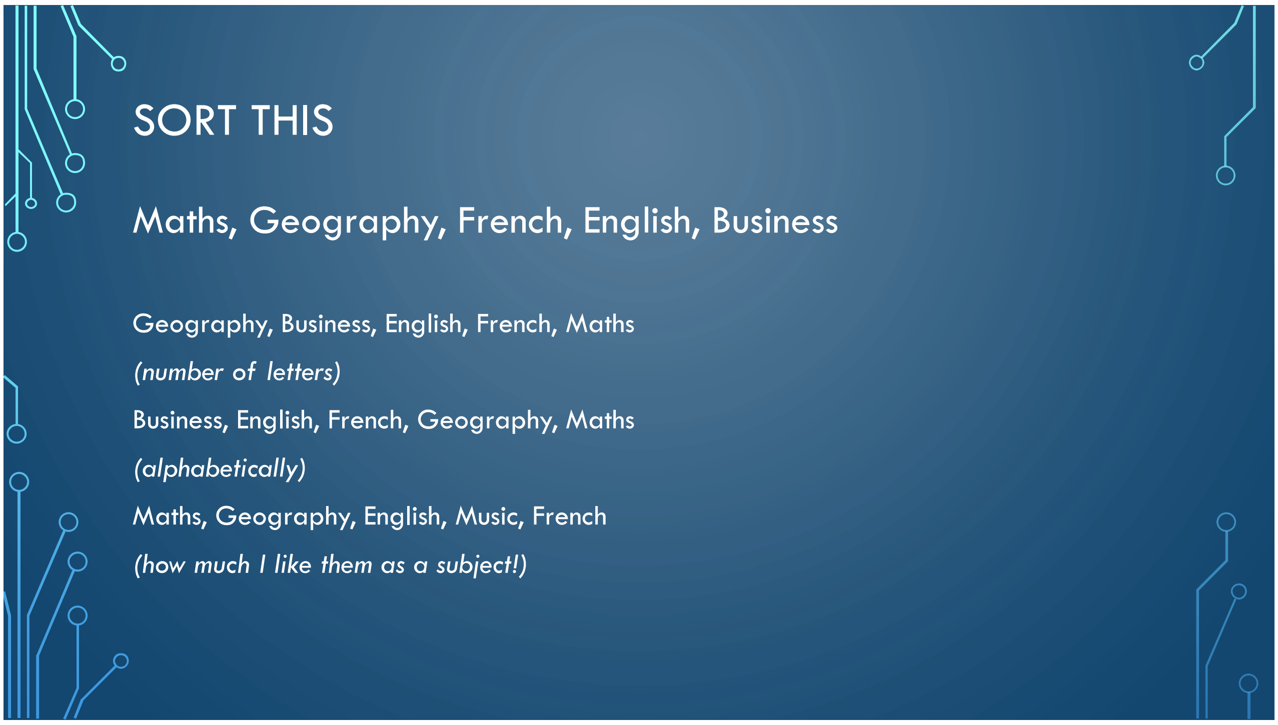
(alphabetically)

Mosquito, dog, tiger, oak tree

(size)

Mosquito, tiger, dog, oak tree

(most likely to do you harm)



SORT THIS

Maths, Geography, French, English, Business

Geography, Business, English, French, Maths

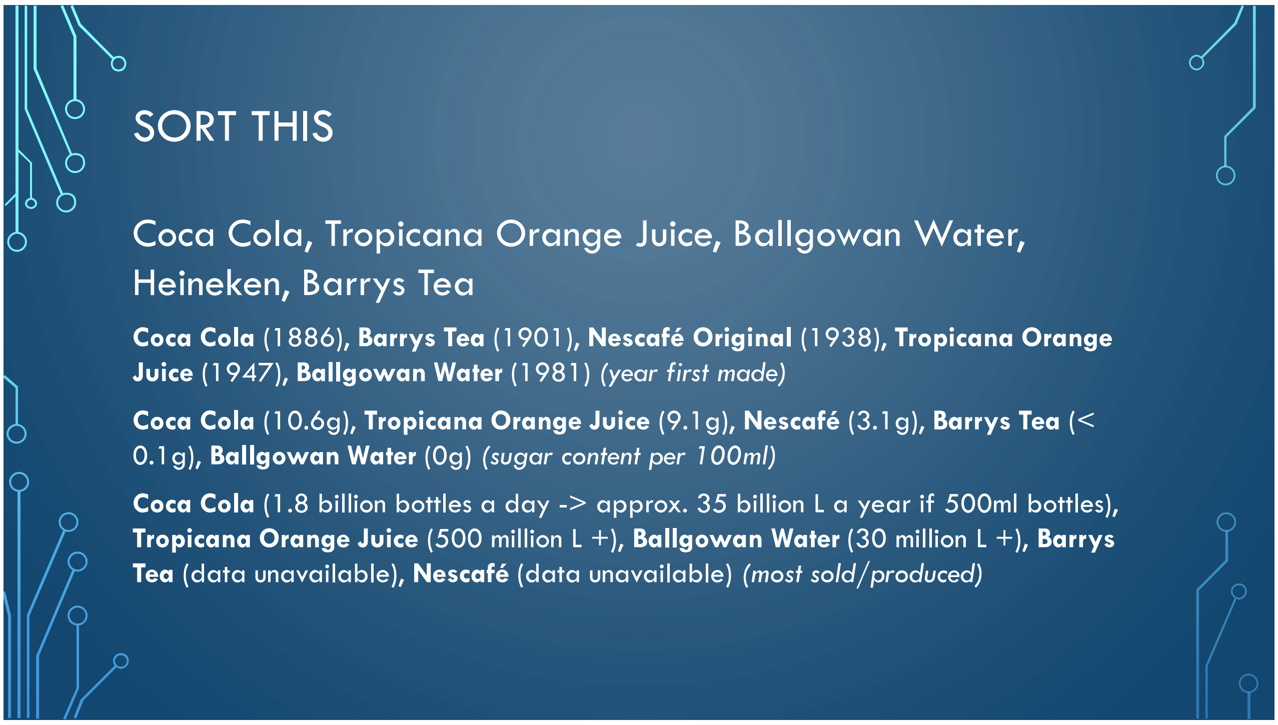
(number of letters)

Business, English, French, Geography, Maths

(alphabetically)

Maths, Geography, English, Music, French

(how much I like them as a subject!)



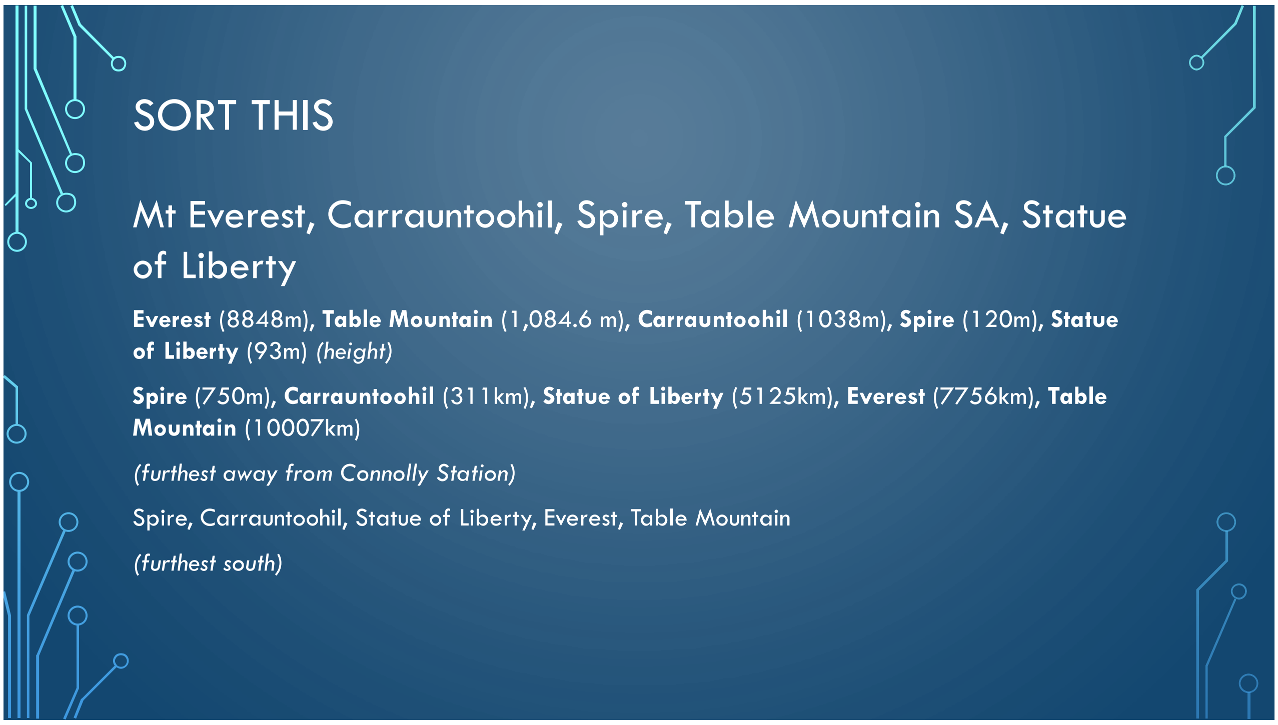
SORT THIS

Coca Cola, Tropicana Orange Juice, Ballgowan Water, Heineken, Barrys Tea

Coca Cola (1886), **Barrys Tea** (1901), **Nescafé Original** (1938), **Tropicana Orange Juice** (1947), **Ballgowan Water** (1981) *(year first made)*

Coca Cola (10.6g), **Tropicana Orange Juice** (9.1g), **Nescafé** (3.1g), **Barrys Tea** (< 0.1g), **Ballgowan Water** (0g) *(sugar content per 100ml)*

Coca Cola (1.8 billion bottles a day -> approx. 35 billion L a year if 500ml bottles), **Tropicana Orange Juice** (500 million L +), **Ballgowan Water** (30 million L +), **Barrys Tea** (data unavailable), **Nescafé** (data unavailable) *(most sold/produced)*



SORT THIS

Mt Everest, Carrauntoohil, Spire, Table Mountain SA, Statue of Liberty

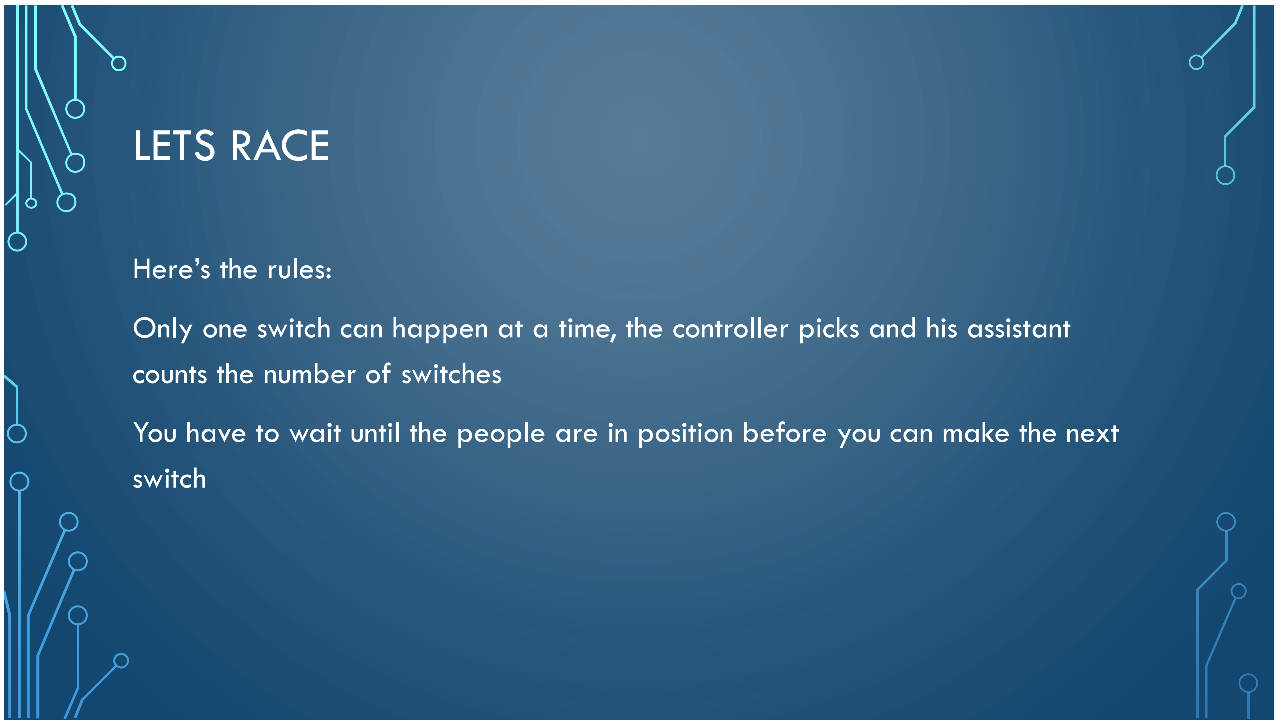
Everest (8848m), **Table Mountain** (1,084.6 m), **Carrauntoohil** (1038m), **Spire** (120m), **Statue of Liberty** (93m) *(height)*

Spire (750m), **Carrauntoohil** (311km), **Statue of Liberty** (5125km), **Everest** (7756km), **Table Mountain** (10007km)

(furthest away from Connolly Station)

Spire, Carrauntoohil, Statue of Liberty, Everest, Table Mountain

(furthest south)

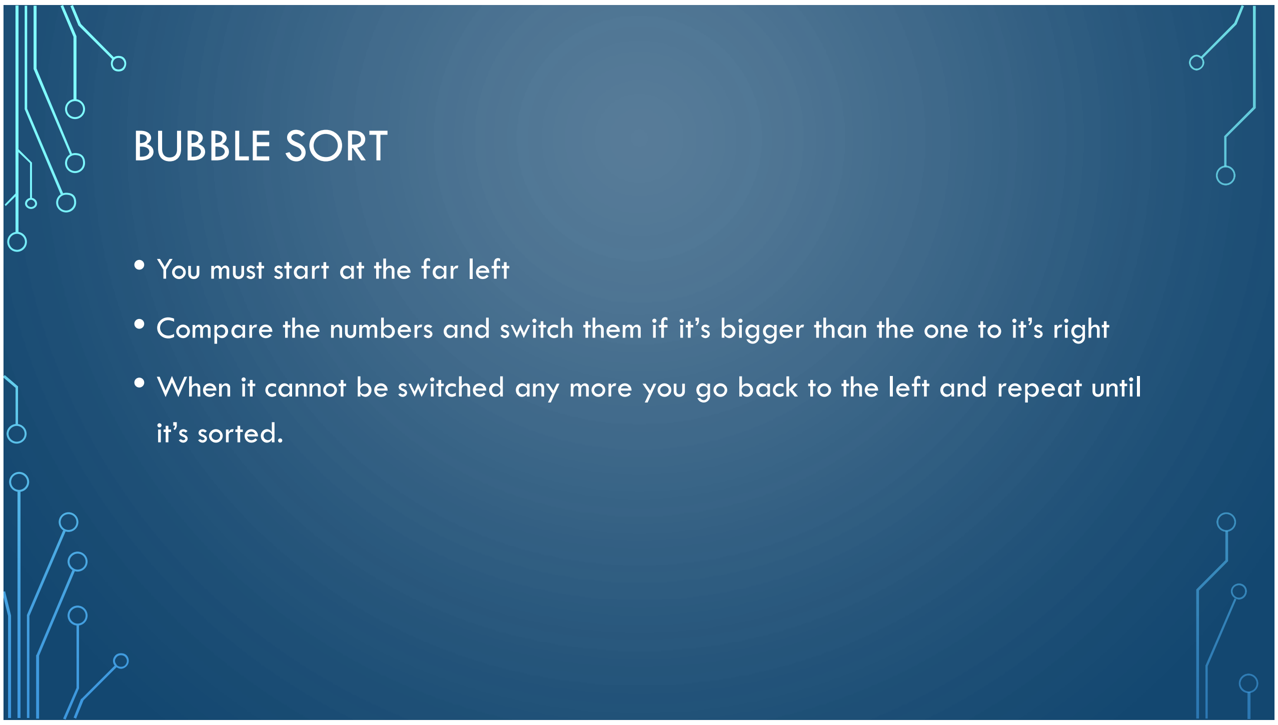


LETS RACE

Here's the rules:

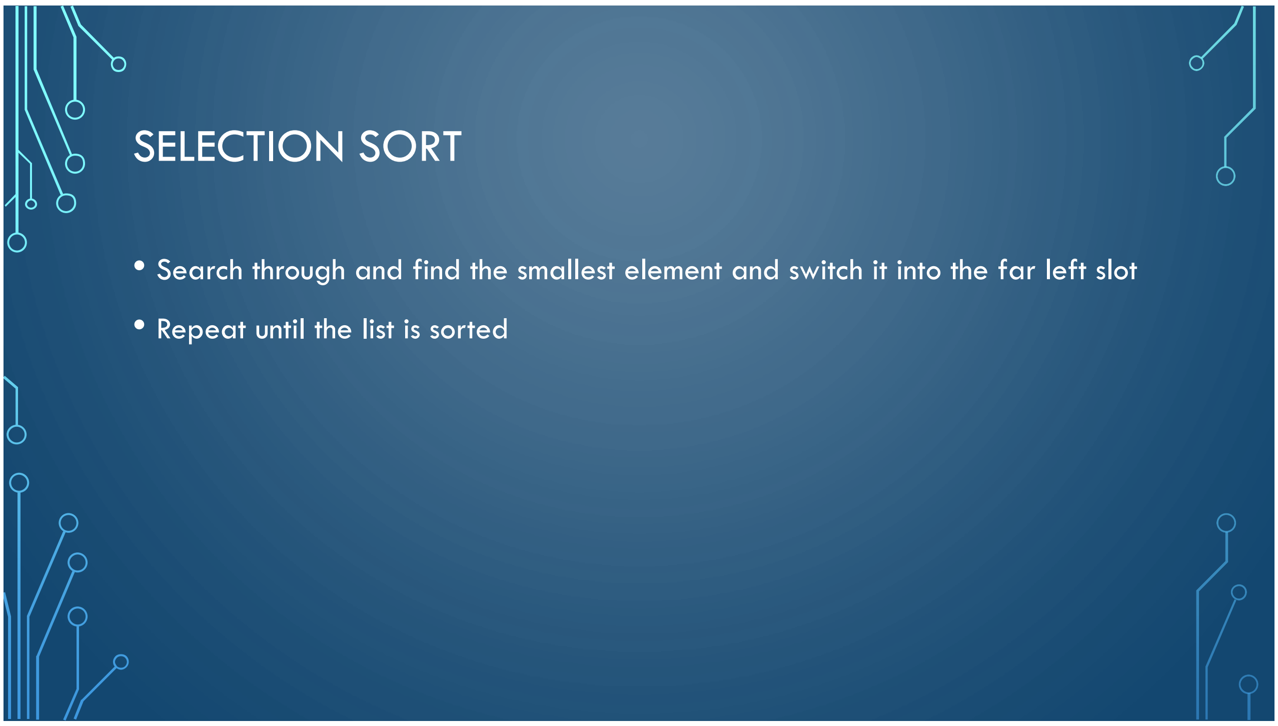
Only one switch can happen at a time, the controller picks and his assistant counts the number of switches

You have to wait until the people are in position before you can make the next switch



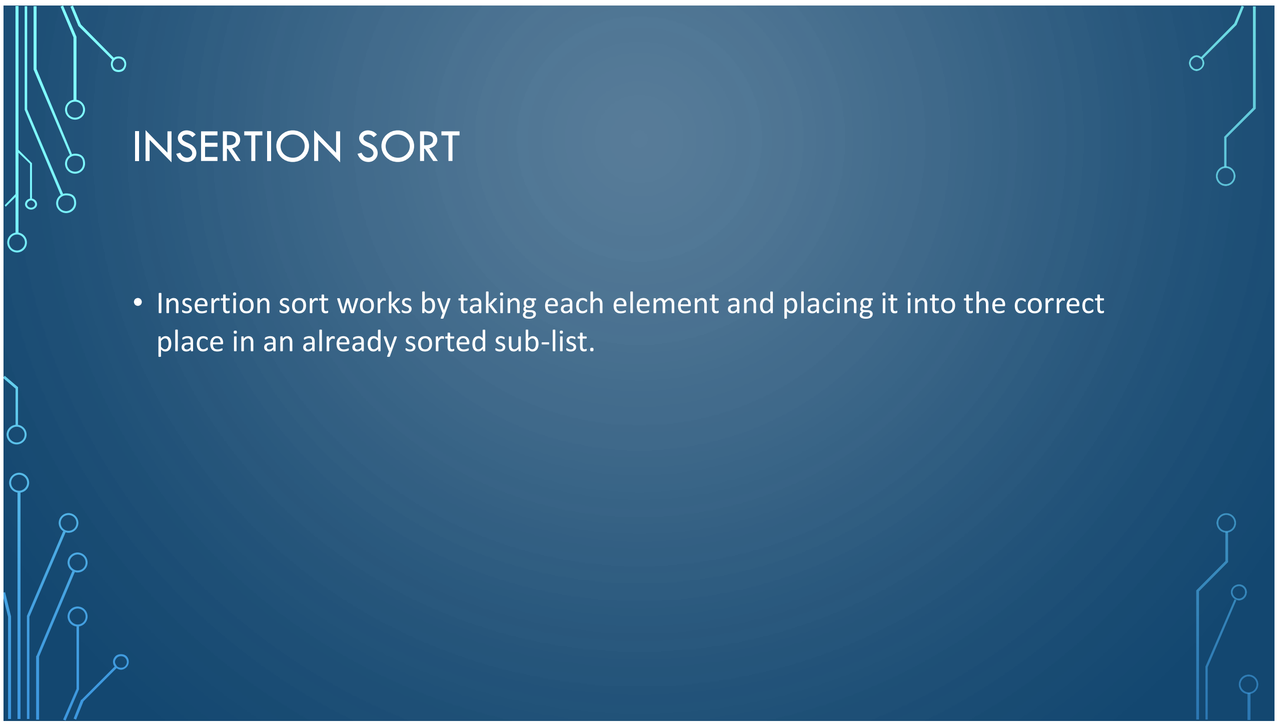
BUBBLE SORT

- You must start at the far left
- Compare the numbers and switch them if it's bigger than the one to it's right
- When it cannot be switched any more you go back to the left and repeat until it's sorted.



SELECTION SORT

- Search through and find the smallest element and switch it into the far left slot
- Repeat until the list is sorted



INSERTION SORT

- Insertion sort works by taking each element and placing it into the correct place in an already sorted sub-list.

Developed by James Lockwood
& Dr Aidan Mooney

Loops & Conditionals

Strand: Computer Science Concepts *Duration of Lesson: 40 mins – 1 hour*

Learning Outcomes

Students will be introduced to some basic programming concepts, namely, Loops and Conditional Statements.

Students will understand why loops, conditionals and functions are helpful in programming and how they work.

Curriculum links

Leaving Certificate Key Skills

- Critical and creative thinking
    - Identifying and analysing problems and decisions, exploring options and alternatives, solving problems and evaluating outcomes
    - Thinking imaginatively, actively seeking out new points of view, problems and/or solutions, being innovative and taking risks
- Communicating
    - Analysing and interpreting texts and other forms of communication
    - Expressing opinions, speculating, discussing, reasoning and engaging in debate and argument
- Working with others
    - Working with others in a variety of contexts with different goals and purposes
    - Identifying, evaluating and achieving collective goals

Junior Certificate Key Skills

- Communicating
    - Listening and expressing myself
    - Discussion and debating
- Being Creative
    - Exploring options and alternatives
    - Implementing ideas and taking action
    - Learning creatively
- Working with others
    - Co-operating
    - Learning with others
- Managing Information and Thinking
    - Thinking creatively and critically

Computational Thinking Skills used

Algorithm

Computer Science concepts/topics involved

Resources / Materials

The Class

Discussion – Introduction – 5-10 mins

In this lesson, we're going to look at three important parts of programming. We'll introduce them without computers to begin with and this can be taught without doing any programming. If you use either the Scratch or Python modules you could use these as an introduction, or just remind students that they've learned these concepts when you get to programming!

If you have already done part 1, recap that lesson on variables and user input.

If not, begin by asking students to brainstorm what they think a good computer program needs in terms of functionality. This means not what it looks like or specifically what it does, but what does a program (piece of software, app etc.) need to be useful?

Examples could include:

- A program needs to be able to remember things like your login information or credit card details, so it needs some sort of memory.
- You usually need to be able to interact with the program through the keyboard, mouse, voice activation etc.
- Being able to do common functions (like copying & pasting) quickly and easily.
- You might also have to have a way of going backwards (undo button or back button on a browser)
- You might want to do something repeatedly, maybe for a set amount of times.

Hopefully your students come up with some of the examples above or others. Explain that during this lesson (and the first part) we'll have a look at some of the common and key building blocks in programming. These don't always look the same, but they are all used in programming languages.

Explanation – Introduction – 10 mins

The first concept we'll look at today is loops. Loops are a way of doing something over and over again in a computer program. This would be useful in many cases, for example if you want a user to give you a list of numbers or names, or if you want to keep checking to see if a specific key is being pressed forever. There are a number of different loop types, but we won't look at them we'll just look at the concept in general. Usually in programming languages they can look something like this:

```
while a condition is true:
```

```
do this

update something so the condition eventually becomes true
```

One way to begin an introduction to loops is to tell a student to stand up and walk around their table and sit down. Then do the same thing again, and repeat it several times. Ask the students if there was a better way of doing this. The answer is a loop. Another analogy is a running track, if you're in a race you keep running around the track a number of times before you finish.

This is what a loop does in programming, it does something many times until some condition is met. For example, it might display the message "Hello" 10 times. To do this a counter (or variable) would be set to 0, you would then print the message "Hello" and increase the counter. You would then keep printing the message until the counter is 10.

Activity – Development – 20 minutes

There are a number of good resources developed to teach the concept of loops as unplugged lessons. One of these is developed my Microsoft as part of their Microbit curriculum (https://makecode.microbit.org/courses/csintro/iteration/unplugged). This makes students act out the repeated steps of an algorithm in real life. Another great resource can be found here: https://researchparent.com/coding-a-lego-maze/. This allows students to learn about while loops, for loops, if statements and infinite loops through Lego mazes. This is a great activity and has all of the resources available to print, but you will need a lot of Lego bricks to run it.

Explanation – Introduction – 10 mins

Another key programming concept is selection statements. These are usually written in something like this:

```
if something is true:

do this

else if something else is true:

do this

…

else none of the above are true:

do this
```

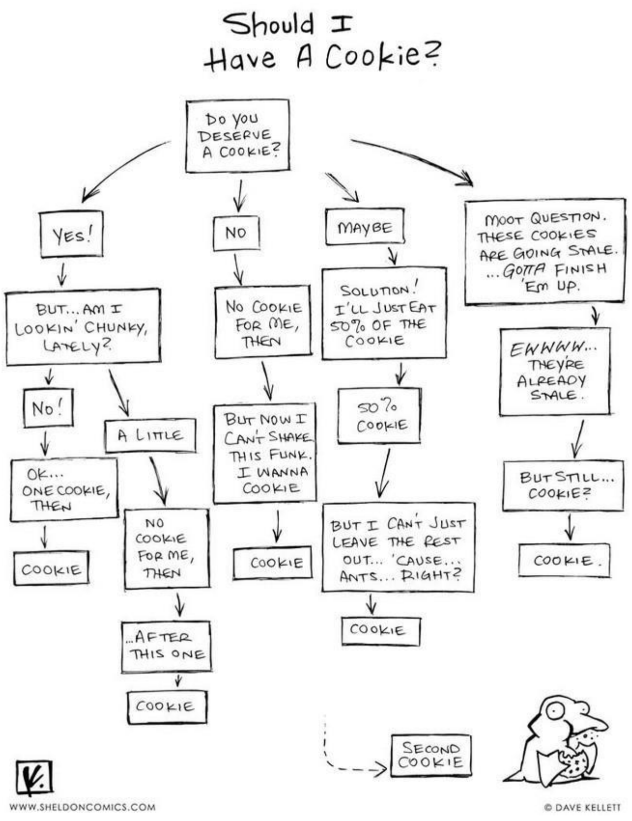
Selection statements (also known as conditional statements) are very useful if you want your program to behave in different ways depending on certain things being true. There are many examples of this in real life, see if the class can come up with any.

Examples include being allowed to buy 18's movies, **if** you're below 18 years of age you can't but **if** you are then you can. **If** you're queuing for a flight and your number is below 20 they might call you first,

else you have to wait to be called. **If** you support Manchester United you'll be happy when they win, **else if** you support Manchester City you'll be sad, **else** you might not care!

Activity – Development – 20 minutes

A good activity to introduce the idea of selection statements are decision tree diagrams. These can be useful for making choices but are also sometimes just good fun like the image below. Get your students to come up with some ideas and draw up decision trees. In our one's we are always going to have options that are true or false (i.e. 15 years old or younger, red hair, likes reading) so that we remove any ambiguity. Examples could be what food to choose at a restaurant based on things like how hungry you are, how much money you have etc. or which football team to support based on things like favourite colour, where you live etc.


Should I Have A Cookie?
Do you deserve a cookie?
Yes!
But... am I lookin' chunky, lately?
No!
Ok... one cookie, then
Cookie
A little
No cookie for me, then
...after this one
Cookie
No
No cookie for me, then
But now I can't shake this funk. I wanna cookie
Cookie
Maybe
Solution! I'll just eat 50% of the cookie
50% cookie
But I can't just leave the rest out... 'cause... ants... right?
Cookie
Moot question. These cookies are going stale. ...Gotta finish 'em up.
Ewwww... they're already stale.
But still... cookie?
Cookie.
Second cookie
WWW.SHELDONCOMICS.COM
© DAVE KELLETT

Make sure you drive home to the students that each branch of the tree is an if/else statement, only one can be true and the "program" does something different depending on the outcome.

Feedback – Conclusion – 5 minutes

Please get your class to fill out the Loops and Conditionals feedback form.

Activity – Development – 20+ minutes (optional, could be done after both programming concept lessons)

Now that students are aware of different programming concepts, it makes sense to move onto some sort of programming. However, before you move on the following activity can be a good way to show students how some of these concepts work very practically but also in a very fun way. The idea is to use the concepts of variables, user input, loops, conditionals and functions to make games.

Have a variety of game material and equipment available (see the list at the end for suggestions) and tell students they're going to make their own games. Using whatever materials, they like, they must come up with the rules and general gameplay. This is then to be written in a pseudo-code format using the programming concepts we've looked at. Below is a very basic example:

The game:

Players take it in turns to roll a die. The roll of the die is added to their current score. First player to reach 20+ wins.

Variables:

Player's scores

Current die roll

Conditional:

If the players score is over 20, they win

Loop:

Keep playing until the above condition is true

User input:

Players must roll the die and then pass it onto the next person

Material examples

- Dice
- Playing cards
- Tokens (could be buttons, stones or anything similar)
- Game boards (could design their own or use a chess board or other board)
- Game pieces (chess pieces, dominos, jenga blocks etc.)

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Variables & User Input Lesson Plan

Strand: Computer Science Concepts *Duration of Lesson: 40 mins – 1 hour*

Learning Outcomes

Students will be introduced to some basic programming concepts, namely, variables and user input.
Students will understand what a variable is and why it is useful.
Students will understand how variable naming and assignment work.

Curriculum links

Leaving Certificate Computer Science Specification

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

1.4 solve problems using skills of logic

1.5 evaluate alternative solutions to computational problems

1.6 explain the operation of a variety of algorithms

1.9 use modelling and simulation in relevant situations

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

2.5 use pseudo code to outline the functionality of an algorithm

2.6 construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement

Junior Certificate Coding Short Course

1.2 describe the main components of a computer system (CPU, memory, main storage, I/O devices, buses)
1.3 explain how computers are devices for executing programs via the use of programming languages
1.6 discuss and implement core features of structured programming languages, such as variables, operators, loops, decisions, assignment and modules

3.4 describe program flow control, e.g. parallel or sequential flow of control – language dependent

Computational Thinking Skills used

Algorithm

Decomposition
Abstraction

Computer Science concepts/topics involved

Variables
User input

Resources / Materials

The Class

Discussion – Introduction – 5-10 mins

In this lesson, we're going to look at two important parts of programming. We'll introduce them without computers to begin with and this can be taught without doing any programming. If you use either the Scratch or Python modules you could use these as an introduction, or just remind students that they've learned these concepts when you get to programming!

If you have done part 2 of this lesson on Loops, Conditionals and Functions then recap that lesson.

If not, begin by asking students to brainstorm what they think a good computer program needs in terms of functionality. This means not what it looks like or specifically what it does, but what does a program (piece of software, app etc.) need to be useful?

Examples could include:

- A program needs to be able to remember things like your login information or credit card details, so it needs some sort of memory.
- You usually need to be able to interact with the program through the keyboard, mouse, voice activation etc.
- Being able to do common functions (like copying & pasting) quickly and easily.
- You might also have to have a way of going backwards (undo button or back button on a browser)
- You might want to do something repeatedly, maybe for a set amount of times.

Hopefully your students come up with some of the examples above or others. Explain that during this lesson (and the second part) we'll have a look at some of the common and key building blocks in programming. These don't always look the same but they are all used in programming languages.

Explanation – Introduction – 5-10 mins

The first one of these building blocks are variables. If you have done lessons talking about data ask students why storing and being able to access data is helpful, or perhaps why being able to sort and search data efficiently is useful.

One of the most important things that computers do is process data. They store values (text, images, numbers etc.) and then manipulate them, do calculations with them and much more.
For example, a program like Photoshop takes and image, edits it, combines them with others and then stores a final version. A messaging or emailing program stores the words you type, allowing you to edit them until you are ready to send the message.

When programming we need a way to tell the computer to store information, move it around and do things with it. To do that we need a way of specifying places to store data, these are called ***Variables***. As a program is likely to need many variables, we give variables names. These variables are then *assigned* a value.

Demonstration & Activity – Development – 10-15 mins

A common analogy and physical explanation of how this works uses boxes or letters. To help with this it is useful to have already prepared two boxes or two envelopes.
Take one box, the box is your variable. Give the variable a name, animal1, and write this on the box (or on a card which you stick onto the box). What we've done so far is *initialise* a variable. Now we need to *assign* it a value. Get a suggestion of an animal from the students in the class and write this on a piece of card, place this in the box. Repeat this with another box or envelope called animal2, and assign it a different value.

Now get the students, in pairs or groups, to try and figure out how we might swap the values around. Give them a few minutes to try and figure something out. If a group gets the correct answer (see below) get them to explain whilst you demonstrate. If they don't then explain the below answer.

To swap the variables we need to introduce a new variable or box, we'll call it temp. We then assign temp to be a copy of the value of animal1. Go over to the animal1 variable, and make a copy of what's inside (make sure you emphasise this, so they can see you're making a copy). Take this card and place it in temp. Now we are going to assign animal1 to be a copy of the value in the animal2 variable. Again, do this by making a copy and placing it into animal1. It's important to note at this point that variables can only hold one value at a time, so take out the value currently in animal1 and destroy it, and place the new value into it. See if the students can figure out the final step here!
The solution ends with assigning animal2 to be a copy of the value in temp, again destroying the current value in animal2.

Activity – Development – 10 mins (optional or good homework)

See Assignment Dry Run activity developed by Queen Mary University of London. There is one version for Scratch and one for Python, if you are going to be doing either of those modules we recommend using the appropriate example, if not choose whichever you're most comfortable with.

Discussion – Introduction – 5-10 mins

Another vital part of a program is user interaction. Get students to think of all the different ways in which computers and computer programs get users to interact with them or use them. Examples include:
Hardware examples:

- Keyboard (typing things)
- Mouse (moving around or clicking on things)
- Camera (facial recognition, videoing)
- Fingerprint scanner (unlocking, access)

Software examples:

- Filling in a form
- Clicking on a button/link
- Uploading or downloading something
- Answering questions or selection options

You'll notice that most of the software examples use either the mouse or the keyboard. These are the main ideas we'll focus on, and what we'll focus on in all our programming modules. Thankfully with programming languages like Scratch and Python the hard work of knowing when a specific button or key is being pressed is done for us. The point of this discussion is just to show students that user input is vital to most programs.
Explain that in the next lesson we'll look at a couple of other core parts of programming languages.

Feedback – Conclusion – 5 minutes

Please get your class to fill out the Variables and User Input feedback form.

Activity – Development – 20 + minutes (optional, could be done after both programming concept lessons)

Now that students are aware of different programming concepts, it makes sense to move onto some sort of programming. However, before you move on the following activity can be a good way to show students how some of these concepts work very practically but also in a very fun way. The idea is to use the concepts of variables, user input, loops, conditionals and functions to make games.

Have a variety of game material and equipment available (see the list at the end for suggestions) and tell students they're going to make their own games. Using whatever materials, they like, they must come up with the rules and general gameplay. This is then to be written in a pseudo-code format using the programming concepts we've looked at. Below is a very basic example:

The game:

Players take it in turns to roll a die. The roll of the die is added to their current score. First player to reach 20+ wins.

Variables:

Player's scores

Current die roll

Conditional:

If the players score is over 20, they win

Loop:

Keep playing until the above condition is true

User input:

Players must roll the die and then pass it onto the next person

Material examples

- Dice
- Playing cards
- Tokens (could be buttons, stones or anything similar)
- Game boards (could design their own or use a chess board or other board)
- Game pieces (chess pieces, dominos, jenga blocks etc.)

D

SCRATCH LESSONS

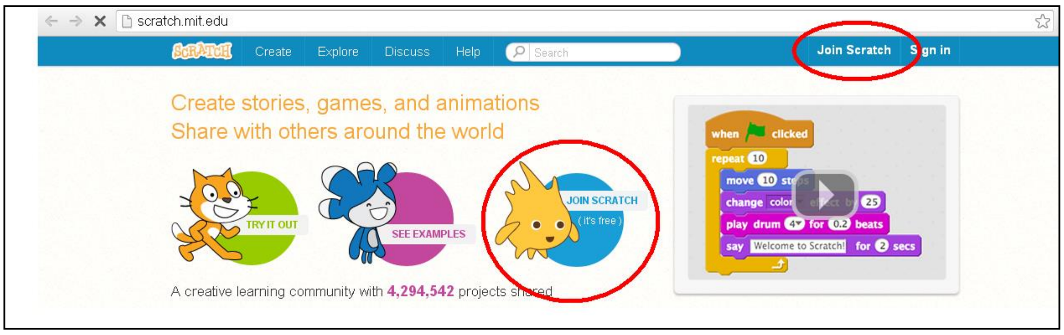
Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

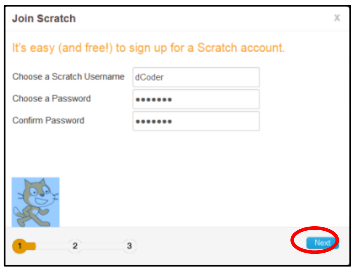
How to make a Scratch account

1. Go to http://scratch.mit.edu/

2. Click on "Join Scratch"

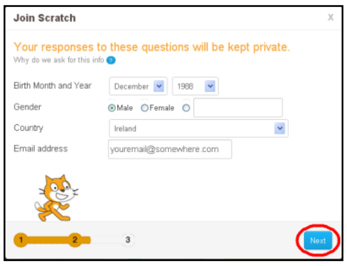

scratch.mit.edu
SCRATCH
Create
Explore
Discuss
Help
Search
Join Scratch
Sign in
Create stories, games, and animations
Share with others around the world
TRY IT OUT
SEE EXAMPLES
JOIN SCRATCH
(it's free)
when clicked
repeat 10
move 10 steps
change color effect by 25
play drum 4 for 0.2 beats
say Welcome to Scratch! for 2 secs
A creative learning community with 4,294,542 projects shared

3. Enter your information and click "Next" until you reach the end where you click "OK Let's Go"


Join Scratch

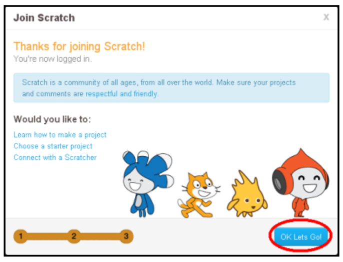
It's easy (and free!) to sign up for a Scratch account.

Choose a Scratch Username	dCoder
Choose a Password	•••••••
Confirm Password	•••••••


Join Scratch

Your responses to these questions will be kept private.
Why do we ask for this info

Birth Month and Year	December 1988
Gender	[✓ Male] Female
Country	Ireland
Email address	youremail@somewhere.com


Join Scratch

Thanks for joining Scratch!
You're now logged in.

Scratch is a community of all ages, from all over the world. Make sure your projects and comments are respectful and friendly.

Would you like to:
Learn how to make a project
Choose a starter project
Connect with a Scratcher

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Cat and Mouse Teachers Guide

Strand: Introduction to Programming - Scratch   Duration: 30 minutes +

Computer Science topics:

Programming
User input
Loops
Event handling

Computational Thinking skills:

Algorithm
Abstraction
Decomposition

Scratch tools:

User input: **Keyboard & Mouse**
Loops: **Repeat until**
Condition: **touching _?**
Motion: **Move, point towards mouse, change y, go to**
Appearance: **Import pictures, look/hide, draw a basic Sprite, Say**
Event handling: **When flag clicked, broadcast, receive, when key pressed**

Maynooth University
National University of Ireland Maynooth

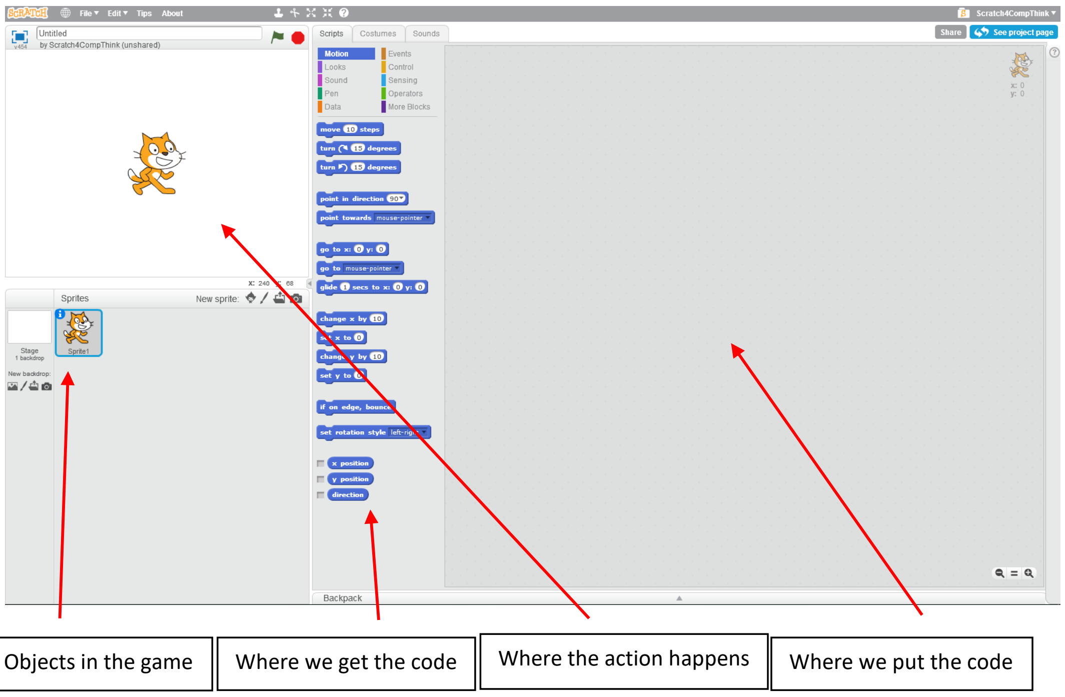
Developed by James Lockwood
& Dr Aidan Mooney

Cat and Mouse

We're going to make a game using Scratch which two players can play. The idea is one player will be trying to catch the other in a game of Cat & Mouse, or Cat & Dragon in my case!

Part 1 – Setup and Introduction

Go to the scratch website and make a new project. This is what the screen should look like:



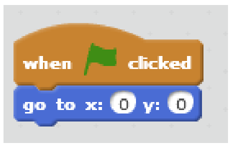
This is the start of all of your projects. We have one "Sprite" called Sprite1, this is the cat. In this game we're going to make the cat move using the arrow keys.

We're also going to make another Sprite and get it to follow the mouse. If this Sprite "catches" the other, we'll get them to both shout out something and player 2 will win.

Part 2 – Making the cat move

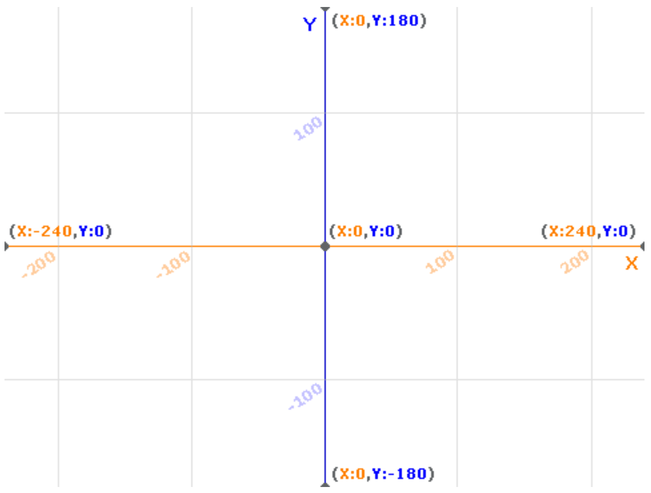
1. For now, we'll keep Sprite1 as it is, you can change it later if you like! For the rest of this tutorial we'll refer to Sprite1 as the Cat.

2. Our first step is to make sure whenever we restart our game that the Cat starts where we want it to. This is always a good idea as it resets everything.
3. To do this, get a "*When green flag clicked*" block from **Events**. Drag this into the coding pane. Next go to **Motion** and get a "*go to x: _ y: _*" block. Place this block under the flag block, it should "click" into place like the picture below.



4. I'll use the "_" symbol to show that the Scratch block can have a different value, that you can change it, so in our "*go to x: _ y: _*" block, you can change it the 0's to be something else, give it a try!

5. To check this works, drag the Cat somewhere else on the screen. Then hit the green flag, it should snap back to the middle!
6. Now we have to get the cat moving, to do this we're going to use the 4 arrow keys. This is one way which Scratch deals with something called *user input.* User input is how we can communicate with computer programs, common ways are clicking on buttons, pressing keys on the keyboard, speaking into a microphone or dragging something!
7. Go to **Events** and drag a "*when _ key pressed*" button from Events into the pane. Hit the little black arrow and click on "left arrow". Now we have to make the cat move when this is pressed. To do this go to **Motion** and place a "*move _ steps*" block under the key pressed block. Scratch uses an X-Y coordinate system to move Sprites around, below is the layout:

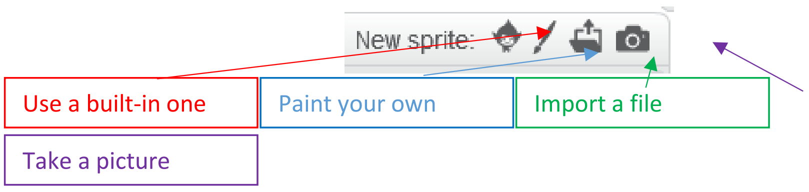


8. Try out your code so far! Click the green flag and press the left arrow.
9. What happens? Not quite right? See if you can fix it so that the cat moves left instead of right! **If you're stuck look at Hint 1.**
10. Now you know how to move the cat left using the left arrow, do the same thing for the right arrow (make sure you move it right not left!).

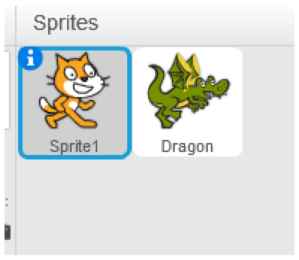
11. We're now halfway there! What we want to do now is get the cat to move up and down using the up/down arrow key. There's a few ways you could do this so give it a go**, if you get stuck take a look at Hint 2.**

Part 3 - Make another character

1. Now this game isn't really a game at all at the moment. Most games require at least two characters/players to make them any fun!
2. To do this we can click on the New Sprite button. We'll use one of the built-in pictures for now, but you can also draw your own, import a picture from a file or take a picture for one.



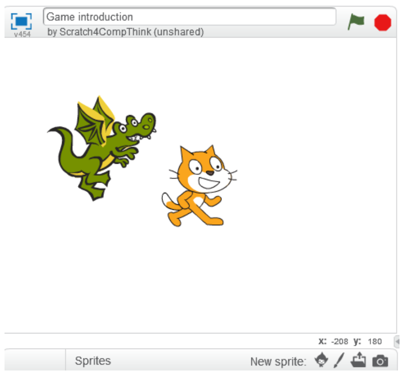
3. Click on the built-in button and pick anything you like from the clipart Scratch provides, I chose the Dragon.
4. When you hit ok it should appear next to the Cat like this:



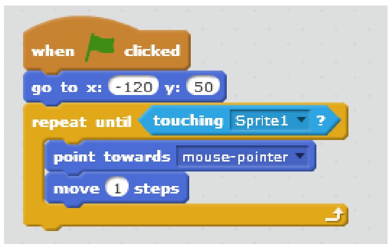
5. You'll see that the Script pane (where we put the code) is blank now. That's because everything we did previously was only to program the cat sprite. Now we need to make our new character move.

Part 4 – Make Player 2 follow the mouse

1. As with the Cat, we first want to make our new character always start in the same place. Do this and choose wherever you want on the screen to put them. (see the X,Y coordinate picture for an idea of how to move it). It should look something like the picture below when you hit the green flag!



2. Now we need to make the Dragon able to move, this time using the mouse. To do this we're going to have to use a "*repeat until _*" block. This is one of Scratch's version of what's called a ***loop*** in programming. These are helpful things as they let us do the same thing repeatedly without the need to write/move the same code over and over again!
3. Go to **Control** and place a repeat until block under the blocks you currently have. This is our loop! A loop always needs some ***condition***. If that condition is true then whatever is inside the loop ***executes***, if not it moves on. Our condition will be keep going until this Sprite touches the other one. To do this go to Sensing and get a "*touching _?*" block and place it in the slot on the "*repeat until _*" block. Click the drop-down arrow and select Sprite1, this should be the name of your Cat sprite.
4. Now we need to make the Dragon keep following the mouse pointer until it "catches" the cat. To do this Scratch has a handy block in **Motion** called "*point towards _*". Drag this into the "*repeat until _*" block and make sure to select mouse-pointer from the drop down arrow. This will make the Dragon "look" at the mouse but we need to make it move towards it. To do this just take a "*move _ steps*" block and put it into the "*repeat until _*" block and set it to be 1. Your code should look something like this:



```
when [flag] clicked
go to x: -120 y: 50
repeat until <touching Sprite1 ?>
    point towards mouse-pointer
    move 1 steps
```

5. Test this out and check the game is doing what it should! The cat should move using the arrow keys, the other sprite should follow the mouse and when it touches the cat it should stop moving.

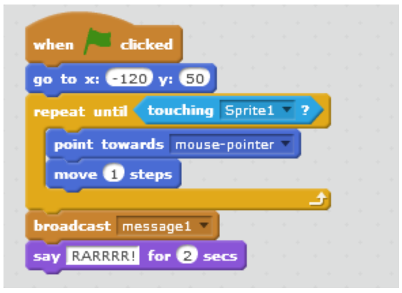
Part 5 – Winning the game

We now have a game that does what we want, but we don't really know if there's a winner.

1. We'll start with the winning part. To do this we're going to make the Cat "shout out" when it's caught and the Dragon roar. To do this we're going to need to add code to both Sprites. Seeing as we still have the dragons code open we'll start there.
2. The way our loop ("*repeat until _*") block works is that it'll keep executing what's inside until the condition is met. Afterwards it'll move on with the code. Anything we put after the "*repeat until _*" block will only happen once the cat has been "caught". Go to **Looks** and drag a "*say _ for _ secs*" and place it after the loop. Change "Hello!" to be something more suitable as a victory cry!

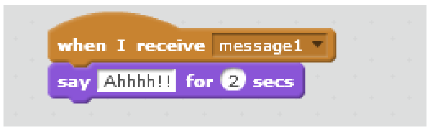
   Now for the cat it's a little bit trickier. What we're going to have to do is set up a "hidden" message that sends when the Dragon catches the Cat. What we mean by hidden is not something someone playing the game can see!

3. To do this Scratch has a "*broadcast _*" block. This is found in **Events**, drag it and place it above the "*say*" block we just used. This is so the Cat can speak at the same time as the Dragon, otherwise we'd have to wait 2 seconds! Your code should look like this:



```
when [flag] clicked
go to x: -120 y: 50
repeat until touching Sprite1 ?
    point towards mouse-pointer
    move 1 steps
broadcast message1
say RARRRR! for 2 secs
```

4. Now we have this "hidden" message called "message1" we can use this to tell the Cat the game is over!
5. Click on the picture of the Cat and the code should change to the Cat's code.
6. Go to **Events** and get a "*when I receive _*" block. Place this on and then get a "*say*" block from **Looks**, just like we did with the Dragon. Again get it to say something more appropriate! It should look like this:



```
when I receive message1
say Ahhhh!! for 2 secs
```

Now we have a completely working game! But we're not quite finished!

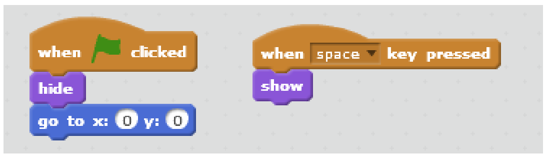
Part 6 – Making it look better

We have a complete game but it doesn't look very good! To do this we can change the backdrop, and even add in one!

1. To do this click on the Stage, it's to the left of the Sprites. Again the code will vanish as we haven't put any code for the Stage yet. Click on the Backdrops tab up at the top. We'll select a

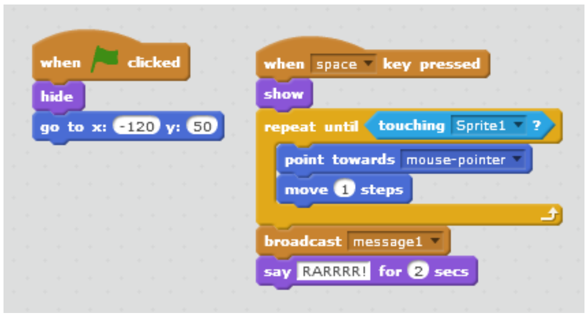
new backdrop from the Scratch library, but like with Sprites you can also draw your own, import a file or take a picture.

2. Click on the new backdrop button (from the library) and pick one. It should change automatically!
3. We're also going to add in a start screen. To start the game, we'll wait for players to press the space bar. To do this we'll have to change our Cat and Dragon code slightly. Go to the Cat sprite.
4. We're going to "*hide*" the Cat until the space bar is pressed. Go to **Looks** and get a "*hide*" block, place this under the "*When green flag clicked*" block. Then go to events and get a "*when _ key pressed*" and make sure it says space. Then go back to looks and get a "*show*" block, place it under this block. Your code should look like this:



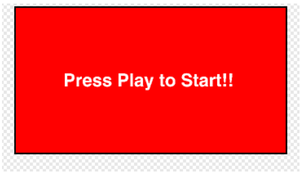
```
when [flag] clicked
hide
go to x: 0 y: 0

when space key pressed
show
```

5. For the Dragon it's the same except, instead of just placing the "*show*" under the "*when _ key pressed*" block, you're going to move our loop and all the code below it there as well. This is because we don't want the Dragon to move until the space bar is pressed. This is what your code should look like:



```
when [flag] clicked
hide
go to x: -120 y: 50

when space key pressed
show
repeat until touching Sprite1 ?
    point towards mouse-pointer
    move 1 steps
broadcast message1
say RARRRR! for 2 secs
```

6. Now we'll make a "begin game" message appear that tells the user to press space to start. To do this create a new Sprite and draw it using the rectangle ▬ and then fill it any colour by clicking this icon .
7. Now click on the text box icon T and write out the message "Press Play to Start". Make sure you move it into the middle of the rectangle so that it looks something like this:



Press Play to Start!!

8. Now move to this Sprites script page and this time we need this Sprite to appear when we press the green flag and hide when we press space (the opposite of what we've just done!). Give this a try, if you're stuck see Hint 3 below for the code.

Congratulations! You've just written your first computer program! Give the game a go with someone to check it all works correctly!

What now??

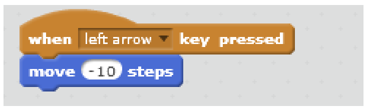
You can move on to the next tutorial!

Or…

If you want, think of ways you could make it easier/harder, look better etc. For example, you could make the Dragon move faster if it's too easy.

Hint 1

This is what you need to do to make the cat move left instead of right, it's -10 as you want it to move towards the negative area of the x axis!

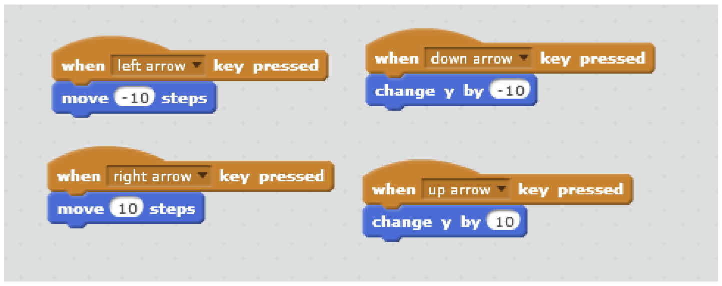


```
when left arrow key pressed
move -10 steps
```

Hint 2

This is how all the movement blocks can look. The right arrow is basically the same as the left arrow but change the value to 10 so it moves towards the positive area of the x axis!

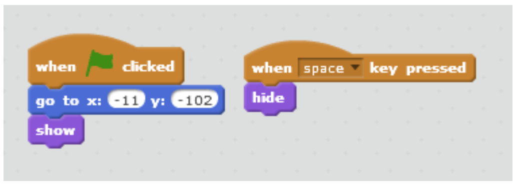
The way I've got the cat to move up and down is by using the "*change y by _*" block. This does a similar thing to the "*move _ steps*" block but it just adds/takes away the value from the y value the Sprite is currently at.



```
when left arrow key pressed
move -10 steps

when down arrow key pressed
change y by -10

when right arrow key pressed
move 10 steps

when up arrow key pressed
change y by 10
```

Hint 3

This is how to make the Press Space message appears at the start and then vanishes when the game starts. For the "*go to x: _ y: _* " block you can move it to anywhere you want.



```
when clicked
go to x: -11 y: -102
show

when space key pressed
hide
```

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Pong Teachers Guide

Strand: Introduction to Programming – Scratch Duration: 30 mins +

Computer Science Topics

Loops **(infinite loops)**
User Input
Programming
Event handling
Conditionals
Defining functions

Computational Thinking Skills

Algorithm
Abstraction
Decomposition

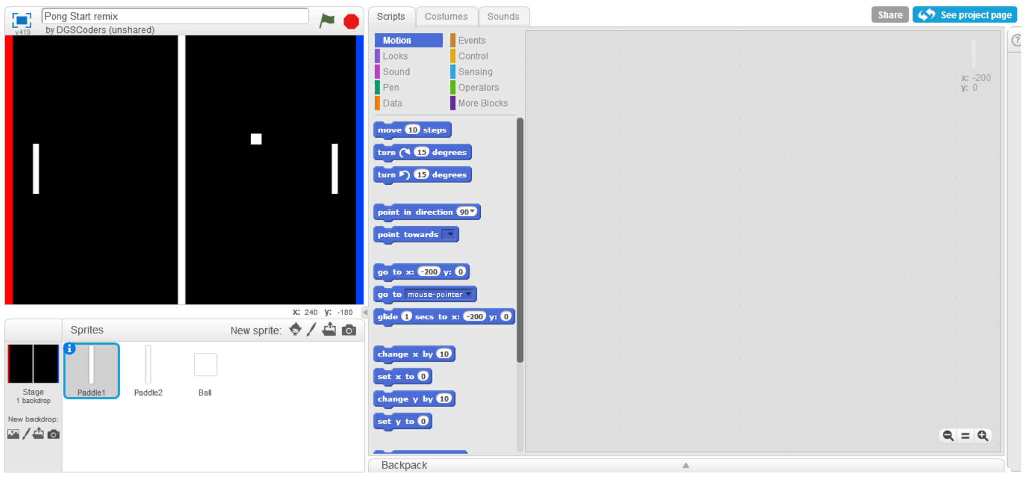
Scratch tools

User input: Keyboard
Variables: Number, random number, mathematical operators
Motion: Go to, move, turn, change y, **point in direction, if on edge bounce, direction**
Loop: **forever**
Conditional: **if then**
Event handling: When flag clicked, **wait, touching color**
Appearance: **Draw stage and sprites**
Defining our own block

Developed by James Lockwood
& Dr Aidan Mooney

Pong

Pong is a classic arcade game that is a bit like air hockey!



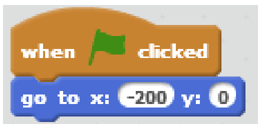
Before you get into coding you're going to have the make the stage and sprites like the picture above. It's a black background with a white line in the middle and a red strip on the left and a blue strip on the right side. You can make this by using the draw option on the backdrops.
You'll also need three Sprites. Two will look the same, thin white rectangles which will be our paddles and a white square which will be the ball. Again, use the draw option on the Sprites costumes. Name them Paddle1, Paddle2 and Ball.

Part 1 - Coding Paddle 1

1. Click on Paddle1 and then click on "Scripts".

2. When we start the game, we want our paddle to be in the correct position for a game so the first thing we will do is set this up. To start a game, we will press the green flag so we will need the *"When green flag is clicked"* code block.

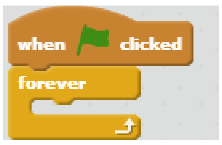
3. When our game starts we want the paddle to be in the right place. We want it to be at x=-200 and y=0 as you can see on the graph above. To move our paddle, go to the **Motion** section and drag a *"go to x: _ y: _"* block under the flag block. Change the x value to -200 and the y value to 0.



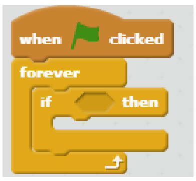
4. Try moving the paddle by clicking on it and dragging it and pressing the green flag and you will see the paddle jump into place.

5. The job of the paddle is to move up and down when they player of the game commands it to do so. If the player of your game hits the "a" key on the keyboard we want paddle1 to move up. If they hit the "z" key we want it to move down. We want the paddle to be able to move very fluidly and forever. To do this we're going to use something called an *infinite loop.* An infinite loop is a loop, but it has a condition that is always true, usually this is a bad thing but it can be helpful!
For example, you could set the condition to be while 2>1, as 2 is always going to be bigger than 1 this will go on forever! Scratch has a block called "*forever*" which we can use to do this without setting condition, you can find it in **Control.**

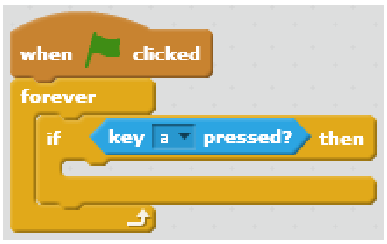
6. Put another "*When green flag is clicked*" code block into your script. Place one of these "forever" blocks under it.



7. We want the paddle to do different things depending on what key is pressed. To do this we'll need to use something called a *conditional.* This is a programming concept where your program splits depending on something. It's usually done with something called an *if, else statement.* Scratch has something like this and it's called an *"if then"* block. It checks some condition, and if it's true it runs the code inside the block. Place one of these inside your *"forever"* block.



8. We want to know whether the up key is being pressed, so we need to go to **Sensing** and place a *"key _ pressed"* block as our condition. We don't want to sense if a space has been pressed. We want to know if an "a" has been pressed. To do this, press the little down arrow and select "a".



9. Each time the "a" is pressed, we would like paddle 1 to move up. We will change our y position by increasing it by 5. To do this put a *"change y by _"* block inside the *"if _ then"* block and change the value to 5.

Test your code to see if it works. Press the green flag and press the "a" key on the keyboard.

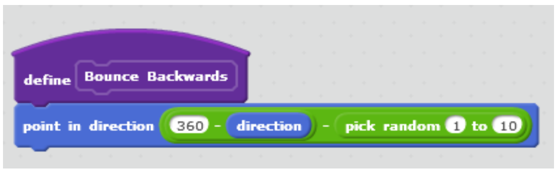
Exercises

- Make a code block that will change "y" by -5 when the "z" key is ever pressed during our game.
- Make Paddle2 work so that it begins at x=200 and y=0. Also, write scripts for this paddle so that the "up arrow" is up and "down arrow" is down.

  Make sure to test that your code works by moving both paddles.

Part 2 - Moving the ball

1. Click on the ball. We need to make our own block to get the ball code to work properly. Go to **More Blocks** and select "Make a Block", call it Bounce Backwards. This is a useful tool in Scratch which means we can make our own blocks and use them over and over again!
2. For this Bounce Backwards block we're going to change the direction the ball is facing so that when it hits the paddle it "bounces". To do this get a *"point in direction _"* block. Into this block place a "_ - _" block.
3. On the right-hand side place a *"pick random _ to _"* block and set the values to 1 and 10. On the left-hand side place another *"_ - _"* block. On the left-hand side put 360, on the right-hand side place a "*direction*" block. Your Bounce Backwards code block should look like this:

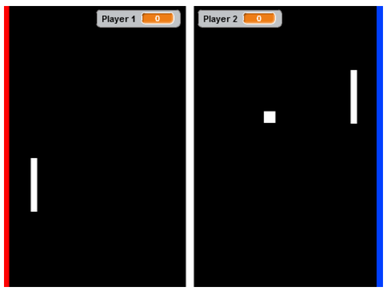


```
define Bounce Backwards
point in direction ((360 - direction) - pick random 1 to 10)
```

4. Get a *"When Flag is clicked"* block. When the game starts we want the ball to go to the centre. To do this, use a *"go to x : _ y: _"* block and set both values to 0 and put this under the flag piece.

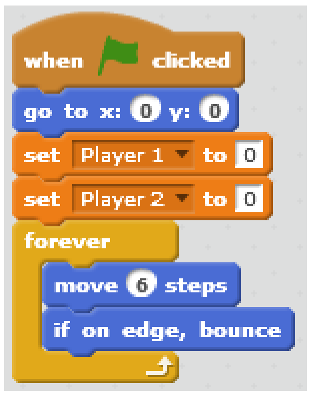
5. Pong wouldn't be much fun if you didn't know who was winning. To do this we will keep track of player one's and players two's score. We will store these scores in a *variable.* So go to **Data** and make two variables, call the 1st one "Player 1" and the 2nd one "Player 2".

6. You should notice that the scores appear in the top right corner. These might get in the way of seeing the ball. Move them to look like this:



7. When the game starts, we want to set these scores both to 0. Get two *"set _ to _"* blocks from **Data**, set them to Player1 and Player2 and set them both to 0.

8. For the duration of the game we want the ball to bounce around and hit off the paddles. To do this get a *"forever"* block. Now put a *"move _ steps"* block into the loop and change it to 6. Also from **Motion** get a *"if on edge, bounce"* block.

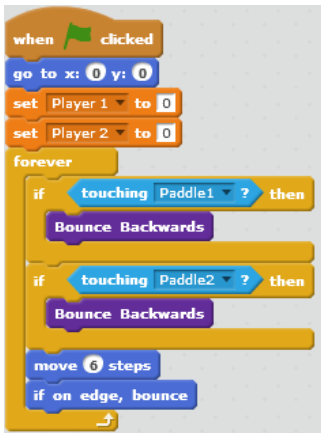


```
when [flag] clicked
go to x: 0 y: 0
set Player 1 to 0
set Player 2 to 0
forever
    move 6 steps
    if on edge, bounce
```

Give the game a go. You're very close to having a working version!

Now we just need the ball to hit off the paddles and we need to change the score when it hits the two sides.

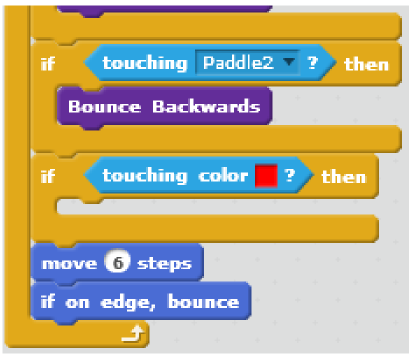
9. We will now check if we are touching a paddle. To do this get an "*if _ then*" block and place it in the loop before the move block. Into this we're going to place a *"touching _?"* block from and set it to Paddle1. Finally go to **More Blocks** and place a "*bounce backwards*" block into the "if" block.
10. You need to do the same thing for Paddle2. When you've done this your code will should like this:



```
when [flag] clicked
go to x: 0 y: 0
set Player 1 to 0
set Player 2 to 0
forever
    if touching Paddle1 ? then
        Bounce Backwards
    if touching Paddle2 ? then
        Bounce Backwards
    move 6 steps
    if on edge, bounce
```

Part 3 - Getting the score working

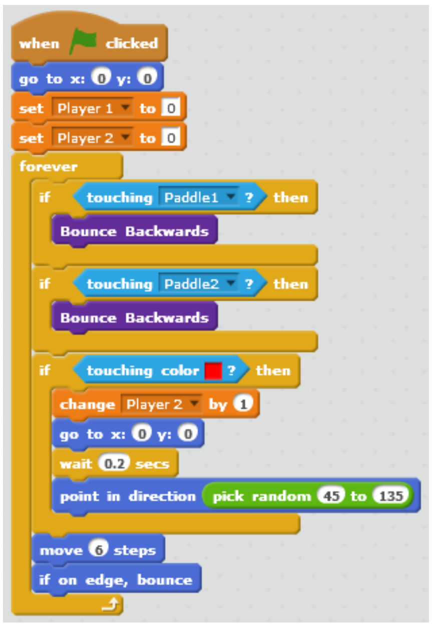
1. Now we need to get the score working. If the ball hits the red side then we will add one to player 2's score. To do this, get another "*if _ then*" block and place it after the last *"if _ then"* block but before the *"move _ steps"* block. Into this get a *"touching colour _ ?"* block. Set this to the same shade of red as the edge of the background using the dropper.



```
    if touching Paddle2 ? then
        Bounce Backwards
    if touching color 🟥 ? then
    move 6 steps
    if on edge, bounce
```

2. Now we need to change the score of Player 2 if the ball goes into the edge. Go to **Data** and get a *"change _ by"* block. Set it to Player 2 and the number to 1. Now get a *"go to x: _ y: _"* block and set both values 0. This will make sure the ball goes back to the middle after someone scores.

3. When someone scores we will give everyone a short pause so that they can regroup. To do this, go to **Control** and get a "*wait 1 secs*" block, put this in and change it to 0.2. To keep the game interesting, we will make the ball move towards player 2 who just scored, but in a random direction. To do this, go to get a *"point in direction _"* block. To this we must add a *"pick random _ to _"* block, set the first number to 45 and the second number to 135.



```
when 🏴 clicked
go to x: 0 y: 0
set Player 1 ▾ to 0
set Player 2 ▾ to 0
forever
    if < touching Paddle1 ▾ ? > then
        Bounce Backwards
    if < touching Paddle2 ▾ ? > then
        Bounce Backwards
    if < touching color 🟥 ? > then
        change Player 2 ▾ by 1
        go to x: 0 y: 0
        wait 0.2 secs
        point in direction pick random 45 to 135
    move 6 steps
    if on edge, bounce
```

Exercise

The final bit of code you need is code to change the score for Player 1. Place this code after the last "if" block but before the move block. Make it so that:

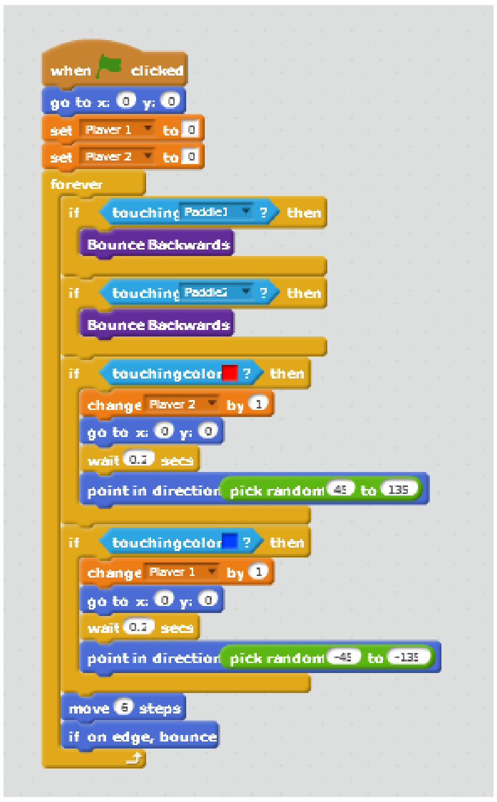
- If the ball is touching the same shade of blue that is on the edge
- Change the score of Player 1 by 1
- Make sure the ball goes back to the middle after someone scores
- Give everyone a short pause of 0.2s so that they can regroup
- Keep the game interesting by making the ball move towards player 1 who just scored, but in a random direction, the direction should be between -45 and -135

See the code at the end of this tutorial if you get stuck!

Now you have a working game! Challenge someone to play your game against you and see who will win!

To make the game harder, you could change the speed of the ball by changing the move block to a higher number.

The final code for the ball should look like this:

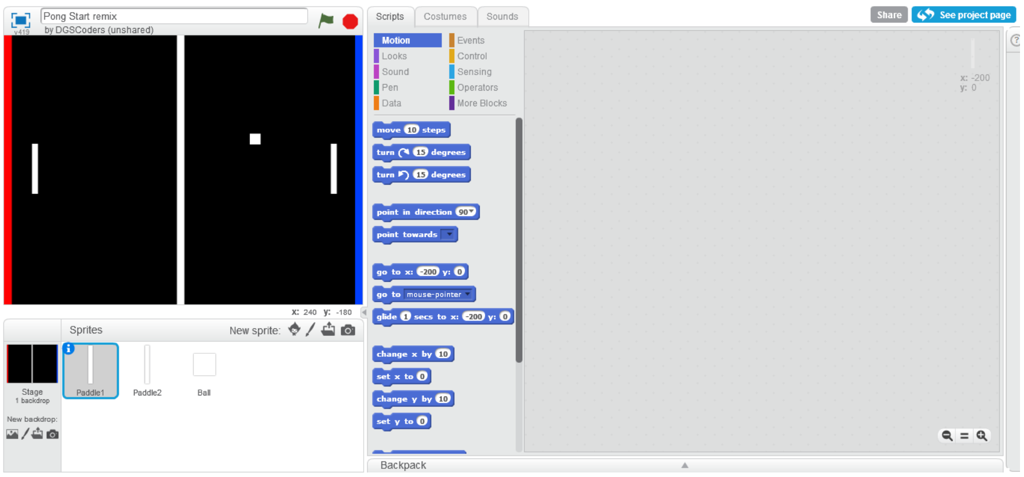


```
when 🏁 clicked
go to x: 0 y: 0
set Player 1 to 0
set Player 2 to 0
forever
    if touching Paddle1 ? then
        Bounce Backwards
    if touching Paddle2 ? then
        Bounce Backwards
    if touching color 🟥 ? then
        change Player 2 by 1
        go to x: 0 y: 0
        wait 0.2 secs
        point in direction pick random 45 to 135
    if touching color 🟦 ? then
        change Player 1 by 1
        go to x: 0 y: 0
        wait 0.2 secs
        point in direction pick random -45 to -135
    move 6 steps
    if on edge, bounce
```

Developed by James Lockwood
& Dr Aidan Mooney

Pong

Pong is a classic arcade game that is a bit like air hockey!



Before you get into coding you're going to have the make the stage and sprites like the picture above. It's a black background with a white line in the middle and a red strip on the left and a blue strip on the right side. You can make this by using the draw option on the backdrops.
You'll also need three Sprites. Two will look the same, thin white rectangles which will be our paddles and a white square which will be the ball. Again, use the draw option on the Sprites costumes. Name them Paddle1, Paddle2 and Ball.

Coding Paddle 1

What you'll need:

- We want our paddle to be in the correct position for a game.
- We want Paddle1 to move up using the "a" key and down using the "z" key. Paddle2 will move using the "up" and "down" arrows.
- We will need to check whether these keys are being pressed all the time otherwise it'll be a very slow game!
- We need to do different things depending on whether "a" or "z" is currently being pressed i.e. move up or down.

Hints:

Experiment with the x and y values for Paddle1 and Paddle2 until they're in a similar place to the picture above!

We want the paddle to be able to move very fluidly and forever. To do this we're going to use something called an *infinite loop.* An infinite loop is a loop, but it has a condition that is always true, usually this is a bad thing but it can be helpful! For example, you could set the condition to be while 2>1, as 2 is always going to be bigger than 1 this will go on forever! Scratch has a block called "*forever*" which we can use to do this without setting condition, you can find it in **Control.**

We have 2 possible options for each paddle. For Paddle1 either the player presses "a" or "z". If they hit the "a" key on the keyboard we want Paddle1 to move up. If they hit the "z" key, we want it to move down. To do this we need to use another programming concept called a *conditional.* This is where you can check something (a condition), in our case what key is pressed, and do something different depending on what happened. In Scratch this is an "*if _ then*" piece. You'll need two of these, one for a being pressed, another for z. You then need to make sure the paddle moves correctly!

Moving the ball

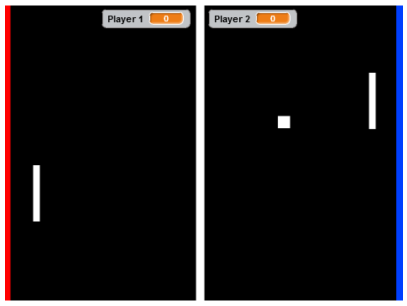
What you'll need:

- To set the balls initial position.
- Set the scores to zero, to keep track of the score you'll need to use *variables.*
- A way to make it bounce off the paddles.
- A way to increase each player's score when they score.
- A way to make the ball move.
- A way to make it bounce off the sides.

For the game to flow smoothly and everything to work it's important you put things in the correct order. Scratch *executes* the code from top to bottom. The order above is the best way to make it all work, so set the position at the top of the code block, and move it or bounce it off the side last.

Hints:

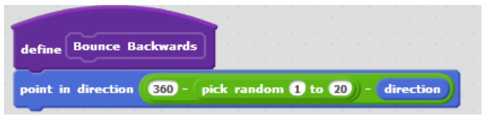
When you've made the variables, they'll appear on the screen. To make the game look better and so they're out of the way, you can drag them into place. Make it look something like the picture below:



For the duration of the game you want the ball to bounce around and hit off the paddles. To do this use a "*forever*" piece. All the code to check whether the ball is touching the paddles, to move, to bounce and to keep track of the score needs to go inside this "*forever*" block.

You need to check if we are touching a paddle. To do this get an "*if _ then*" piece and a "*touching _?*" piece from **Sensing**.

You need a way of getting the ball to bounce backwards off the paddles. Scratch can't do this by default so we need to use a handy tool where you can make your own blocks! Make sure you're in the Ball's code pane and go to the section called **More Blocks**. Click on this and click on "Make a Block". A screen should pop up, give the block the name "Bounce Backwards". You then want to make the ball move by making it point in a random direction. It should end up looking like the picture below.



```
define Bounce Backwards
point in direction ((360) - (pick random (1) to (20)) - (direction))
```

You must use a *conditional* (an "*if _ then*" block) to check whether the Ball is touching either Sprite, if it is the you want it to Bounce Backwards.

If the ball hits the red side, then we will add one to player 2's score. To do this you need to use another *conditional*. Instead of checking whether it's touching a Sprite this time though you want to check if it's touching a colour. This can be found in **Sensing**.

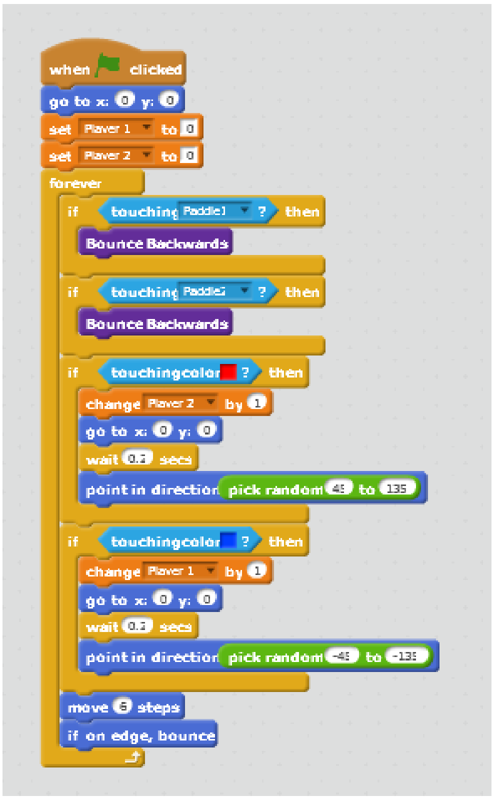
When someone scores, it might be good to give everyone a short pause so that they can regroup. To do this use a "*wait _ secs*" piece. You also want to put the ball back to the centre of the screen.

It might also be good to make the ball head in the direction of the person who's just scored using a "*point in direction _*" piece and a random number!

Scratch has a built-in way of getting things to bounce off edges. Have a look in **Motion** to find it!

If you're really stuck, look at the end of the tutorial for what the Ball code should look like.

Make sure that your game works; then challenge someone to play your game against you and see who will win! To make the game harder, you could change the speed of the ball by changing the move block to a higher number.



```
when 🏁 clicked
go to x: 0 y: 0
set Player 1 to 0
set Player 2 to 0
forever
    if touching Paddle1 ? then
        Bounce Backwards
    if touching Paddle2 ? then
        Bounce Backwards
    if touchingcolor 🟥 ? then
        change Player 2 by 1
        go to x: 0 y: 0
        wait 0.2 secs
        point in direction pick random 45 to 135
    if touchingcolor 🟦 ? then
        change Player 1 by 1
        go to x: 0 y: 0
        wait 0.2 secs
        point in direction pick random -45 to -135
    move 6 steps
    if on edge, bounce
```

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Polygon Drawer Teachers Guide

Strand: Introduction to Programming – Scratch Duration: 30 mins +

Computer Science Topics

Loops **(nested loops)**
User Input
Programming
Event handling

Computational Thinking Skills

Algorithm
Abstraction
Decomposition

Scratch tools

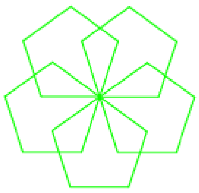
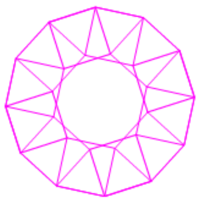
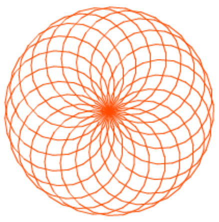
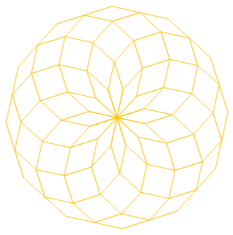
User input: Keyboard, **ask**
Variables: **Number, random number, mathematical operators**
Motion: Go to, move, **turn**
Appearance: **Pen**
Loop: **repeat, nested, variables**
Event handling: When flag clicked, broadcast, receive

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Polygon Drawer

This tutorial will show you how to make a program that draws different polygon shapes (squares, pentagons, hexagons etc.). We can then put this together to make some impressive looking designs like the one below!

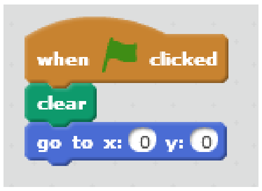





There's 3 versions of this program, each is a bit more complex than the other and allows you to make more complex designs easier!

Basic Version

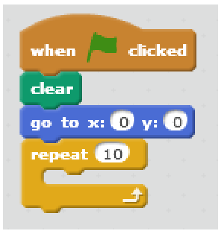
This version will just draw a shape once. This will all be "*hard-coded*" into your scripts. This means that to change the shape we want to draw; you must change the actual numbers in the code.

1. Grab a *"when green flag clicked"* block from **Events**.
2. To make sure our screen is clear when we start, get a *"clear"* block from **Pen**.
3. We then want to make sure our shapes are always drawn at the same spot, so take a *"go to x_ y_"* block from **Motion** and set it to [0, 0] (or something else if you'd prefer!).



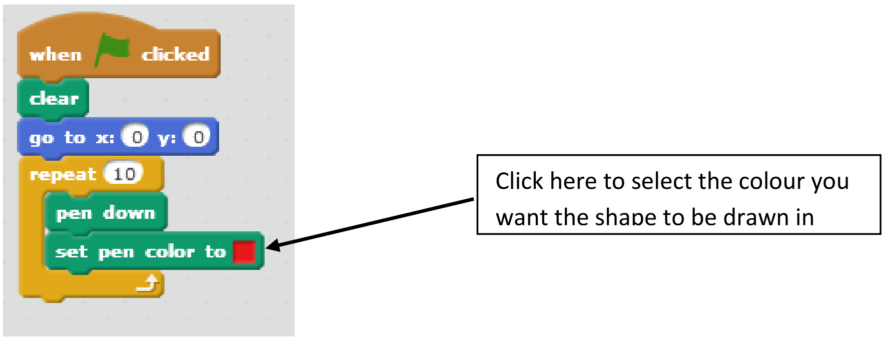
```
when [flag] clicked
clear
go to x: 0 y: 0
```

4. Now we want to make it draw our shapes multiple times, and we want it to draw them in a complete circle. To do this we'll need a *"repeat _"* block from **Control**. Put in any number you want as our condition.



```
when [flag] clicked
clear
go to x: 0 y: 0
repeat 10
```

5. Now we're going to draw our shapes. To do this we need to place a *"pen down"* block and a *"set pen color to _"* block from **Pen**. Put these inside your loop. Set the colour to whatever you want using the dropper. To use it, click on the little coloured square, then click on something on your screen that is the colour you want!



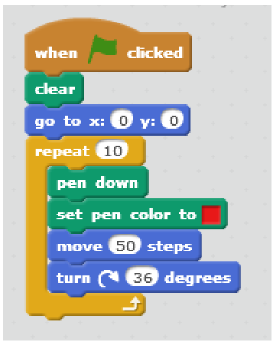
```
when [flag] clicked
clear
go to x: 0 y: 0
repeat 10
    pen down
    set pen color to 🟥
```

Click here to select the colour you want the shape to be drawn in

6. Now to be able to draw on the screen, we need a *"move _ steps"* block from **Motion**, set it to whatever you want.
7. To make it draw a complete Polygon (which are always closed shapes) we need to turn the drawer, so grab a *"turn _ degrees"* block from **Motion**. To make sure it's a complete circle you

need to set the degrees to match the number of times the loop is running (the number you've given in repeat). It's (degrees in a circle)/(number of sides).

8. This is the basic program! Try changing the number of sides of the shape (repeat block). Make sure you change the degrees to match.



```
when 🏴 clicked
clear
go to x: 0 y: 0
repeat 10
    pen down
    set pen color to 🟥
    move 50 steps
    turn ↻ 36 degrees
```

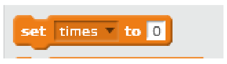
Medium Version

This version will draw a shape of a specific number of sides a specific number of times. This will make patterns like the ones shown above. As before this will be "hard-coded" into your scripts, but this time it'll be using something called a *variable.* A variable is a way of storing data/information in computer programs. They're called variables because they do just that, the vary/change! An example would be if you wanted to store someone's age, you would have something like age = 20. This means the **value** 20 is stored in a **variable** called age.

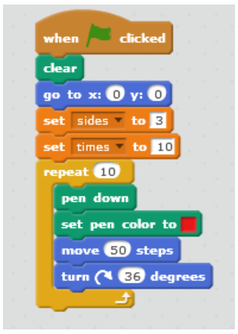
You'll need everything we've done so far and we'll add to it.

1. The first thing we're going to add is two variables. One of these will keep track of the number of sides our shape has (I called this sides). The other will keep track of the number of times you want to draw the shape (I called this times). To do this go to **Data** and click

2. You're going to have to do this twice to make each variable, make sure to give them significant names!
3. So now we have a variable, we're going to need to set it to some value. Take two *"set _ to _"* blocks (see below) from **Data**, using one for each of your variables, and have them set to whatever numbers you want.

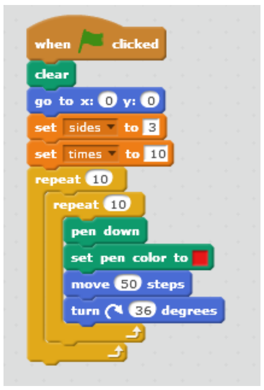


Your code should look something like this:



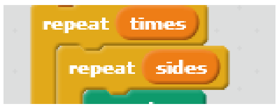
```
when [flag] clicked
clear
go to x: 0 y: 0
set sides to 3
set times to 10
repeat 10
    pen down
    set pen color to [red]
    move 50 steps
    turn (36) degrees
```

4. We now are going to use something called ***nested loops***, this is when you have one loop inside of the another.
5. Inside the current *"repeat _"* block, place another *"repeat _"* like in the picture here:



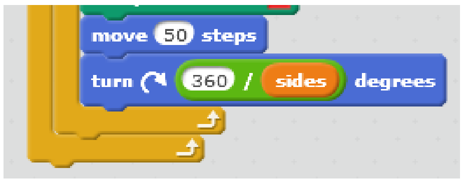
```
when ⚑ clicked
clear
go to x: 0 y: 0
set sides to 3
set times to 10
repeat 10
    repeat 10
        pen down
        set pen color to 🟥
        move 50 steps
        turn ↻ 36 degrees
```

6. Now we need to use our *variables* to set the stop points of the *loop*. The outer one will be the number of times we want to draw the shape, the inner one is going to be the number of sides. Drop the *"variable"* blocks for each loop like below.



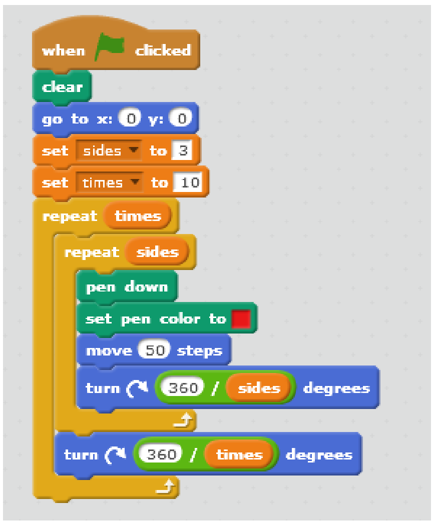
```
repeat times
    repeat sides
```

7. All the code that was inside the previous programs loop will be in the inner loop we've just made. We need to change something so that our code will draw the correct shapes based on the variables. To do this we need to use a *"_ / _"* block from **Operators**. We'll use the same formula as last time, on the left-hand side will be 360, on the right-hand side use the variable *"sides"*. Drop this block into the *"turn _ degrees"* block.



```
move 50 steps
turn ↻ (360 / sides) degrees
```

8. We're almost there! All we need to do now is make sure that we move the correct angle to draw the shapes in a nice circle. To do this we need another *"turn _ degrees"* block, place this just outside the inner loop (make sure it's inside the outer loop!). This time we're going to have turn 360/times degrees, so get another *"_ / _"* from **Operators** and *"times"* block from **Variables**. Your finished program should look like the picture below:



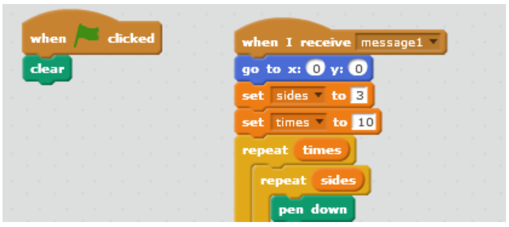
```
when ⚑ clicked
clear
go to x: 0 y: 0
set sides ▾ to 3
set times ▾ to 10
repeat times
    repeat sides
        pen down
        set pen color to 🟥
        move 50 steps
        turn ↻ (360 / sides) degrees
    turn ↻ (360 / times) degrees
```

Advanced version

This time we're going to get the user (the person playing with the polygon drawer), to give the values they want for the number of sides and times.

You'll need everything from the previous two parts to be able to finish off the game.

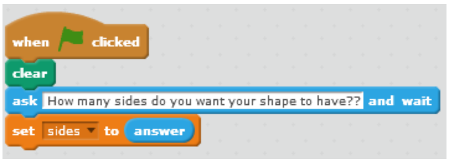
1. First off, we're going to replace the *"when green flag clicked"* block in our script with a *"when I receive _"* block from **Events**. Call your message whatever you want, I left it as the default message1. This block works similarly to the flag block, but instead of running the code when the flag is clicked it only runs the code when it receives the broadcasted message given.
2. We're going to have a second script, to start with we need a *"when green flag clicked"* block. Take the *"clear"* block from the other script and place it after the flag block.



```
when green flag clicked
clear
```

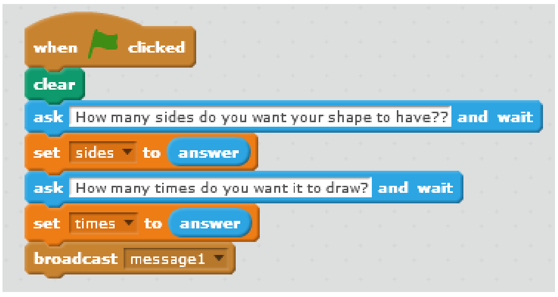
```
when I receive message1
go to x: 0 y: 0
set sides to 3
set times to 10
repeat times
    repeat sides
        pen down
```

3. This version of the program will ask a user (person) to give the number of sides and the number of times to draw the shape. Take out the *"set _ to _"* blocks from the first script, we'll need them in a second so don't delete them!
4. We're going to write up a message to the user so they know what to input. Get an *"ask _ and wait"* block from **Sensing** and place it under the *"when green flag clicked"* block. In this put the question: "How many sides do you want the shape to have?".
5. We now need to set sides equal to what the user types. For this take the *"set _ to _"* block we used previously and place it under the *"ask _ and wait"* block. Now instead of having a number, go to **Sensing** and take an "*answer*" block and place it into the *"set _ to _"* block.



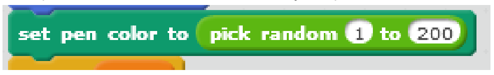
```
when green flag clicked
clear
ask How many sides do you want your shape to have?? and wait
set sides to answer
```

6. Repeat this step for the "*times"* variable, make sure you change the question!
7. Remember that *"when I receive _"* block we used earlier? Well we need to broadcast a message to it to make that code run! Go to events and get a *"broadcast _"* block and make sure it's message name is the same as the *"when I receive"* block. Place it after the two questions.



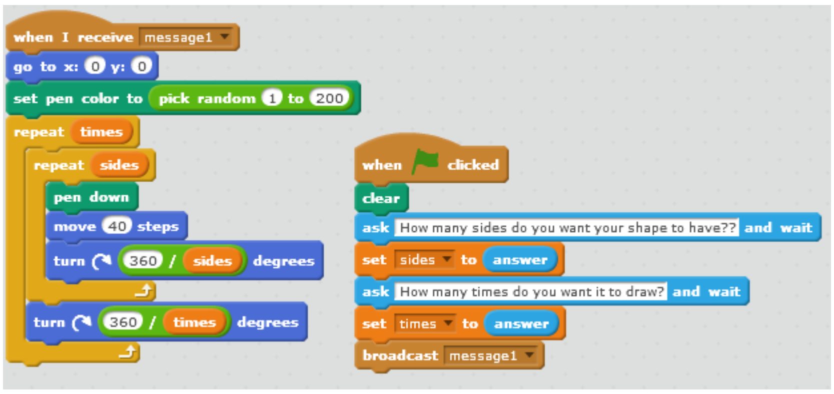
```
when [flag] clicked
clear
ask [How many sides do you want your shape to have??] and wait
set [sides] to (answer)
ask [How many times do you want it to draw?] and wait
set [times] to (answer)
broadcast [message1]
```

8. One last step you can do is to make the pen colour a random colour. To do this go to **Pen** and get a *"set pen color to 0"* block. This block allows us to pick a colour based on numbers rather than using the dropper. Now go to **Operators** and get a *"pick random _ to _"* block and place it into the *"set pen color to _"* block. Mess around with the numbers and see what different varieties of colours you get! (I did 1 to 20 as an example!).



```
set pen color to (pick random (1) to (200))
```

9. Your completed program should look similar to the picture below:



```
when I receive [message1]
go to x: (0) y: (0)
set pen color to (pick random (1) to (200))
repeat (times)
    repeat (sides)
        pen down
        move (40) steps
        turn ↻ ((360) / (sides)) degrees
    end
    turn ↻ ((360) / (times)) degrees
end
```

```
when [flag] clicked
clear
ask [How many sides do you want your shape to have??] and wait
set [sides] to (answer)
ask [How many times do you want it to draw?] and wait
set [times] to (answer)
broadcast [message1]
```

Other ways you could advance it:

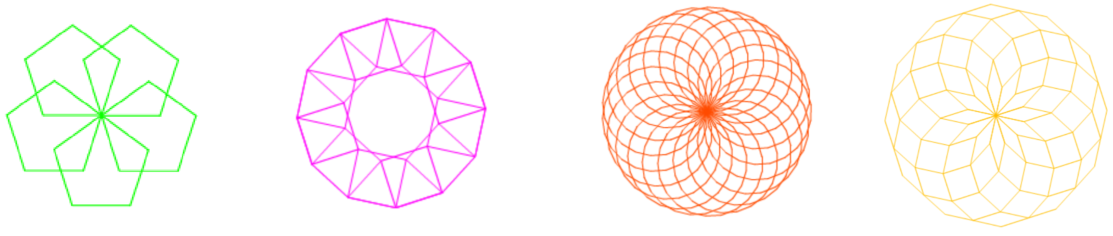
- Let the user pick the colour of the pen
- Change the colour for each side (this will use random numbers in the Operators section)
- Remove the cat so that it looks like nothing is drawing the shapes

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Polygon Drawer

This tutorial will show you how to make a program that draws different polygon shapes (squares, pentagons, hexagons etc.). We can then put this together to make some really impressive looking designs like the one below!



There's 3 versions of this program, each is a bit more complex than the other and allows you to make more complex designs easier!

Unlike the previous tutorial this won't give you all the steps you need, but hints and pointers to how you can make it.

Basic Version

This version will just draw a shape once. This will all be "*hard-coded*" into your scripts. This means that to change the shape we want to draw; you must change the actual numbers in the code.

What you'll need:

- A way to clear the screen so that you can draw a new design each time.
- A way to set the initial position of the Sprite.
- A way of doing something multiple times (so you can draw the lines multiple times to make a square/pentagon etc.). This is a loop!
- A way to draw. Scratch has a whole section given to this called **Pen**.
- A way of moving the sprite so that it draws a specific shape, such as a square.

Hints:

Make sure you don't draw the shapes too big, Scratch won't let you go off the screen!

To figure out how to move it, think of how you draw a square, move forward X steps, turn Y degrees, repeat X times. You can then change these to make it work for different shapes!

To do the maths part you'll need to look in the **Operators** section.

You can change the colour of the pen, just in case you want to!

To make the program look better you can remove the picture of the cat so it looks like "nothing" is drawing the shape. Go to the costumes of the Sprite to change it or get rid of it.

Medium Version

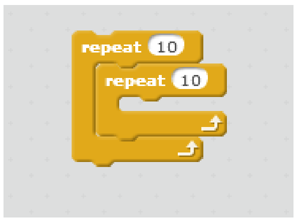
This version will draw a shape of a specific number of sides a specific number of times. This will make patterns like the ones shown above. As before this will be "hard-coded" into your scripts, but this time it'll be using something called a *variable.* A variable is a way of storing data/information in computer programs. They're called variables because they do just that, the vary/change! An example would be if you wanted to store someone's age, you would have something like age = 20. This means the **value** 20 is stored in a **variable** called age.

What you'll need:

- Everything we've done so far
- Two *variables*, one to hold the number of sides and the other to hold the number of times you want to draw the shape. Scratch has a whole section for this called **Data**. You must then set these variables equal to the number you want.
- A way of moving the Sprite so that it draws a specific shape, such as a square. This won't be the same as the last version but it's close.
- A way of doing something multiple times so you can draw a shape as per the previous version. This time however, you'll want another loop to pick how many times we want to draw that shape.
- A way of moving the Sprite after its drawn one copy of the shape. This is so it doesn't just draw the shape repeatedly.

Hints:

You're going to have two loops, one inside the other like the picture below. Instead of 10 and 10 though, you're going to use your two variables to limit the number of times each loop runs.


```
repeat 10
  repeat 10
```

To make a variable, go to **Data** and click on "Make a Variable".

To make it easier for you I would call one variable something like "*sides***"** and one something like "*times***".** This is also good practice to name your variables something that makes sense!

Make sure you set the variables to be whatever numbers you want **before** your loops.

To get the Sprite to draw the same shape in a slightly different place you're going to have to put a turn block after the inner loop.

Try and figure out how you can get it to draw them in a perfect circle like the pictures above. It'll be similar, but different to how you got the initial shape to be drawn. Think about how the angles in a circle!

Advanced version

This time we're going to get the user (the person playing with the polygon drawer), to give the values they want for the number of sides and times.

What you'll need:

- Everything we've done so far.
- A way to take input from the user and save these as our variables. Scratch has a whole section of this called **Sensing**.
- A way of telling the program that the user has inputted the values and so it can draw.
- A way to clear the screen so that you can draw a new design each time.
- A way to set the initial position of the Sprite.

Hints:

Instead of setting the variables equal to a number we're going to set them this time equal to a variable. Scratch has a variable it uses to get *user input* which is called "*answer*" and can be found in the Sensing section.

This time around our loops as well are going to use this variable. So instead of using a number in the loop we're going to put a variable block. These turn up in the **Data** section once you've made a variable.

To figure out how to tell the computer it can draw, think about how we broadcasted a message in the Cat & Mouse game to tell the Cat it had been caught. Consider the **Events** section if you're stuck!

For this program, you're going to want to use two blocks of code rather than one!

Other ways you could advance it:

Let the user pick the colour of the pen

Change the colour for each side/each shape (this will use random numbers in the **Operations** section).

Developed by James Lockwood
& Dr Aidan Mooney

Guessing Game Teachers Guide

Strand: Introduction to Programming – Scratch   Duration: 30 mins +

Computer Science Topics

Loops
User Input
Programming
Event handling
Conditional **(if then, else)**

Computational Thinking Skills

Algorithm
Abstraction
Decomposition

Scratch tools

User input: Keyboard, ask
Variables: Number, random number, mathematical operators, **Boolean operators (and, or)**
Appearance: Pen, say
Loop: repeat, nested, variables
Conditional: if then, **if then else**
Event handling: When flag clicked, broadcast **(multiple)**, **receive (multiple)**

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Guessing Game

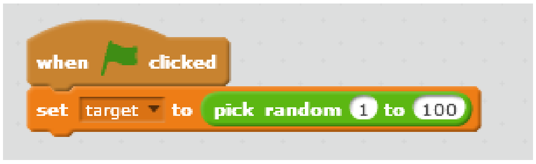
For this project, we're going to make a game in which you can guess a number the computer has picked! You may have done this during the searching and sorting class, it's based on a searching algorithm called binary search!

Like the Polygon drawer this game has a few iterations where it gets more complicated.

Basic Version

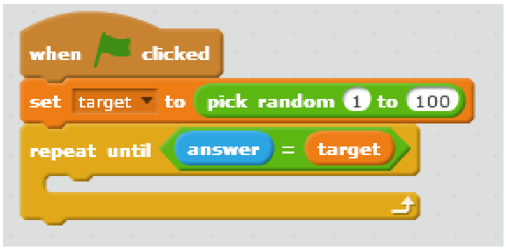
This first version will play the game through one time and get the user to input a guess between 1-100 until it reaches a randomly selected number!

1. Our game will begin when we click the green flag, get one of these blocks from **Events**.
2. The first thing we need to do is set up our target number. To do this we need to make a *variable*, so go to **Data** and do this, I called mine target. Now take a "*set _ to _*" block and set the it to your variable.
3. For the value, we're going to make it a random number between 1 and 100, so go to **Operators** and place a *"pick random _ to _"* block into the "*set*" block. Change the values to be 1 to 100.



```
when [flag] clicked
set target to (pick random 1 to 100)
```

4. Now we want the game to keep letting us guess until we have guessed the correct number. To do this we'll need a *loop*, so get a *"repeat until _"* block from **Control.**
5. For the condition, we'll need to use the "*answer*" block from **Sensing** and our target variable. We'll place these into the left and right-hand side of an *"_ = _"* block from **Operators**.

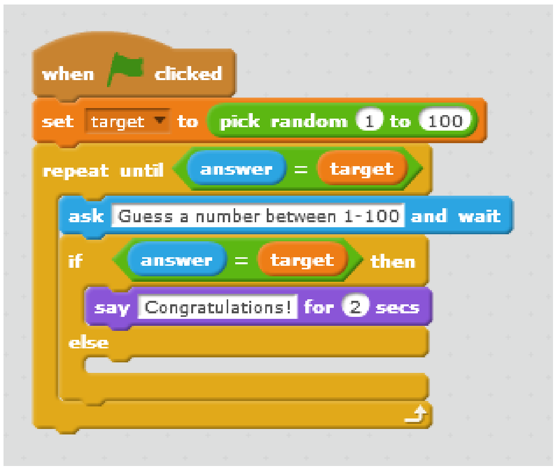


```
when [flag] clicked
set target to (pick random 1 to 100)
repeat until <(answer) = (target)>
```

6. Now we need to give the user an instruction, so get an *"ask _ and wait"* block from **Sensing** and give it a helpful instruction like "Guess a number between 1-100".
7. Next we need to check for the 3 possibilities, either their guess is the target, guess is less than the target or it's greater than. To do that we'll use *conditionals*, in Scratch's case an *"if _ then,*

else" block. For the condition, you are going to put the same blocks as our loop (the *"repeat until _ "* block).

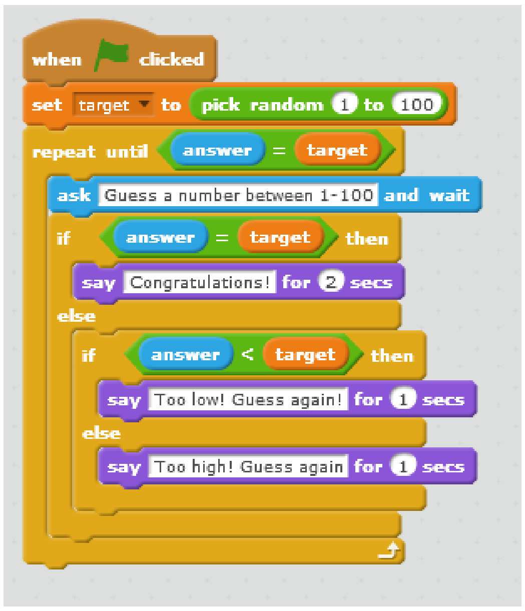
8. If their answer equals target then we should congratulate them! Get a "*say _ for _ secs*" block from **Looks** and put in a well-done message for 2 seconds.



```
when [flag] clicked
set target to (pick random 1 to 100)
repeat until <answer = target>
    ask Guess a number between 1-100 and wait
    if <answer = target> then
        say Congratulations! for 2 secs
    else
```

All that's left to do now is to program what to do if they don't get the correct answer. We'll give them a hint by telling the user whether their guess was too low or high, that way you can play it well if you know the binary search algorithm!

9. Get another *"if _ then, else*" block and place it inside the "*else*" part of our first "*if _ then, else*" block.
10. For our condition, this time we're going to say if answer < target. Grab a *"_ < _"* block and put an answer and target block on either side.
11. If their guess is less than the target number then we should tell them it's too low, so they should guess again. Do this by using another "*say _ for _ secs*" block.
12. If their guess wasn't the target and wasn't less than it, then the only other option is they went too high! Inside the final "*else*" part give the user a message telling them their guess was too high!



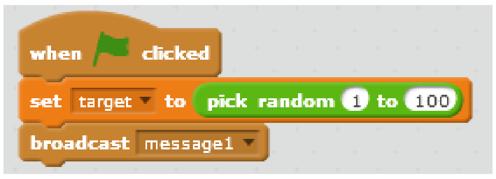
```
when [flag] clicked
set target to pick random 1 to 100
repeat until answer = target
    ask Guess a number between 1-100 and wait
    if answer = target then
        say Congratulations! for 2 secs
    else
        if answer < target then
            say Too low! Guess again! for 1 secs
        else
            say Too high! Guess again for 1 secs
```

Now you have a working game, but we’ll make it even better (and harder!) over the next two phases!

Medium version

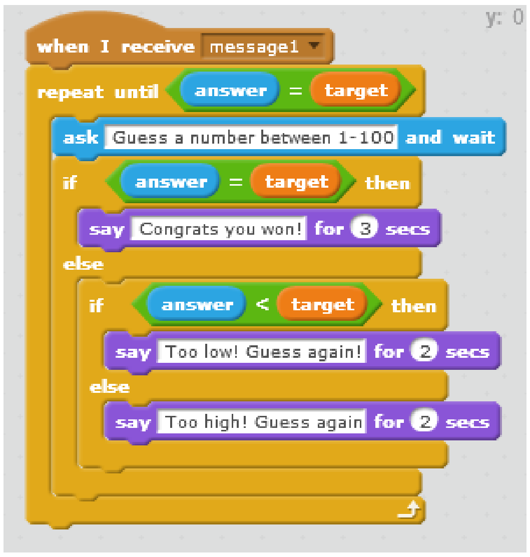
This version of the game will be very similar, but it'll let the player play again if they want without having to press the green flag!

1. The first step for this is using a broadcast message. This is because we can't make the flag be clicked again so the message will be sent each time the user says "yes" to wanting to play the game again.
2. Get everything below the "*set _ to _*" block and put it to one side, we'll need it in a minute! All we need to do now with our "when green flag clicked" script is to add a third block. This will be the "*broadcast _*" block from **Events.** Rename your message whatever you want.



```
when clicked
set target to pick random 1 to 100
broadcast message1
```

3. Now we need to tell the program when to run our second script. We want to do this when it receives the broadcasted message. Go to **Events** to get a "*when I receive _*" block. Your script should look like this now.

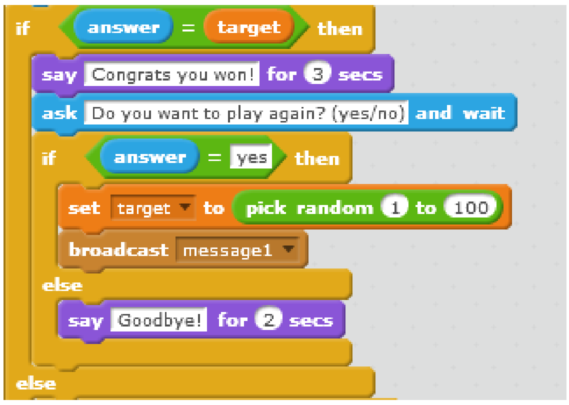


```
when I receive message1
repeat until answer = target
    ask Guess a number between 1-100 and wait
    if answer = target then
        say Congrats you won! for 3 secs
    else
        if answer < target then
            say Too low! Guess again! for 2 secs
        else
            say Too high! Guess again for 2 secs
```

4. Now all we need to do is ask the user if they want to play again. If they do we should reset the target to some new number and restart, if they don't we should say goodbye!

All the code we're going to do here goes after the "*say 'Congrats you won!' for 3 secs*" block in the picture above.

5. Firstly, we need to ask them to do they want to play again. Get an "*ask _ and wait*" block from **Sensing** and give a sensible message, something like "Do you want to play again? (yes/no)". Place this block after the congratulations message.
6. Now we need another *"if _ then, else"* block. Place this under the above question.
7. For our condition, it's whether the user answered yes, so get an *"_ = _"* block, an "*answer*" block and type yes into the right-hand side of the *"="* block.
8. All we need to do now is stick a *"say _ for _ secs"* block from **Looks** in the "*else*" part, give it an appropriate message like "Goodbye".
9. Then in the "*if*" part we place the same "*set _ to _"* block we have in our other script. After this place a "*broadcast _"* block, make sure the message name is the same as the one you used previously.



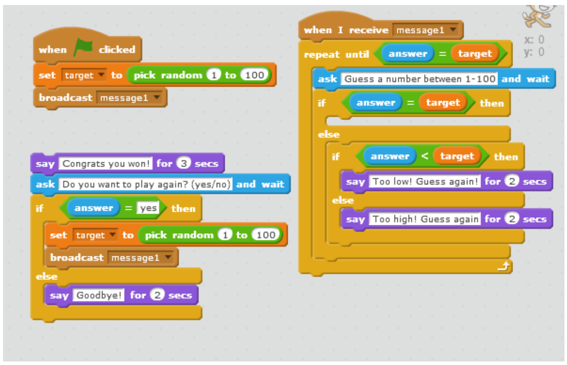
```
if answer = target then
  say Congrats you won! for 3 secs
  ask Do you want to play again? (yes/no) and wait
  if answer = yes then
    set target to pick random 1 to 100
    broadcast message1
  else
    say Goodbye! for 2 secs

else
```

Now your game is even better, but there's more to come! We'll make the game easier or harder and keep a track of the number of guesses they've made!

Advanced version

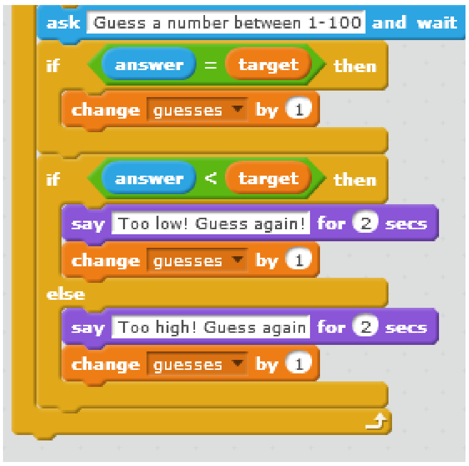
In this game we'll give the user the option of playing an easy/medium or hard version of the game. We'll also track the number of guesses they make and tell them if they could have done better! This will be the most complex Scratch project we've made so far, we'll have 6 different scripts! But don't worry, 3 of them are almost identical and you've already made most of them!

1. The first thing we need to do is make a new *variable* to hold the number of guesses the player has used. Go to **Data** and do this, I called mine guesses.
2. Now take everything below in the first "*if*" section and put it to one side, we'll need it later in a different spot!



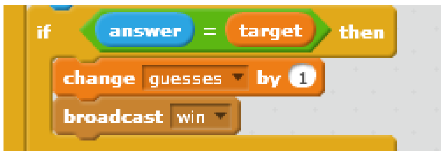
```
when [flag] clicked
set target to (pick random 1 to 100)
broadcast message1

say Congrats you won! for 3 secs
ask Do you want to play again? (yes/no) and wait
if answer = yes then
    set target to (pick random 1 to 100)
    broadcast message1
else
    say Goodbye! for 2 secs

when I receive message1
repeat until answer = target
    ask Guess a number between 1-100 and wait
    if answer = target then
    else
        if answer < target then
            say Too low! Guess again! for 2 secs
        else
            say Too high! Guess again for 2 secs
```

3. We're now going to place a "*change _ by _*" block into the two "*if*" sections as well as the "*else*" section. Set it to change your guesses variable by 1. This is because each time this code executes the user will have guessed once more! The code should look like this:



```
ask Guess a number between 1-100 and wait
if answer = target then
    change guesses by 1
if answer < target then
    say Too low! Guess again! for 2 secs
    change guesses by 1
else
    say Too high! Guess again for 2 secs
    change guesses by 1
```

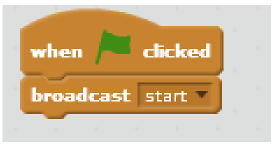
4. Now you need to set up another broadcast message which says we've won. I called mine "win". To do this go to **Events** and drag a "*broadcast _*" block into the first "*if*" section. Change the message using the drop-down arrow and call it win. We're going to need a few more messages but for now place this block into the script here:



```
if answer = target then
    change guesses by 1
    broadcast win
```

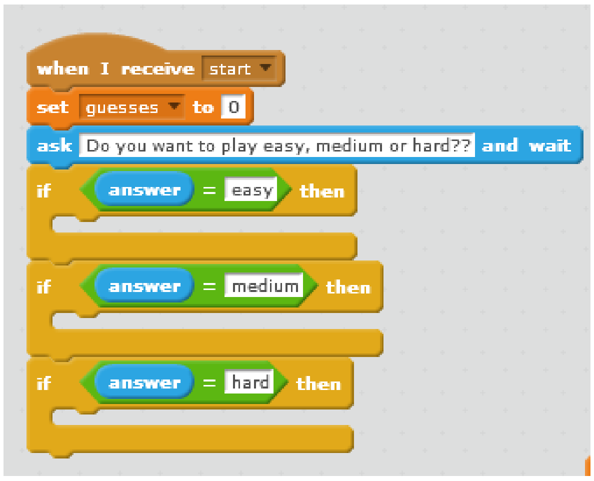
We'll come back to what our program should do when it receives win, for now let's make the beginning of the game, when players choose whether they want to play an easy, medium or hard game.

5. To do this we'll need to change our "when green flag clicked" script so that it just has one block below it, so remove the *"set _ to _"* block and bin it. I also changed my broadcast message to a new one called "start", just to make it easier to see what's happening!



```
when clicked
broadcast start
```

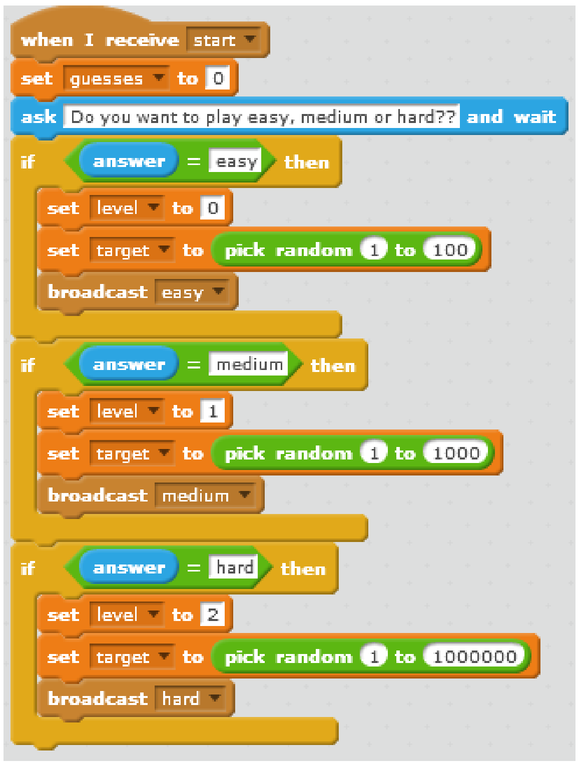
6. Now we're going to use the "start" message (or whatever yours is called), to let the user choose their game level. Get a "*when I receive _*" block from **Events** and change it to match your message name.
7. After this get a "*set _ to _*" piece and change the left and side to guesses and the right-hand side to 0. This will reset the players guess count each game.
8. Next, we'll ask the user if they want to play an easy, medium or hard game. To do this get a "*ask _ and wait*" block from **Sensing**, give an appropriate message like "Do you want to play easy, medium or hard?".
9. We now need to do different things based on their answer, so this is a *selection* or *conditional*, so get three "*if _ then*" blocks from **Control.** The conditions of each are similar, they should say answer = "easy", answer = "medium" and answer = "hard". This uses an "*answer*" block from **Sensing** and an *"_ = _"* block from **Operators.**



```
when I receive start
set guesses to 0
ask Do you want to play easy, medium or hard?? and wait
if answer = easy then

if answer = medium then

if answer = hard then

```

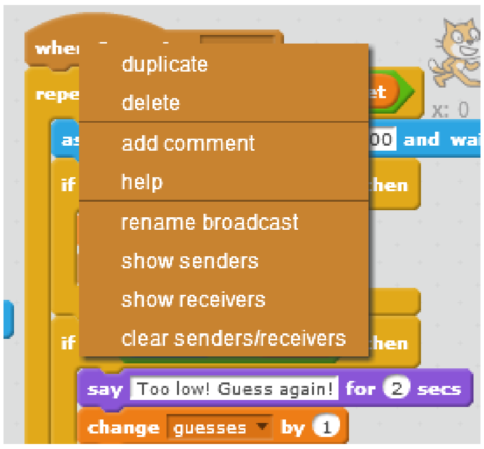
10. The next step is to set up each level. Each will be similar, we need to set the level to 0, 1 or 2 (easy, medium, hard), set the target to a random number and then broadcast a message.

11. To do this we need to make a new variable called level, do this in **Data.** Place a "*set _ to _*" into each "*if _ then*" block, for the easy set level to 0, for medium set it to 1 and set it to 2 for hard.
12. Next place another *"set _ to _"* block into each, this time setting target to a random number, just like we did in the previous sections. For easy you'll set target to a random number between 1-100, for medium a number between 1-1000 and for hard a number between 1-1000000!
13. The final step in this part is to broadcast a message to say which version of the game to play. For this you need to place a "*broadcast _*" block into each "*if _ then*" block. Make three new messages, the easy one should be called easy, the medium one medium and the hard one hard! Hopefully this script looks like the picture below!

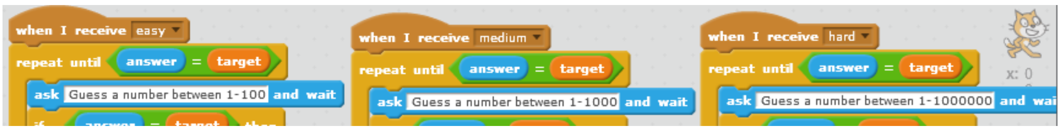


```
when I receive start
set guesses to 0
ask Do you want to play easy, medium or hard?? and wait
if answer = easy then
    set level to 0
    set target to pick random 1 to 100
    broadcast easy
if answer = medium then
    set level to 1
    set target to pick random 1 to 1000
    broadcast medium
if answer = hard then
    set level to 2
    set target to pick random 1 to 1000000
    broadcast hard
```

14. Now we need to make the code for the easy, medium and hard games. Thankfully we've already done this!
15. We'll duplicate our first script, the one that tells the user whether their guess is too high or low. To do this right click on the top block and you should get the menu below.



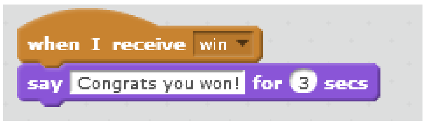
16. Click duplicate and a copy of the script should appear. Drag it out of the way and do the same thing again.
17. Now all we need to do is to change each of them so that they match our game plan. Change the "*when I receive _*" blocks to match the easy, medium and hard messages.
18. The other part we need to change is the text to guess a number, keep the easy one to "Guess a number between 1-100" but change the medium and hard one to say, "Guess a number between 1-1000" and "1-1000000". They should look like the image below.


when I receive easy
repeat until answer = target
ask Guess a number between 1-100 and wait
when I receive medium
repeat until answer = target
ask Guess a number between 1-1000 and wait
when I receive hard
repeat until answer = target
ask Guess a number between 1-1000000 and wai

Almost there! All we need to do now is congratulate them if they win and ask them if they want to play again! We're going to change this a bit and tell them if they could have guessed the number quicker, based on using the binary search algorithm!

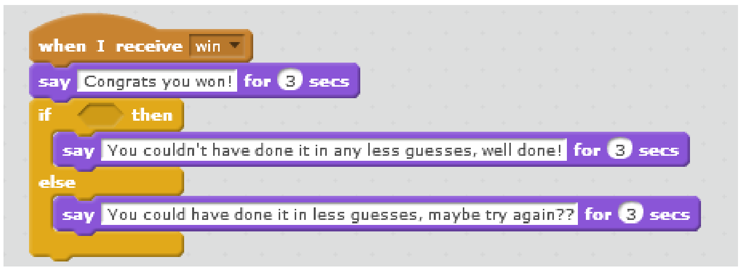
This will involve a complicated looking block using "*_ and _*" and "*_ or _*" blocks from **Operators**, but we'll get there!

19. To start with we need to use that "win" broadcast we made way back at the start of this Advanced tutorial.
20. Get a "*when I receive _*" block and set it to the "win" message. Below this block put the congratulations message you've hopefully still got, or make a new one using a "*say _ for _ secs*" block.



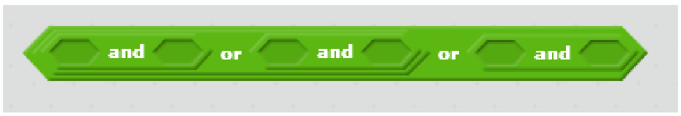
21. Now we're going to check if they could have guessed the number with less tries! To do this get a "*if _ then, else*" block from **Control**.

22. Into both sections place a “*say _ for _ secs*” piece. In the “*if*” part say something like “You couldn’t have done it in any less guesses, well done!”. In the “*else*” part, say something like” You could have done it in less guesses, maybe try again??”.



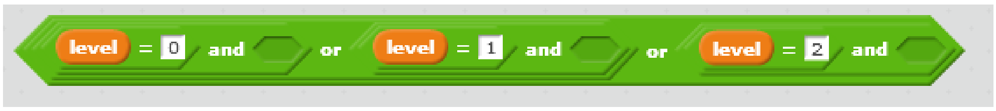
```
when I receive win
say Congrats you won! for 3 secs
if < > then
    say You couldn't have done it in any less guesses, well done! for 3 secs
else
    say You could have done it in less guesses, maybe try again?? for 3 secs
```

23. Now we come to the check. Based on the binary search algorithm and using a mathematical method called logs, we can know the guaranteed number of guesses that the player should be able to get the target in. For level 0 (easy game) it’s 8 or less (roughly 7.1), for level 1 it’s 10 or less (roughly 9.) and for level 2 it’s less than 20 (roughly 19).
24. To start the block, we need an *“_ or _”* block from Operators. Into the left-hand side of this place another *“_ or _”* block. Now into all three gaps place an *“_ and _”* block like the picture below.



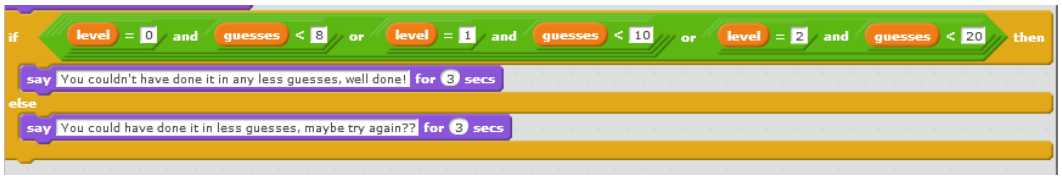
```
< < > and < > > or < < > and < > > or < < > and < > >
```

25. Into the left-hand side of each *“_ and _”* block, we’re going to place an *“_ = _”* block. Into the left-hand side of each of these *“_ = _”* blocks place a “*level*” block. On the right-hand side type a 0 in the left-most one, a 1 into the middle and a 2 into the right-most.



```
< (level) = 0 and < > > or < (level) = 1 and < > > or < (level) = 2 and < > >
```

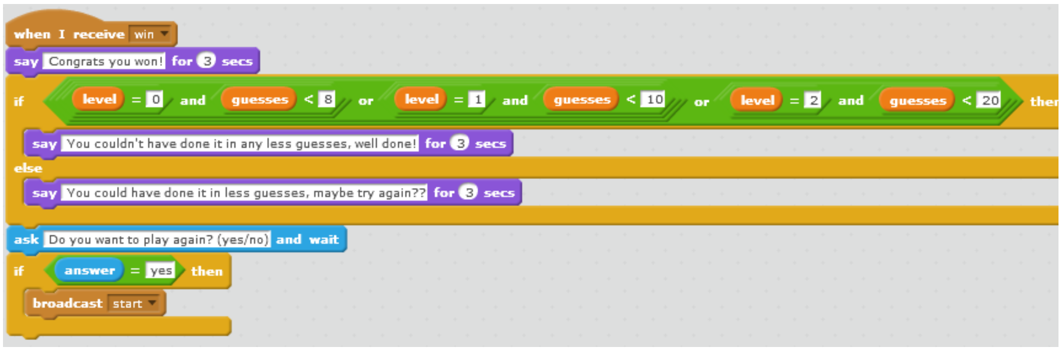
26. Into the three empty slots place a *“_ < _”* block. Into the left-hand side of each place a “*guesses*” block. Into the right-hand side type 8, 10 and 20 from left to right.
27. Your completed *“if _ then, else”* block should look like this:



```
if < (level) = 0 and (guesses) < 8 or (level) = 1 and (guesses) < 10 or (level) = 2 and (guesses) < 20 > then
    say You couldn't have done it in any less guesses, well done! for 3 secs
else
    say You could have done it in less guesses, maybe try again?? for 3 secs
```

28. The last step is to ask the user whether they want to play again.
29. To do this use the same blocks as in the previous versions of the game, but this time change the broadcast message name to the one that starts the game (mine was called start). If you can’t figure out which one this is, it’s the message that is broadcast from the “*when green flag clicked*” block.

Hopefully your code looks like below:

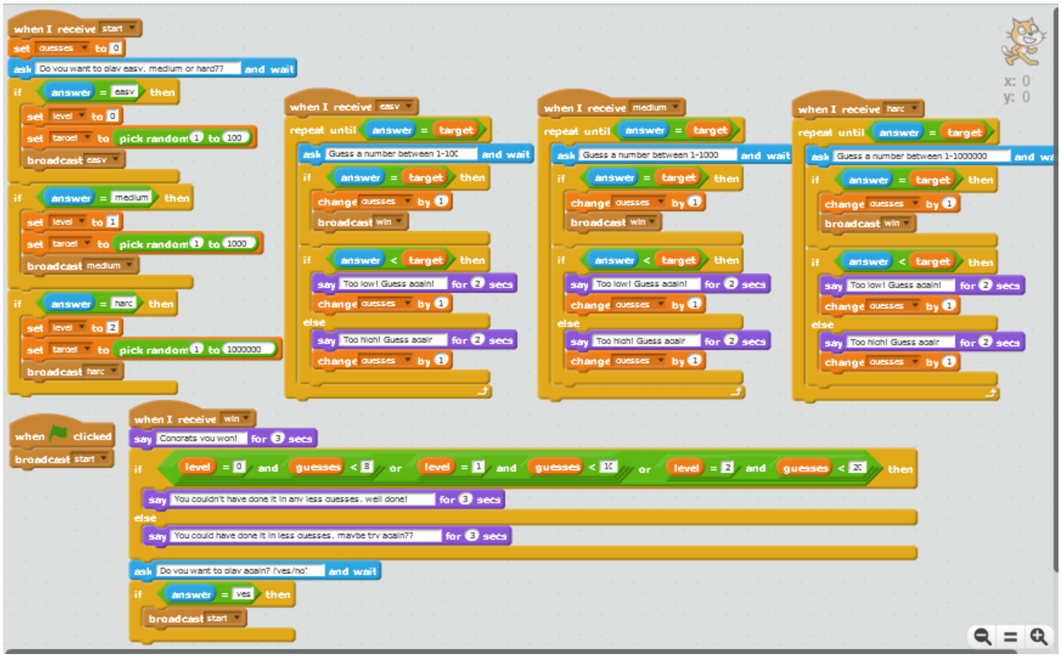


```
when I receive win
say Congrats you won! for 3 secs
if < <<(level) = 0> and <(guesses) < 8>> or <<(level) = 1> and <(guesses) < 10>> or <<(level) = 2> and <(guesses) < 20>> the
    say You couldn't have done it in any less guesses, well done! for 3 secs
else
    say You could have done it in less guesses, maybe try again?? for 3 secs
ask Do you want to play again? (yes/no) and wait
if <(answer) = yes> then
    broadcast start
```

Congratulations! You've made a pretty complex guessing game! Feel free to mess around with it and make it look nicer!

Hopefully you now have all the tools and ideas to make your own creations in Scratch. You'll find loads more ideas on the Scratch website and more tutorials will be available too if you want to do more!

This is all the code for the advanced guessing game program:



```
when I receive start
set guesses to 0
ask Do you want to play easy, medium or hard?? and wait
if <(answer) = easy> then
    set level to 0
    set target to (pick random 1 to 100)
    broadcast easy
if <(answer) = medium> then
    set level to 1
    set target to (pick random 1 to 1000)
    broadcast medium
if <(answer) = hard> then
    set level to 2
    set target to (pick random 1 to 1000000)
    broadcast hard
```

```
when I receive easy
repeat until <(answer) = (target)>
    ask Guess a number between 1-100 and wait
    if <(answer) = (target)> then
        change guesses by 1
        broadcast win
    if <(answer) < (target)> then
        say Too low! Guess again! for 2 secs
        change guesses by 1
    else
        say Too high! Guess agair for 2 secs
        change guesses by 1
```

```
when I receive medium
repeat until <(answer) = (target)>
    ask Guess a number between 1-1000 and wait
    if <(answer) = (target)> then
        change guesses by 1
        broadcast win
    if <(answer) < (target)> then
        say Too low! Guess again! for 2 secs
        change guesses by 1
    else
        say Too high! Guess agair for 2 secs
        change guesses by 1
```

```
when I receive hard
repeat until <(answer) = (target)>
    ask Guess a number between 1-1000000 and wa
    if <(answer) = (target)> then
        change guesses by 1
        broadcast win
    if <(answer) < (target)> then
        say Too low! Guess again! for 2 secs
        change guesses by 1
    else
        say Too high! Guess agair for 2 secs
        change guesses by 1
```

```
when green flag clicked
broadcast start
```

```
when I receive win
say Congrats you won! for 3 secs
if < <<(level) = 0> and <(guesses) < 8>> or <<(level) = 1> and <(guesses) < 10>> or <<(level) = 2> and <(guesses) < 20>> > then
    say You couldn't have done it in any less guesses, well done! for 3 secs
else
    say You could have done it in less guesses, maybe try again?? for 3 secs
ask Do you want to play again? (yes/no) and wait
if <(answer) = yes> then
    broadcast start
```

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Guessing Game

For this project, we're going to make a game in which you can guess a number the computer has picked! You may have done this during the searching and sorting class, it uses a searching algorithm called binary search!

Like the Polygon drawer this game has a few iterations where it gets more complicated.

Basic Version

This first version will play the game through one time and get the user to input a guess between 1-100 until it reaches a randomly selected number!

What you'll need:

- A way to pick a random number as the target.
- A way to get the player to input numbers.
- A way to check if the guess equals the target.
- A way to tell the player if their guess was too low/high.
- A way to tell them they've won.

Hints:

You'll need a *variable* to hold the target value which should be a random number.

You'll need some sort of *loop* to keep allowing the player to guess the answer.

You'll need to use a *conditional* to check whether the guess is equal, less than or more than the target.

Think back to how we used *input* in the Polygon drawer as a way of using the number the player enters.

Medium version

This version of the game will be very similar, but it'll let the player play again if they want without having to press the green flag!

What you'll need:

- Everything we did in the last version.
- A way of asking the player if they want to play the game again.
- A way of picking a new random number.
- A way of restarting the game.

Hints:

Depending on how you did the previous version you'll have to change more/less of your game to get this version to work. The basic gameplay (the guessing and telling the player) is the same but some parts will have to change.

You'll need a message other than the green flag that tells the game that the game should start (think about "*broadcast*").

You'll need to send this message if they type in yes. You can check this using a *conditional*. You'll also need to send this message at the beginning of the game.

You need to make sure that the target *variable* is changed!

Advanced version

In this game we'll give the user the option of playing an easy/medium or hard version of the game. We'll also track the number of guesses they make and tell them if they could have done better!

What you'll need:

- Everything we've done so far.
- A way of taking in *input* to see whether they want to play an easy, medium, hard game.
- A way of picking a bigger random number depending on what they choose (1-100 = easy, 1-1000 = medium, 1-1000000 = hard).
- A way of knowing which level they picked.
- A way of tracking the number of guesses they've made.
- A way of telling them whether or not they guessed more or less than the expected amount.

Hints:

To know which level they've picked you'll probably need a *variable* and a broadcasted message. You could set the variable equal to 0, 1 or 2 depending what level they choose.

Make sure you change what the game tells the player to guess between depending on the level.

You'll need a *variable* to keep track of the number of guesses. Each time you check if it's the correct number you should increase this variable by 1, even the one when they get it correct!

To know whether they've used the fewest number of guesses or not you'll have to use something called an "and". You'll need to check if level equals 0, 1 or 2 AND whether or not the guessing variable is less than a specific value.

These specific values are as follows: For 1-100 it can always be done in 7 or less guesses, 1-1000 it can be done in 10 or less, 1-1000000 it can be done in 20. *

To do the checking of levels and guess value there's (at least) two options:

You could have a different "*if-then, else*" (*conditional*) piece for the 3 possibilities. Then in the condition you can use an "_ and _" block which can be found in **Data.** You then have to use the *"_ = _"* and *"_ < _"* pieces also found in Data.

You could also use the *"_ or _"* piece found in Data. You only need an "*if-then, else*" piece and inside the "if" you would put something like below:

(Level = 0 and guesses < 8) OR (Level = 1 and guesses < 10) OR (level = 2 and guessed < 20)

** To figure this out, we use the binary search algorithm. The way it works is using logs. As we're halving the search space each time, we can take the log of the number base 2.*

So log2(100) = 6.643856, log2(1000) = 9.965784, log2(1000000) = 19.931569

E

PYTHON LESSONS

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 1

Make sure you are comfortable with the content covered in class.

Log on to repl.it and follow the instructions given to create an account.

Each question below will have a separate exercise in the class you join on repl.it, simply type the code into it and it will submit it for correction.

Warm up question – write the answer on paper.

What is printed after the following code executes?

```
day = "Thursday"

day = 32.5

day = 19

print(day)
```

Q1. Hello World!

You've hopefully done this one in class already! If not give it a go.

Print the message "Hello World" to the screen.

Q2. Make some variables

Make three variables, one that holds a String, one that holds an integer and one that holds a floating point (decimal) number.

Print them all to the screen.
Then print their types to the screen.

Q3. Some maths!

Write a Python program that declares two variables which hold decimal point numbers and prints their average, sum and product to the screen. Use the following format:

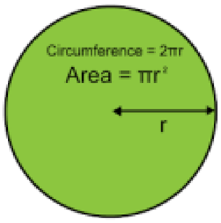
```
The sum of the two numbers is: X
The product of the two numbers is: Y
The average of the two numbers is: W
```

Q4 Area of a circle

Write a program that will calculate the area and the circumference of a circle given a radius of 5.

The area and circumference of a circle can be calculated using the formulas shown below. The value of π = 3.14.

Try different values for the radius when you have it completed for the first value.



Q5. Swapping siblings!

Write a program that will have a variable to store someone's name and their siblings name. Using these variables print it out so that it looks like the following:

```
My name is John.
My brother is called Bob.
```

After these have printed swap the variables around. Then print out the results again so it should say something like this:

```
My name is Bob.
My brother is called John.
```

Feedback

When you've finished this question sheet please give some feedback by clicking this link https://goo.gl/forms/4M5j2MAmSrt0ONUQ2.

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 1 Teachers Guide

Strand: Programming – Python *Duration: 1 hour*

Concepts covered:

Printing, variables, variable types, mathematical operators

Make sure you are comfortable with the content covered in class.

Log on to repl.it and follow the instructions given to create an account.

Each question below will have a separate exercise in the class you join on repl.it, simply type the code into it and it will submit it for correction.

Warm up question – write the answer on paper.

What is printed after the following code executes?

```
day = "Thursday"

day = 32.5

day = 19

print(day)
```

Solution and explanation:

19 will be printed to the screen. This is because the last assignment of day is 19, each time day is overwritten.

Q1. Hello World!

You've hopefully done this one in class already! If not give it a go.

Print the message "Hello World" to the screen.

Solution:

```
print("Hello World")
```

Q2. Make some variables

Make three variables, one that holds a String, one that holds an integer and one that holds a floating point (decimal) number.

Print them all to the screen.
Then print their types to the screen.

Solution:

```
string = "Hello my name is Joe"

number = 5

floating = 10.56

print(string)

print(number)

print(floating)

print(type(string))

print(type(number))

print(type(floating))
```

Q3. Some maths!

Write a Python program that declares two variables which hold decimal point numbers and prints their average, sum and product to the screen. Use the following format:

```
The sum of the two numbers is: X
The product of the two numbers is: Y
The average of the two numbers is: W
```

Solution:

```
num1 = 4.6

num2 = 6.4

print ('The sum of the two numbers is: ', num1+num2)

print ('The product of the two numbers is: ',  num1*num2)

print ('The average of the two numbers is: ', (num1+num2)/2)
```

When printing strings with other variables you must use the comma to separate them as shown above. In the next class, we'll be printing strings together (concatenation) and to do that you use the '+' symbol. It's important to note you can't use this with the above examples as you get an error. Run the below line to see an example!

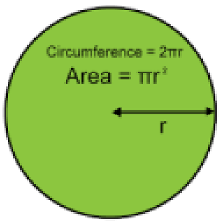
```
print('The sum of the two numbers is: ' + num1+num2)
```

Q4 Area of a circle

Write a program that will calculate the area and the circumference of a circle given a radius of 5.

The area and circumference of a circle can be calculated using the formulas shown below. The value of π = 3.14.

Try different values for the radius when you have it completed for the first value.



Solution:

```
radius = 5

pi = 3.14

area = pi*radius**2

circumference = 2*pi*radius

print ('The area of the circle is: ', area)

print ('The circumference of the circle is', circumference)
```

Q5. Swapping siblings!

Write a program that will have a variable to store someone's name and their siblings name. Using these variables print it out so that it looks like the following:

```
My name is John.
My brother is called Bob.
```

After these have printed swap the variables around. Then print out the results again so it should say something like this:

```
My name is Bob.
My brother is called John.
```

Solution:

```
myName = "John"
sibling = "Bob"

print("My name is " + myName)
print("My brother is called " + sibling)

temp = myName
myName = sibling
sibling = temp

print("My name is " + myName)
print("My brother is called " + sibling)
```

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 2

Make sure you are comfortable with the content covered in class.

Each question below will have a separate exercise in the class you join on repl.it, simply type the code into it and it will submit it for correction.

Q6 String Basics

Create a String that references "Computer Science". Create a String that references "Education".

Use slicing to take "Science" from the first string and store it in a new String.

Use String concatenation to form a new String "Science Education" from the new string and the second string.

Print the concatenated String to the screen.

Q7 My favourite colour

Create a String that references “My favourite colour is yellow”.

Find and print the position of the word “is”, the letter “y” and the word “our”.

Q8 Changing Cases

Create a String with any value you like, just make sure it’s multiple words.

Print the String, then convert it to upper case, print this and then covert it to lower case and print that.

Q9 Replacing the vowels

Create a String which references “It’s the end of this world as we know it”.

Print the String to the screen.

Then replace all ‘o’ with ‘0’, all ‘i’ with ‘!’ and all ‘e’ with ‘£’.

Print the result to the screen.

Q10 Your favourite book

Write a program that asks the user to input their favorite book and the author. You should print these values back to the screen with an appropriate description of each value, see example below.

```
Your favourite book is: Lord of the Rings
Your favourite author is: JK Rowling
```

Q11 All about you

Write a program that takes in a person's first name, surname age, height (in meters) and a random fact about them.

Print out the type of each of these variables. Do you notice anything interesting about the answers?

Replace all a's in either name with the # symbol and any 's' in the random fact with '$'.

Print the answers back to the screen with appropriate descriptions like the example below:

```
Your first name: G#ry
Your surname: Ad#ams
Age: 20
Height: 1.75m
Random fact: My favourite colours are silver and scarlet
```

Hint: make sure it works for both names and the fact by testing it with the above example.

Feedback

When you've finished this question sheet please give some feedback by clicking this link https://goo.gl/forms/fI7SVJof8xTJDyCF3.

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 2 Teachers Guide

Strand: Programming – Python   *Duration: 1 hour*

Concepts covered:

Strings, various String functions, user input, variables, variable types, printing

Q6 String Basics

Create a String that references "Computer Science". Create a String that references "Education".

Use slicing to take "Science" from the first string and store it in a new String.

Use String concatenation to form a new String "Science Education" from the new string and the second string.

Print the concatenated String to the screen.

Solution:

```
myString = "Computer Science"

myOtherString = "Education"

myNewString = myString[9:len(myString)] #or myNewString =
myString[9,16]

print(myNewString)

finalString = myNewString + " " + myOtherString

print(finalString)
```

Q7 My favourite colour

Create a String that references “My favourite colour is yellow”.

Find and print the position of the word “is”, the letter “y” and the word “our”.

Solution

```
myColour = "My favourite colour is yellow"

print(myColour.find("is"))
print(myColour.find("y"))

print(myColour.find("our"))

print(myColour.find("bed"))
```

Q8 Changing Cases

Create a String with any value you like, just make sure it's multiple words.

Print the String, then convert it to upper case, print this and then covert it to lower case and print that.

Solution

```
question3 = "This is an example"

print(question3)

print(question3.upper())

print(question3.lower())

#this is the most efficient way of answering our question, see
SolutionsLab2.py for more solutions
```

Q9 Replacing the vowels

Create a String which references "It's the end of this world as we know it".

Print the String to the screen.

Then replace all 'o' with '0', all 'i' with '!' and all 'e' with '£'.

Print the result to the screen.

Solution

```
question4 = "It's the end of this world as we know it"

print(question4)

question4 = question4.replace('o', '0')

print(question4) #this is just to show you each step as it goes

question4 = question4.replace('i', '!')

print(question4) #this is just to show you each step as it goes

question4 = question4.replace('e', '£')

print(question4)

#this is the most efficient way of answering our question, see
SolutionsLab2.py for more solutions
```

Q10 Your favourite book

Write a program that asks the user to input their favorite book and the author. You should print these values back to the screen with an appropriate description of each value, see example below.

Your favourite book is: Lord of the Rings
Your favourite author is: JK Rowling

Solution

```
favBook = input("What is your favourite book?")
favAuthor = input("Who is your favourite author?")
print("Your favourite book is: ", favBook)
print("Your favourite author is: ", favAuthor)

#you technically don't need the question inside the input() brackets
but it makes it look better, also how is someone supposed to know what
you want them to give you!
```

Q11 All about you

Write a program that takes in a person's first name, surname age, height (in meters) and a random fact about them.

Print out the type of each of these variables. Do you notice anything interesting about the answers?

Replace all a's in either name with the # symbol and any 's' in the random fact with '$'.

Print the answers back to the screen with appropriate descriptions like the example below:

Your first name: G#ry
Your surname: Ad#ams
Age: 20
Height: 1.75m
Random fact: My favourite colours are silver and scarlet

(Hint: make sure it works for both names and the fact by testing it with the above example).

Solution:

```
forename = input("What is your first name?")
surname = input("What is you surname?")
age = input("How old are you?")
height = input("How tall are you in meters?")
randomFact = input("What is a random fact about yourself?")

print(type(forename))
print(type(surname))
print(type(age))
print(type(height))
print(type(randomFact))

newSurname = surname.replace('a','#')
#or newForename = forename.replace('a','#')
newFact = randomFact.replace('s','$')

print("Your first name " + forename)
print("Your surname " + newSurname)
print("Age: " + age)
print("Height " + height + " m")
print("Random fact: " + newFact)
```

The interesting thing about this is even though age and height were numbers (int and double most likely) they are classed as Strings. This is because when you take in user input it is all read in as a String, this is important to note as it means if you wanted to add numbers up the user inputted you'd have to change them to ints! We'll do this later!

To prove the point about the types above, if you give an age of 20 and a height of 1 the below code will print 201. This is because it just takes the string "20" and concatenated it with the string "1" so we get the string "201". Add this line in to see this.

```
print(age + height)
```

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 3

Make sure you are comfortable with the content covered in class.

Each question below will have a separate exercise on repl.it, simply type the code into it and it will submit it for correction.

Q12 Starting with Turtle

Write a program that imports the turtle library and creates a Turtle that **draws a rectangle**.

Q13 Going around in circles

Write a program that imports the turtle library and creates a Turtle that **draws a circle**, you can decide the radius.

Q14 Blue square

Write a program that imports the turtle library and creates a Turtle that **draws a square**. **Set the line colour to blue and the pen size to 5.**

Q15 Make a screen

Write a program that imports the turtle library and creates a Turtle that draws a line of length 150. **This time make a Screen as well.** Set the background colour to black and the line colour to white.

Q16 That's why it's called Python Turtle!

Write a program that imports the turtle library and creates a Turtle **(and Screen)** that draws a square of side length 100. **Change the turtles shape to be a turtle.** Set the background colour to red, the line colour to black and the pen size to 10.

Q17 Triangle Part 1

Write a program that imports the turtle library and creates a Turtle that draws an **equilateral triangle**. If you want to make a screen do. Set the background colour, turtle shape, pen size and line colour to whatever you want.

Q18 Triangle Part 2

Write a program that imports the turtle library and creates a Turtle that draws an **isosceles triangle**. If you want to make a screen do. Set the background colour, turtle shape, pen size and line colour to whatever you want.

If you're finished, have a look at the link given on the slides to all the other methods, play around with some of them and see what you can draw!

Feedback

When you've finished this question sheet please give some feedback by clicking this link https://goo.gl/forms/QwQtYfR9rxp3l75i2.

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 3 Teachers Guide

Strand: Programming – Python *Duration: 1 hour*

Concepts covered:

Importing libraries, creating objects, calling functions on objects, various turtle library functions (movement and appearance)

Make sure you are comfortable with the content covered in class.

Each question below will have a separate exercise on repl.it, simply type the code into it and it will submit it for correction.

Q12 Starting with Turtle

Write a program that imports the turtle library and creates a Turtle that **draws a rectangle**.

Solution

```
import turtle      # allows us to use the turtle library

myTurtle = turtle.Turtle()

#create a turtle object named myTurtle

myTurtle.forward(150)

myTurtle.left(90)

myTurtle.forward(30)

myTurtle.left(90)

myTurtle.forward(150)

myTurtle.left(90)

myTurtle.forward(30)

myTurtle.left(90)
```

Q13 Going around in circles

Write a program that imports the turtle library and creates a Turtle that **draws a circle**, you can decide the radius.

Solution

```
import turtle      # allows us to use the turtle library

myTurtle2 = turtle.Turtle() #create an object named myTurtle2

myTurtle2.circle(50)
```

Q14 Blue square

Write a program that imports the turtle library and creates a Turtle that **draws a square**. **Set the line colour to blue and the pen size to 5.**

Solution

```
import turtle      # allows us to use the turtle library

myTurtle3 = turtle.Turtle() #create an object named myTurtle3

myTurtle3.color("blue") #note the US spelling of color

myTurtle3.pensize(5)

myTurtle3.forward(150)

myTurtle3.left(90)

myTurtle3.forward(150)

myTurtle3.left(90)

myTurtle3.forward(150)

myTurtle3.left(90)

myTurtle3.forward(150)

myTurtle3.left(90)
```

Q15 Make a screen

Write a program that imports the turtle library and creates a Turtle that draws a line of length 150. **This time make a Screen as well.** Set the background colour to black and the line colour to white.

Solution

This one will be hard to see as it goes very quickly but it's just a white line

```
import turtle      # allows us to use the turtle library

win = turtle.Screen()

myTurtle4 = turtle.Turtle() # create an object named myTurtle4

win.bgcolor("black") #note the US spelling of color

myTurtle4.color("white")

myTurtle4.forward(150)

win.exitonclick() #prevents the screen from closing
automatically and allows you to close it when you want
```

Q16 That's why it's called Python Turtle!

Write a program that imports the turtle library and creates a Turtle **(and Screen)** that draws a square of side length 100. **Change the turtles shape to be a turtle.** Set the background colour to red, the line colour to black and the pen size to 10.

Solution:

```
import turtle      # allows us to use the turtle library

win2 = turtle.Screen()

myTurtle5 = turtle.Turtle() # create an object named myTurtle5

win2.bgcolor("red") #note the US spelling of color

myTurtle5.color("black") #note the US spelling of color

myTurtle5.shape("turtle")

myTurtle5.pensize(10)

myTurtle5.forward(150)

myTurtle5.left(90)

myTurtle5.forward(150)

myTurtle5.left(90)

myTurtle5.forward(150)

myTurtle5.left(90)

myTurtle5.forward(150)

myTurtle5.left(90)
```

```
win2.exitonclick() #prevents the screen from closing
automatically and allows you to close it when you want
```

Q17 Triangle Part 1

Write a program that imports the turtle library and creates a Turtle that draws an **equilateral triangle**. If you want to make a screen do. Set the background colour, turtle shape, pen size and line colour to whatever you want.

Solution

```
import turtle      # allows us to use the turtle library

win3 = turtle.Screen()

myTurtle6 = turtle.Turtle() # create an object named myTurtle6

win3.bgcolor("light blue") #another example of a color

myTurtle6.color("#285078") #another example of a color input
format

myTurtle6.shape("classic") #another example of a turtle shape

myTurtle6.forward(150)

myTurtle6.left(120)

myTurtle6.forward(150)

myTurtle6.left(120)

myTurtle6.forward(150)

myTurtle6.left(120)

win3.exitonclick() #prevents the screen from closing
automatically and allows you to close it when you want
```

Q18 Triangle Part 2

Write a program that imports the turtle library and creates a Turtle that draws an **isosceles triangle**. If you want to make a screen do. Set the background colour, turtle shape, pen size and line colour to whatever you want.

If you’re finished, have a look at the link given on the slides to all the other methods, play around with some of them and see what you can draw!

Solution

```
import turtle      # allows us to use the turtle library
```

```
myTurtle7 = turtle.Turtle() # create an object named myTurtle7

myTurtle7.shape("circle") #another example of a turtle shape

myTurtle7.forward(250)

myTurtle7.left(150)

myTurtle7.forward(250)

myTurtle7.left(60)

myTurtle7.forward(250)

myTurtle7.left(150)

myTurtle7.forward(250)
```

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 4

You should comment your code to say what each line of code is doing.

Q19. Make your own circle

Write a program that creates a circle. Change the background colour to black, line colour to white and pen size to 3. The program should use user input for the radius of the circle.

Q20. Shape-ception

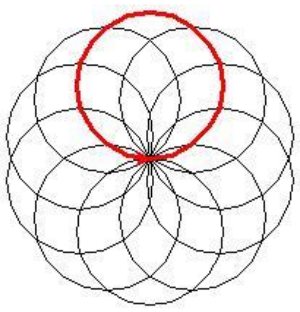
Write a program that creates a square and then a circle inside that fits perfectly. You should also change the background colour and the shape of the turtle.

Q21. Loopy pentagons

Write a program that **uses a for loop** to create a regular pentagon with each side being 80 units in length.

Q22. Going around in circles

Write a program that produces the following shape



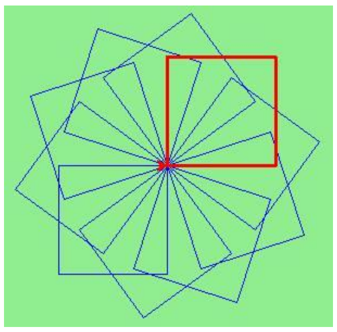
Hint: Notice that it is ten circles

Q23. Asterisk

Write a program that uses a for loop to create a six-sided asterisk (star symbol on your keyboard) with each line being 50 units of length from the centre.

Q24. Starry squares

Write a program that produces the following shape



Hint: Notice that it is ten squares

Feedback

When you've finished this question sheet please give some feedback by clicking this link https://goo.gl/forms/EHw6QAHxqBqDjNCo1.

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 4 Teachers Guide

Strand: Programming – Python   *Duration: 1 hour*

Concepts covered:

For loops, importing libraries, creating objects, calling functions on objects, various turtle library functions (movement and appearance), Strings, various String functions, user input, variables, printing

You should comment your code to say what each line of code is doing.

Q19. Make your own circle

Write a program that creates a circle. Change the background colour to black, line colour to white and pen size to 3. The program should use user input for the radius of the circle.

Solution

```
import turtle

win = turtle.Screen()

myTurtle = turtle.Turtle()

win.bgcolor("black")

myTurtle.color("white")

myTurtle.pensize(3)

radius = input("What radius circle do you want to draw?")

myTurtle.circle(radius)

win.exitonclick()
```

Q20. Shape-ception

Write a program that creates a square and then a circle inside that fits perfectly. You should also change the background colour and the shape of the turtle.

Solution

```
import turtle

win2 = turtle.Screen()

myTurtle2 = turtle.Turtle()

win2.bgcolor("blue")

myTurtle2.shape("turtle")

for x in range(4):

     myTurtle2.forward(100)

     myTurtle2.right(90)

myTurtle2.penup()

myTurtle2.goto(50,-100)

myTurtle2.pendown()

myTurtle2.circle(50)

win2.exitonclick()
```

Q21. Loopy pentagons

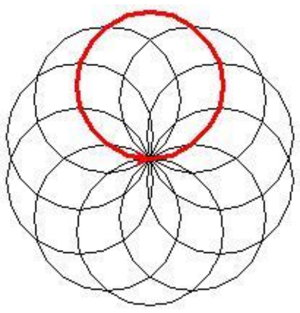
Write a program that **uses a for loop** to create a regular pentagon with each side being 80 units in length.

Solution

```
import turtle

win3 = turtle.Screen()

myTurtle3 = turtle.Turtle()

for x in range(5):

     myTurtle3.forward(80)

     myTurtle3.left(72)

win3.exitonclick()
```

Q22. Going around in circles

Write a program that produces the following shape



Hint: Notice that it is ten circles

Solution

```
import turtle

win4 = turtle.Screen()

myTurtle4 = turtle.Turtle()

for y in range(10):

  myTurtle4.circle(50)

  myTurtle4.left(36)

win6.exitonclick()
```

Q23. Asterisk

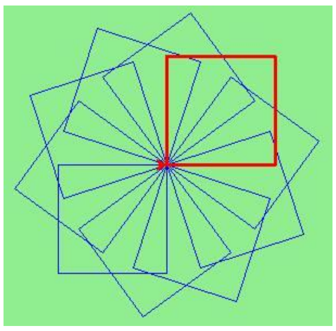
Write a program that uses a for loop to create a six-sided asterisk (star symbol on your keyboard) with each line being 50 units of length from the centre.

Solution

```
import turtle

win5 = turtle.Screen()

myTurtle5 = turtle.Turtle()

myTurtle5.left(90)

for x in range(6):

     myTurtle5.forward(50)

     myTurtle5.backward(50)

     myTurtle5.left(60)

win5.exitonclick()
```

Q24. Starry squares

Write a program that produces the following shape



Hint: Notice that it is ten squares

Solution

```
import turtle

win6 = turtle.Screen()

myTurtle6 = turtle.Turtle()

for y in range(10):

  for x in range(4):

       myTurtle6.forward(80)

       myTurtle6.left(90)
  myTurtle6.left(36)
win6.exitonclick()
```

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 5

You should comment your code to say what each line of code is doing.

Q25. Kicked out of the party

Write a program that has an integer value to represent someone's age, give it any value you want. Using if and else statements, check to see whether someone is allowed into an over-18's birthday party. If they're under 18 (< 18) it should print a message like "You're not allowed into the party" and if they're over 18 (> 18) then it should print a message like "Welcome to the party!".

Note: Try out the code with different values

Q26. VIP

The invites to our party said that if you were 21 or over then you get into a special VIP section! We want to make our program accommodate this by printing a special message if you're over 21 using if/elif/else statements.

If someone is over 21 it should print something like "Thanks for coming, let me show you to the VIP area!".

Note: You can copy the code from the previous question and just make the changes. Make sure it lets you into the VIP section if they're 21!

Q27. Positively negative

Write a program that has an integer value, it can be any value. Using if, else if and else statements we're going to draw a different shape depending on whether the number is negative, positive or zero. If it's 0 we want to draw a circle, if it's negative then we want to draw a square and if it's positive draw a triangle. You can make any changes to colour or how it looks that you want.

Note: Make sure you check that your code is working by changing the values of the variable. Also make sure you import the turtle library and make a turtle object!

Q28. Long name

Write a program that has a variable to hold someone's name. We want to see how long their name is and split them into 3 groups: short (<7 letters), long (7 to 10 letters) and very long (10+ letters). Print out appropriate messages for each such as below:

Hello Tom! Your name is short!

Feel free to change the groupings and the background colour if you want.

Hint: To do this you'll need to use a string method from a previous lesson!

Q29. Odd Square

Write a program that takes an integer number from the user. If the number is even you should draw a square to the screen, if it's even draw a circle. You can choose the size and any colours.

Q30. Your long name!

Edit the program from Q28 so that it takes in user input of their name. You should also keep taking in names until the user types "stop".

You should print out answers that look something like this:

Hello Tom! Your name is short!

Hello Natalie! Your name is long!

Hello Christopher! Your name is very long!

*Hint: To keep allowing the user to type names you'll need to use an **infinite loop**. You'll then need to check if the inputted word was "stop" (using an if statement) and if it is, use the keyword **break** to end the loop.*

Feedback

When you've finished this question sheet please give some feedback by clicking this link https://goo.gl/forms/LulAvXbfKJbJLm4E3.

Developed by James Lockwood
& Dr Aidan Mooney

Python Assignment 5 Teachers Guide

Strand: Programming – Python   *Duration: 1 hour*

Concepts covered:

Conditionals (if, elif, else), while loops, break, for loops, importing libraries, creating objects, calling functions on objects, various turtle library functions (movement and appearance), Strings, various String functions, user input, variables, printing, mathematical operators

You should comment your code to say what each line of code is doing.

Q25. Kicked out of the party

Write a program that has an integer value to represent someone's age, give it any value you want. Using if and else statements, check to see whether someone is allowed into an over-18's birthday party. If they're under 18 (< 18) it should print a message like "You're not allowed into the party" and if they're over 18 (> 18) then it should print a message like "Welcome to the party!".

Note: Try out the code with different values

Solution

```
age = 21

if(age< 18):

    print("You're not allowed in the party!")

else:

    print("Welcome to the party!")
```

Q26. VIP

The invites to our party said that if you were 21 or over then you get into a special VIP section! We want to make our program accommodate this by printing a special message if you're over 21 using if/elif/else statements.

If someone is over 21 it should print something like “Thanks for coming, let me show you to the VIP area!”.

Note: You can copy the code from the previous question and just make the changes. Make sure it lets you into the VIP section if they’re 21!

Solution

```
age = 32

if(age< 18):

    print("You're not allowed in the party!")

elif(age>=21):

    print("Thanks for coming, let me show you to the VIP
area!")

else:

    print("Welcome to the party!")
```

Q27. Positively negative

Write a program that has an integer value, it can be any value. Using if, else if and else statements we’re going to draw a different shape depending on whether the number is negative, positive or zero. If it’s 0 we want to draw a circle, if it’s negative then we want to draw a square and if it’s positive draw a triangle. You can make any changes to colour or how it looks that you want.

Note: Make sure you check that your code is working by changing the values of the variable. Also make sure you import the turtle library and make a turtle object!

Solution

```
import turtle

alex = turtle.Turtle()

number =0

if(number>0):

    alex.forward(100)

    alex.left(120)

    alex.forward(100)

    alex.left(120)
    alex.forward(100)

elif(number<0):

    alex.forward(100)

    alex.left(90)

    alex.forward(100)

    alex.left(90)

    alex.forward(100)

    alex.left(90)

    alex.forward(100)

else:

    alex.circle(100)
```

Q28. Long name

Write a program that has a variable to hold someone's name. We want to see how long their name is and split them into 3 groups: short (<7 letters), long (7 to 10 letters) and very long (10+ letters). Print out appropriate messages for each such as below:

Hello Tom! Your name is short!

Feel free to change the groupings and the background colour if you want.

Hint: To do this you'll need to use a string method from a previous lesson!

Solution

```
name = "Thomas"

if(len(name)<7):

    print "Hi",name,"! You've got a short name!"

elif(len(name)<10):

    print "Hi", name, "! You've got a long name!"

else:

    print "Hi", name, "! You've got a very long name!"
```

Q29. Odd Square

Write a program that takes an integer number from the user. If the number is even you should draw a square to the screen, if it's odd draw a circle. You can choose the size and any colours.

Solution

```
import turtle

alex = turtle.Turtle()

number = input("Please enter a number")

if(number%2==0):

      alex.forward(100)

      alex.left(90)

      alex.forward(100)

      alex.left(90)

      alex.forward(100)

      alex.left(90)

      alex.forward(100)

else:

      alex.circle(100)
```

Q30. Your long name!

Edit the program from Q28 so that it takes in user input of their name. You should also keep taking in names until the user types "stop".

You should print out answers that look something like this:

Hello Tom! Your name is short!

Hello Natalie! Your name is long!

Hello **Christopher**! Your name is very long!

Hint: To keep allowing the user to type names you'll need to use an ***infinite loop****. You'll then need to check if the inputted word was "stop" (using an if statement) and if it is, use the keyword* ***break*** *to end the loop.*

Solution

```
while(True):

  name = input("Please enter a name or hit stop to end")

  if(name=="stop"):

     break

  else:

     if(len(name)<7):

          print "Hi",name,"! You've got a short name!"

     elif(len(name)<10):

          print "Hi", name, "! You've got a long name!"

     else:

          print "Hi", name, "! You've got a very long name!"
```

Maynooth University
National University of Ireland Maynooth

DEVELOPED BY JAMES LOCKWOOD

Python Assignment 6

Make sure you are comfortable with the content covered in class.

Each question below will have a separate exercise on repl.it, simply type the code into it and it will submit it for correction.

Q31. My first function

Write a program that contains a function to draw a square. Call this function twice, changing the position and pen colour in between.

Note: To call a function on a turtle you use the following syntax; turtleName.functionName()

Q32. Pretty average

Write a program that contains a function to calculate the average of 3 numbers. Call this function twice on two different sets of numbers. Print out the results of the method call each time.

Q33. Buy one get one free!

Write a program that contains two functions. One of these should calculate the average of 3 numbers and print this out. The other should find the biggest and smallest number of the same 3 numbers and print these out. Call these functions one two sets of numbers to check that it works.

Q34. 30 days has September…

Write a function that takes in a word. If a correct month is inputted then it should print out the number of days in that month. Call this function on a few different months.

Q35. Pick a shape, any shape

Write a program that contains a few different functions that draw a square, a pentagon and a triangle.

Ask the user to input a string, either “square”, “pentagon”, “triangle” and “stop”. If they type square you should call a square() function, if they type triangle you should call a triangle() function etc. You should keep taking in input and drawing shapes until the user types “stop”.

Hint: You’ll need an infinite loop to keep printing. You also might want to use the .clear() method to make the program look neater.

Feedback

When you’ve finished this question sheet please give some feedback by clicking this link https://goo.gl/forms/geHWfrAN6iu0iwRc2.

Maynooth University
National University of Ireland Maynooth

DEVELOPED BY JAMES LOCKWOOD

Python Assignment 6 Teachers Guide

Strand: Programming – Python   *Duration: 1 hour*

Concepts covered:

Functions (def), conditionals (if, elif, else), while loops, break, for loops, importing libraries, creating objects, calling functions on objects, various turtle library functions (movement and appearance), Strings, various String functions, user input, variables, printing, mathematical operators

Q31. My first function

Write a program that contains a function to draw a square. Call this function twice, changing the position and pen colour in between.

Note: To call a function on a turtle you use the following syntax; turtleName.functionName()

Solution

```
import turtle

win = turtle.Screen()

myTurtle = turtle.Turtle()

def drawSquare():

    for x in range(4):

        myTurtle.forward(60)

        myTurtle.right(90)

myTurtle.color("blue")

drawSquare()

myTurtle.color("red")

drawSquare()

win.exitonclick()
```

Q32. Pretty average

Write a program that contains a function to calculate the average of 3 numbers. Call this function twice on two different sets of numbers. Print out the results of the method call each time.

Solution

```
def calculateAvg(a,b,c):

     print((a+b+c)/3)



calculateAvg(3,4,5)

calculateAvg(7,8,9)
```

Q33. Buy one get one free!

Write a program that contains two functions. One of these should calculate the average of 3 numbers and print this out. The other should find the biggest and smallest number of the same 3 numbers and print these out. Call these functions one two sets of numbers to check that it works.

Solution

```
def calculateAvg(a,b,c):

      print((a+b+c)/3)

def findBigAndSmall(a,b,c):

      largest = a

      smallest = a

      if(b>a):

            if(c>b): #then c is also bigger than a

                 largest =c

                         #if b is bigger than a, and c is bigger than b,
                         then c must be the largest

            else: #b is bigger than c

                  largest=b

                        #if b is bigger than a, and b is bigger than c,
                        then b must be the largest

      else: #b is smaller than a

            if(c>a):

                  largest=c

                  #if a is bigger than b, and c is bigger than a, then c
                  must be the largest

            elif(b>c): #so a must be bigger than c

                  smallest =c

                  #if a is bigger than c, and b is bigger than c, then c
                  must be the smallest

            else: # c is bigger than b

                  smallest = b
```

```
                    #if a is bigger than b, and c is bigger than b, then b
                    is smallest

          print(largest)

          print(smallest)

calculateAvg(3,4,5)

calculateAvg(7,8,9)

findBigOrSmall(13,4,5)

findBigAndSmall(7,8,9)

findBigAndSmall(5,8,6)
```

Q34. 30 days has September...

Write a function that takes in a word. If a correct month is inputted then it should print out the number of days in that month. Call this function on a few different months.

```
def howManyDays(month):

    if(month=="September" or month=="November" or month=="June"
or month=="April"):

        print("30 days")

    elif(month=="February"):

        print("28 days or 29 in a leap year!")

    else:

        print("31 days")


howManyDays("April")

howManyDays("February")

howManyDays("October")
```

Q35. Pick a shape, any shape

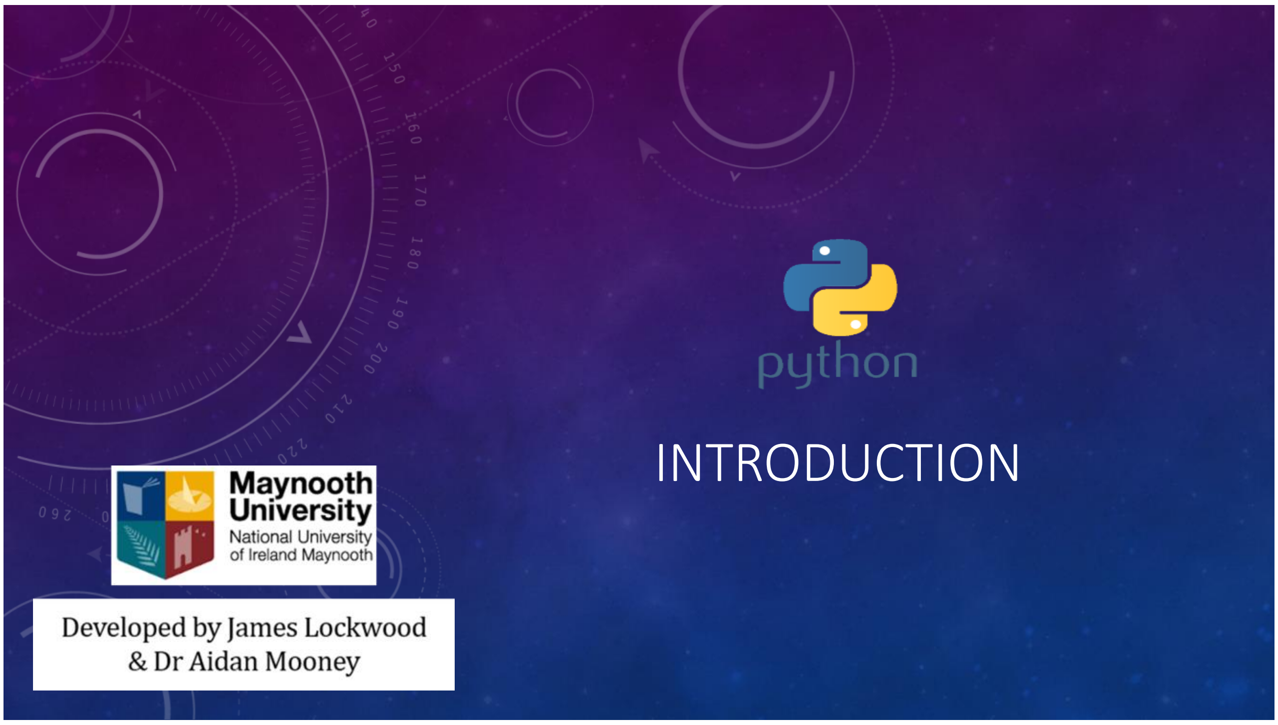
Write a program that contains a few different functions that draw a square, a pentagon and a triangle.

Ask the user to input a string, either “square”, “pentagon”, “triangle” and “stop”. If they type square you should call a square() function, if they type triangle you should call a triangle() function etc. You should keep taking in input and drawing shapes until the user types “stop”.

Hint: You’ll need an infinite loop to keep printing. You also might want to use the .clear() method to make the program look neater.

Solution

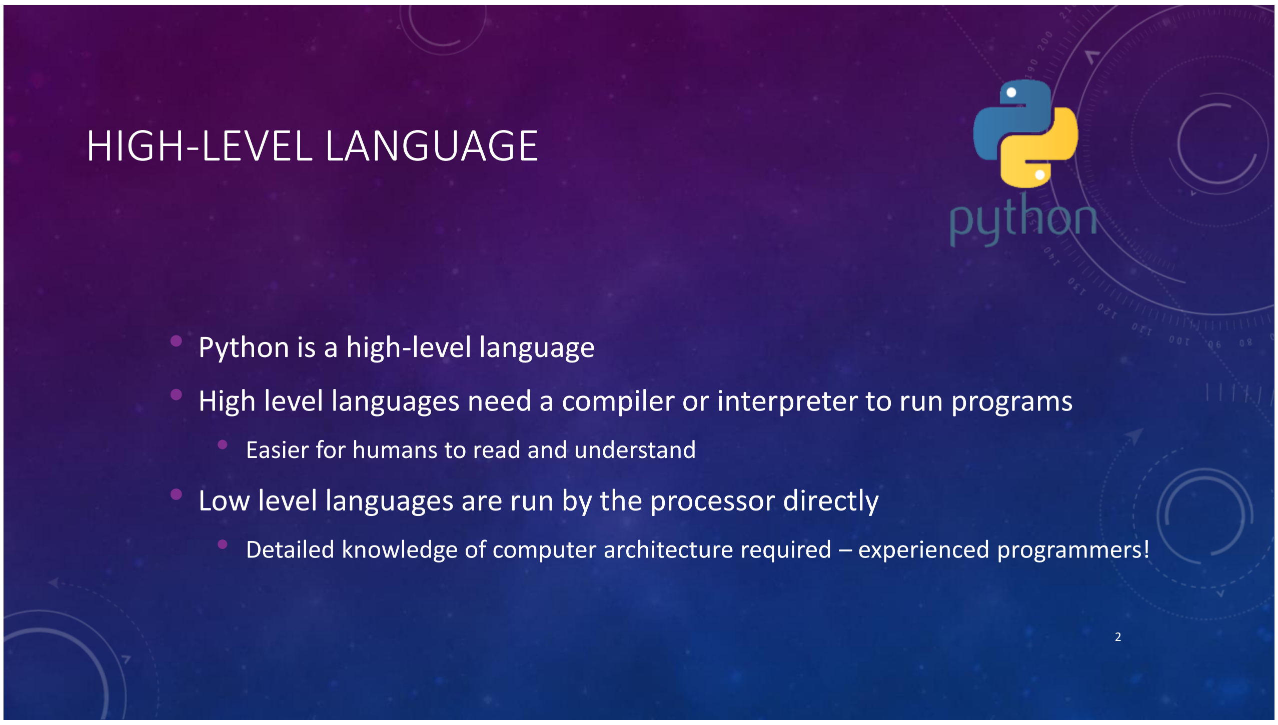
```
import turtle

win = turtle.Screen()

myTurtle = turtle.Turtle()

def drawTriangle():

    for x in range(3):

        myTurtle.forward(60)

        myTurtle.right(120)

def drawSquare():

    for x in range(4):

        myTurtle.forward(60)

        myTurtle.right(90)


def drawPentagon():

    for x in range(5):

        myTurtle.forward(60)

        myTurtle.right(72)


def shapes(word):

  if shape=="triangle":
      drawTriangle()
  elif(shape=="square"):
      drawSquare()
  elif(shape=="pentagon"):
      drawPentagon()
  elif(shape=="quit"):
         print("Goodbye")
  else:
       print("You didn't enter a shape please do so")


shape=""
while(shape!="quit"):
  shape = input("Please enter the shape you want to draw:
triangle, square or pentagon, enter stop to end\n")
  shapes(shape)
win.exitonclick()
```



INTRODUCTION

Maynooth University
National University of Ireland Maynooth

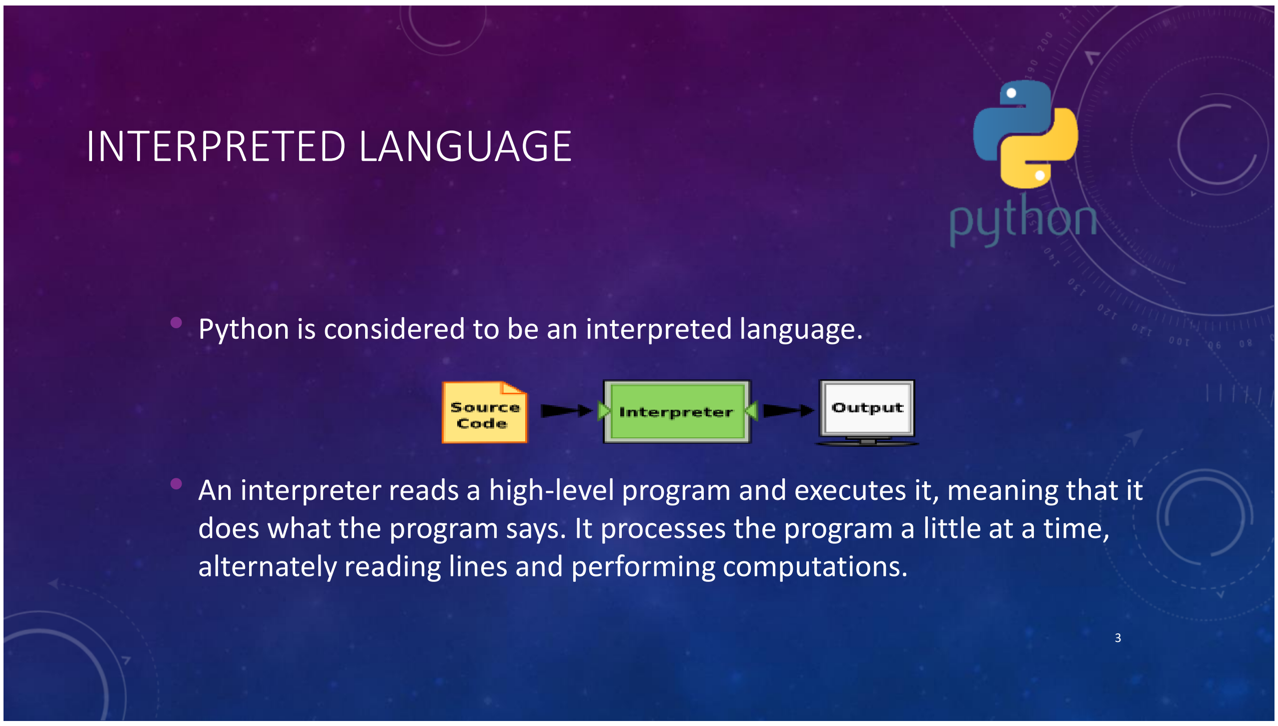
Developed by James Lockwood & Dr Aidan Mooney



HIGH-LEVEL LANGUAGE

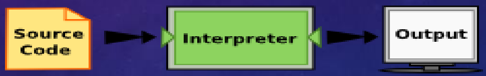
- Python is a high-level language
- High level languages need a compiler or interpreter to run programs
  - Easier for humans to read and understand
- Low level languages are run by the processor directly
  - Detailed knowledge of computer architecture required – experienced programmers!

2



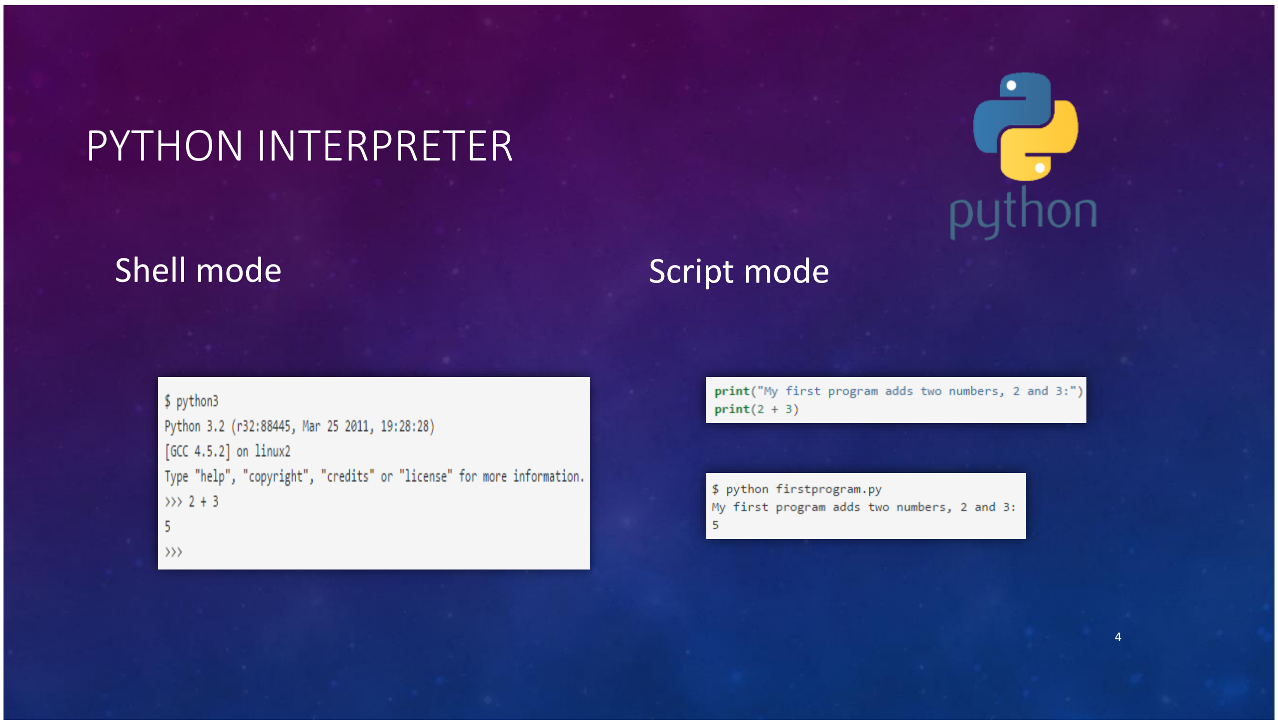
INTERPRETED LANGUAGE

- Python is considered to be an interpreted language.



- An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing computations.

3



PYTHON INTERPRETER

Shell mode

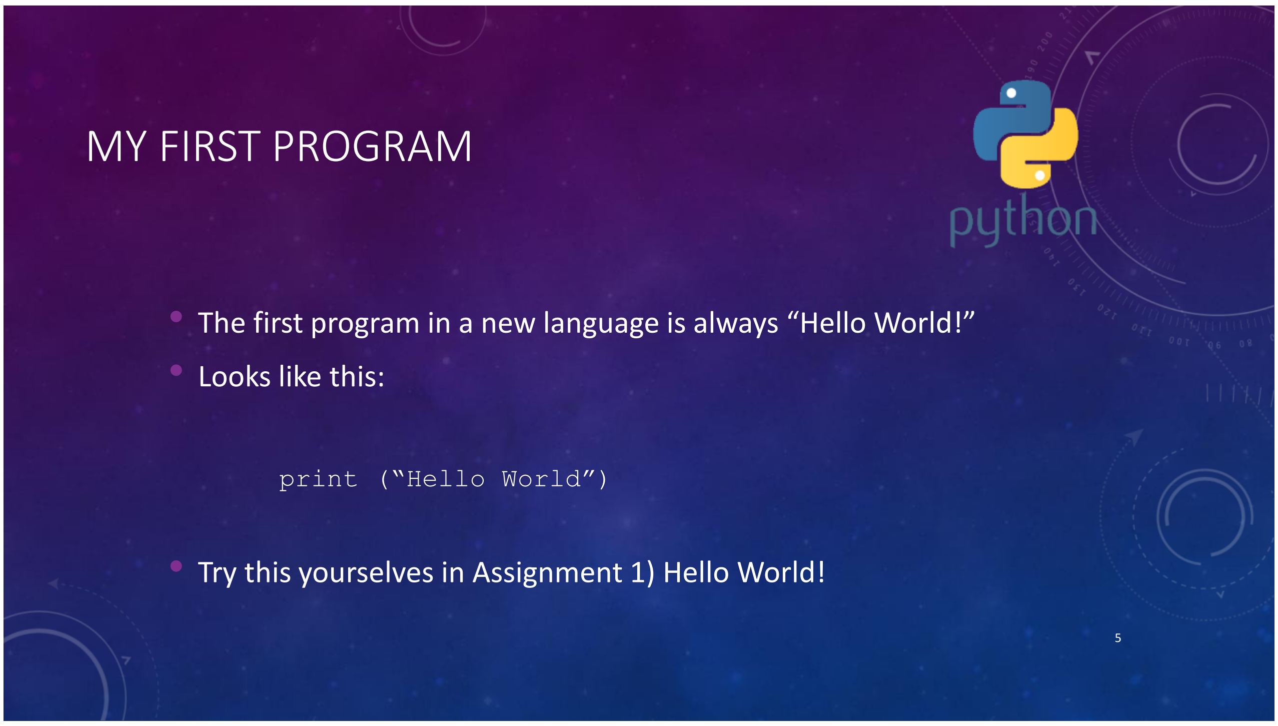
```
$ python3
Python 3.2 (r32:88445, Mar 25 2011, 19:28:28)
[GCC 4.5.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 3
5
>>>
```

Script mode

```
print("My first program adds two numbers, 2 and 3:")
print(2 + 3)
```

```
$ python firstprogram.py
My first program adds two numbers, 2 and 3:
5
```

4



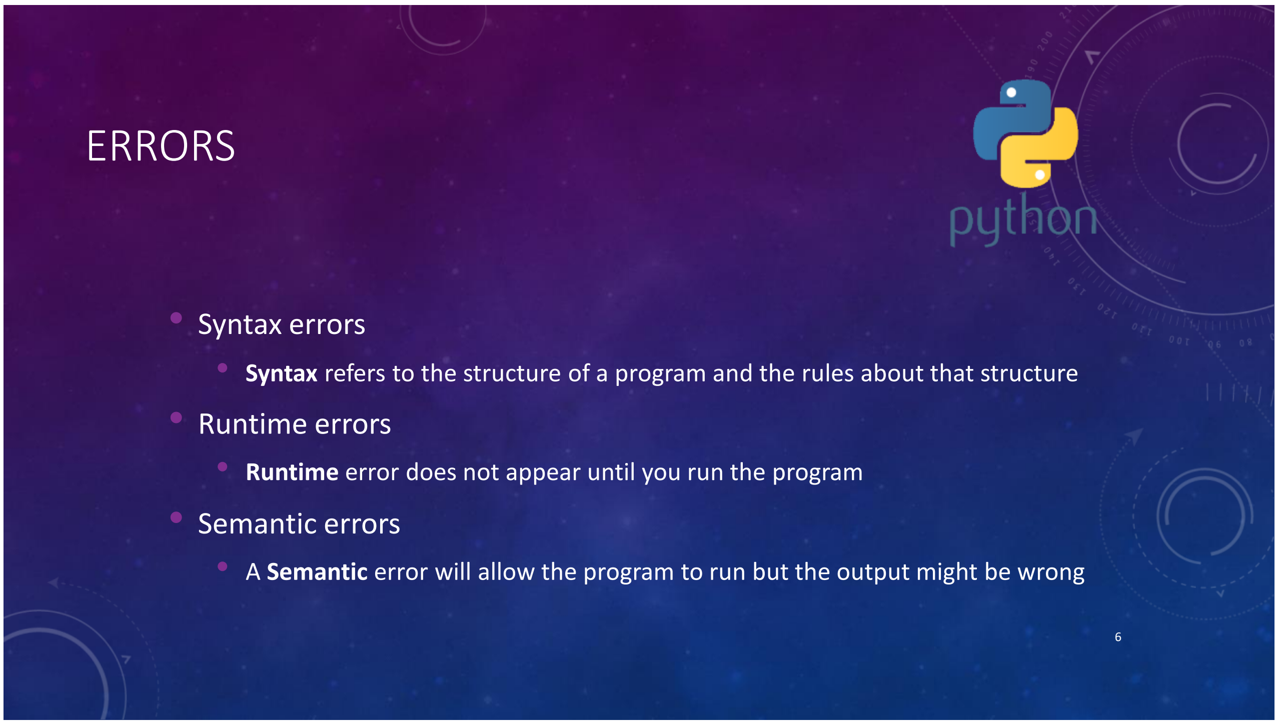
MY FIRST PROGRAM

- The first program in a new language is always “Hello World!”
- Looks like this:

```
print (“Hello World”)
```

- Try this yourselves in Assignment 1) Hello World!

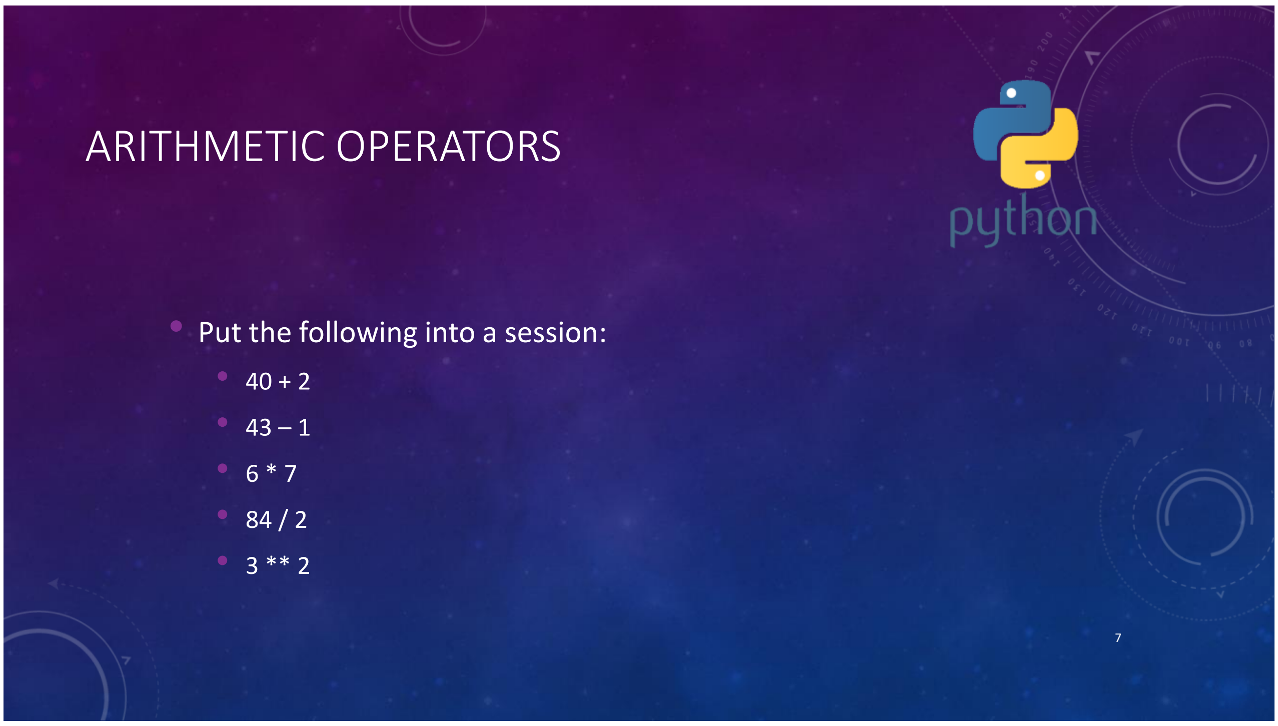
5



ERRORS

- Syntax errors
  - **Syntax** refers to the structure of a program and the rules about that structure
- Runtime errors
  - **Runtime** error does not appear until you run the program
- Semantic errors
  - A **Semantic** error will allow the program to run but the output might be wrong

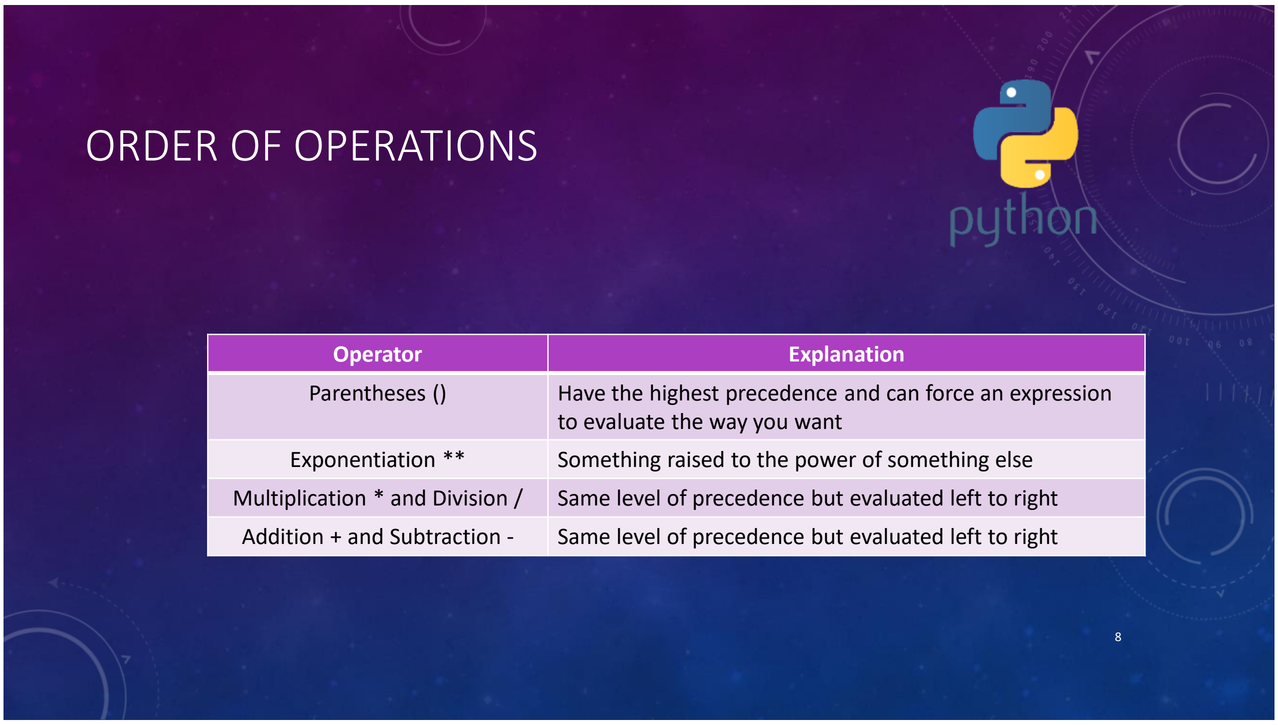
6



ARITHMETIC OPERATORS

- Put the following into a session:
  - 40 + 2
  - 43 – 1
  - 6 * 7
  - 84 / 2
  - 3 ** 2

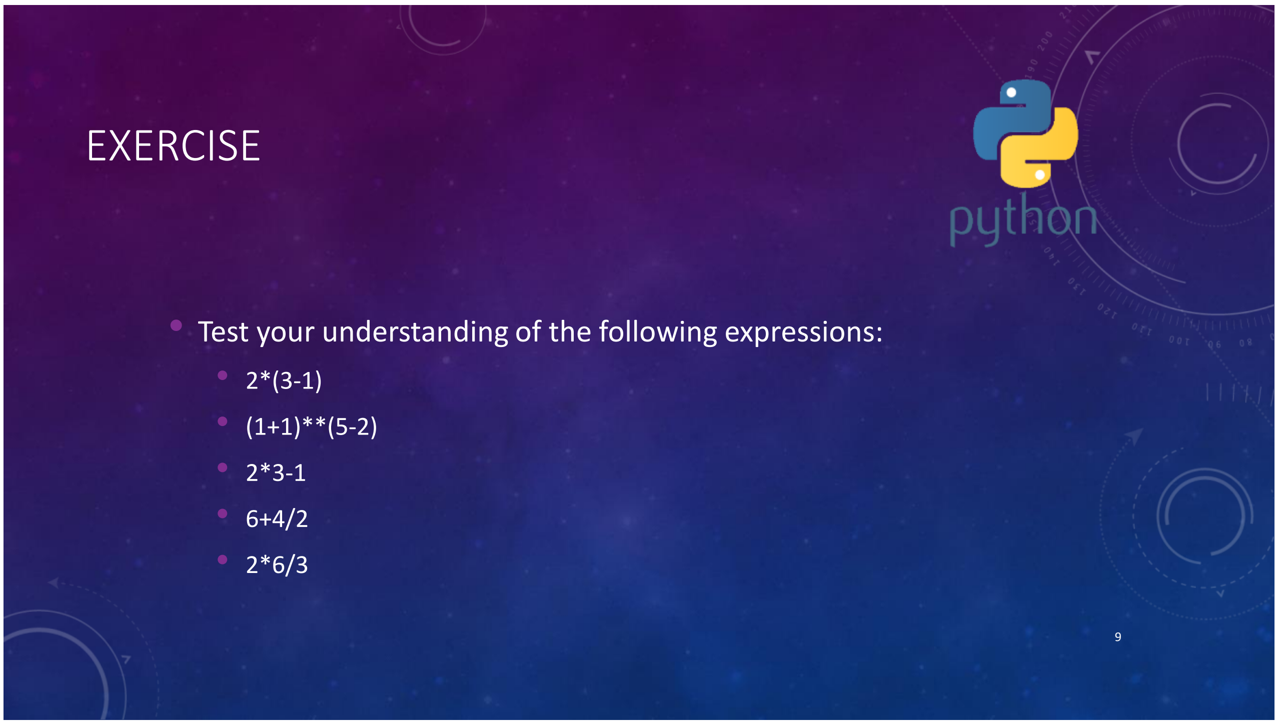
7



ORDER OF OPERATIONS

Operator	Explanation
Parentheses ()	Have the highest precedence and can force an expression to evaluate the way you want
Exponentiation **	Something raised to the power of something else
Multiplication * and Division /	Same level of precedence but evaluated left to right
Addition + and Subtraction -	Same level of precedence but evaluated left to right

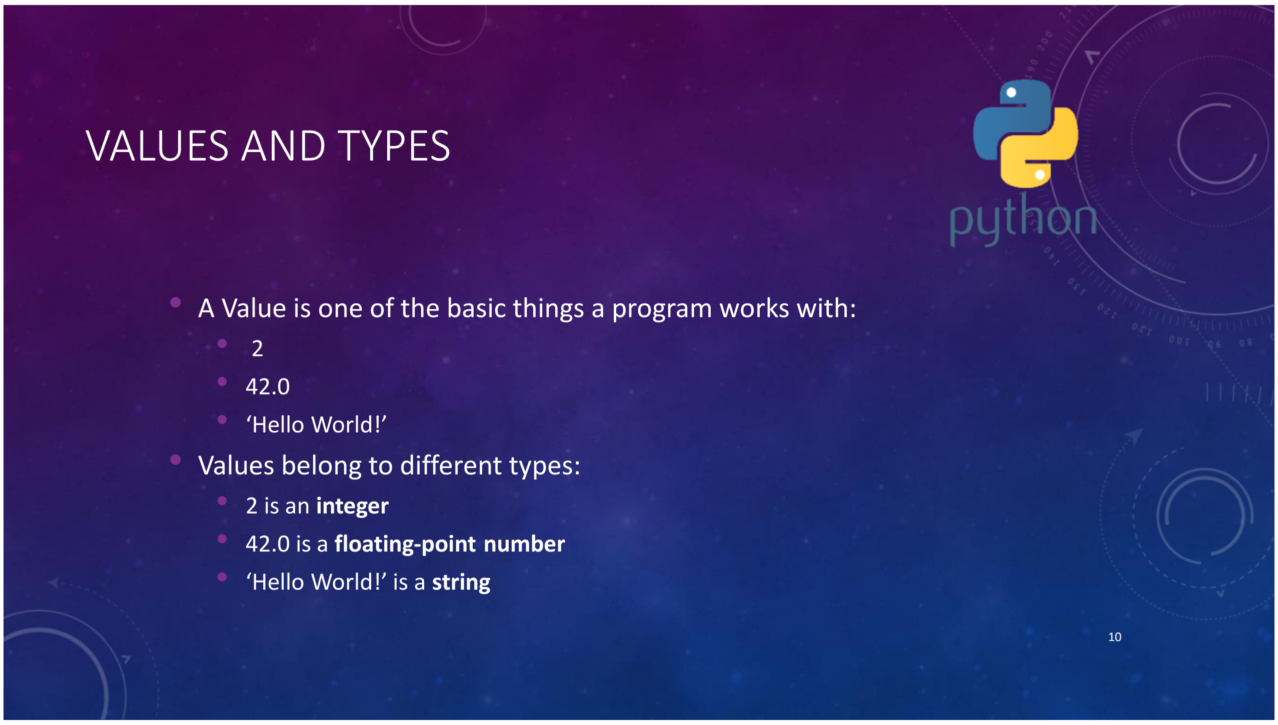
8



EXERCISE

- Test your understanding of the following expressions:
  - 2*(3-1)
  - (1+1)**(5-2)
  - 2*3-1
  - 6+4/2
  - 2*6/3

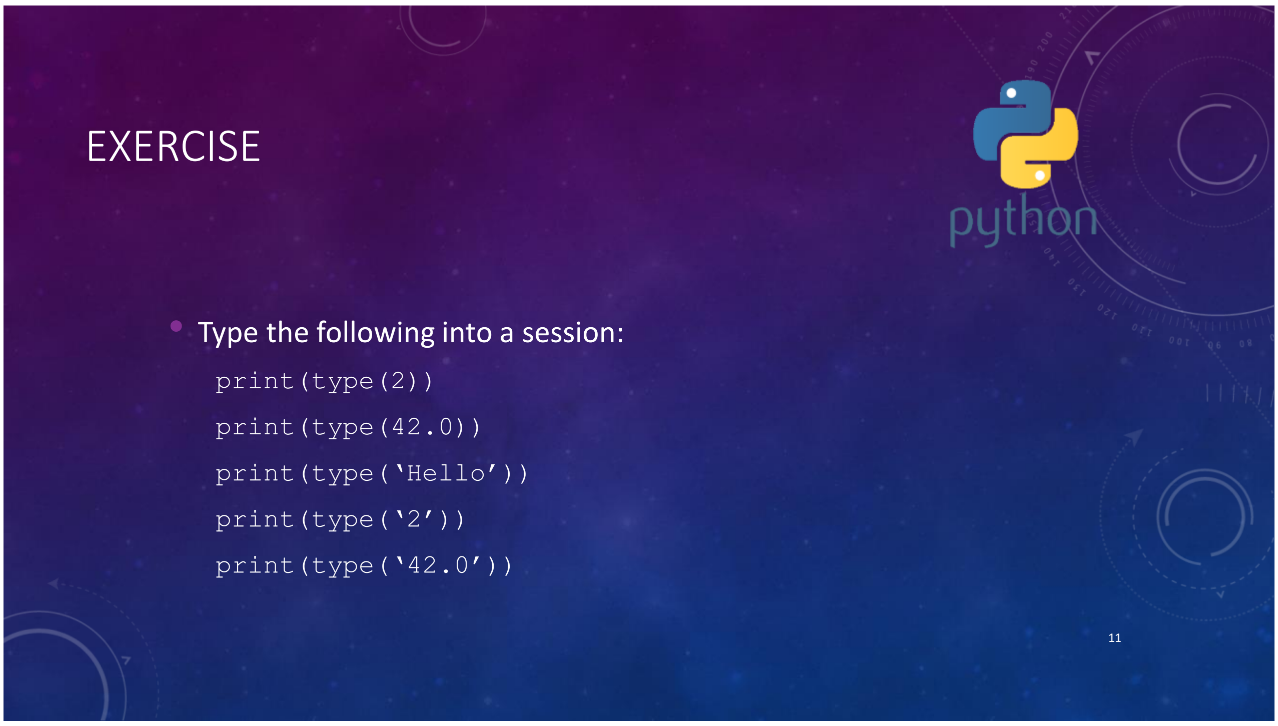
9



VALUES AND TYPES

- A Value is one of the basic things a program works with:
  - 2
  - 42.0
  - ‘Hello World!’
- Values belong to different types:
  - 2 is an **integer**
  - 42.0 is a **floating-point number**
  - ‘Hello World!’ is a **string**

10

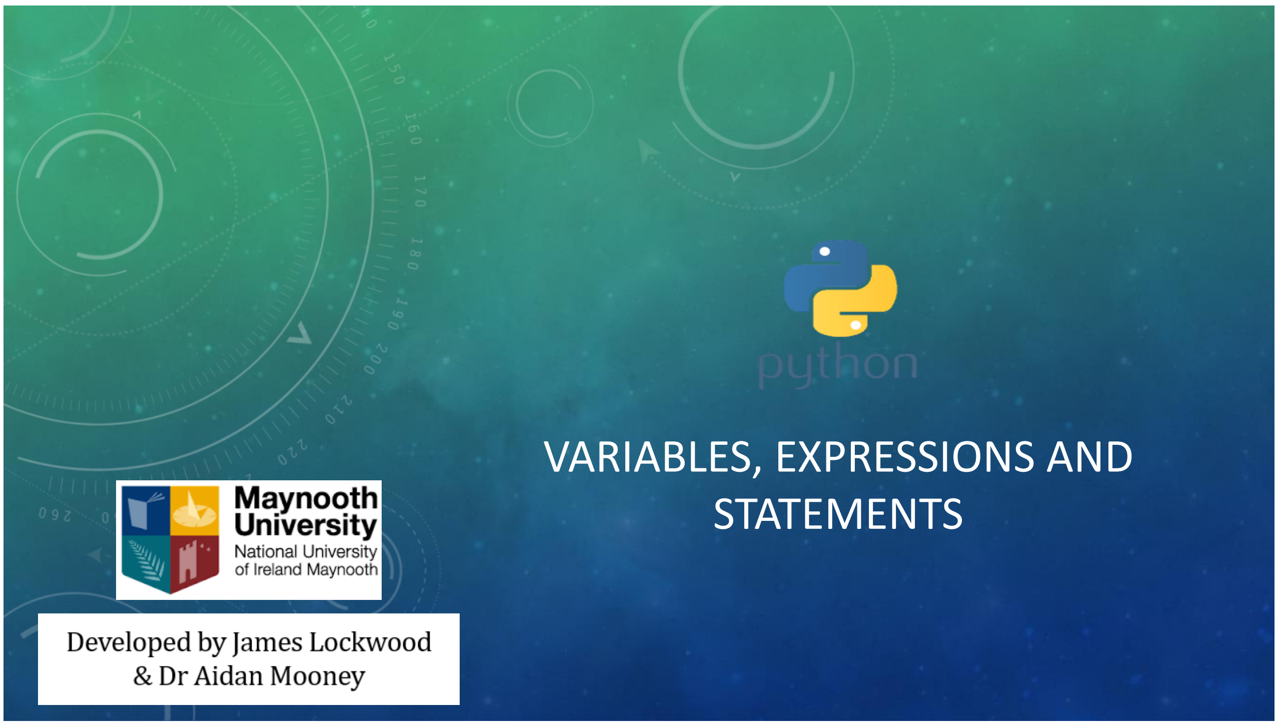


EXERCISE

- Type the following into a session:

```
print(type(2))
print(type(42.0))
print(type('Hello'))
print(type('2'))
print(type('42.0'))
```

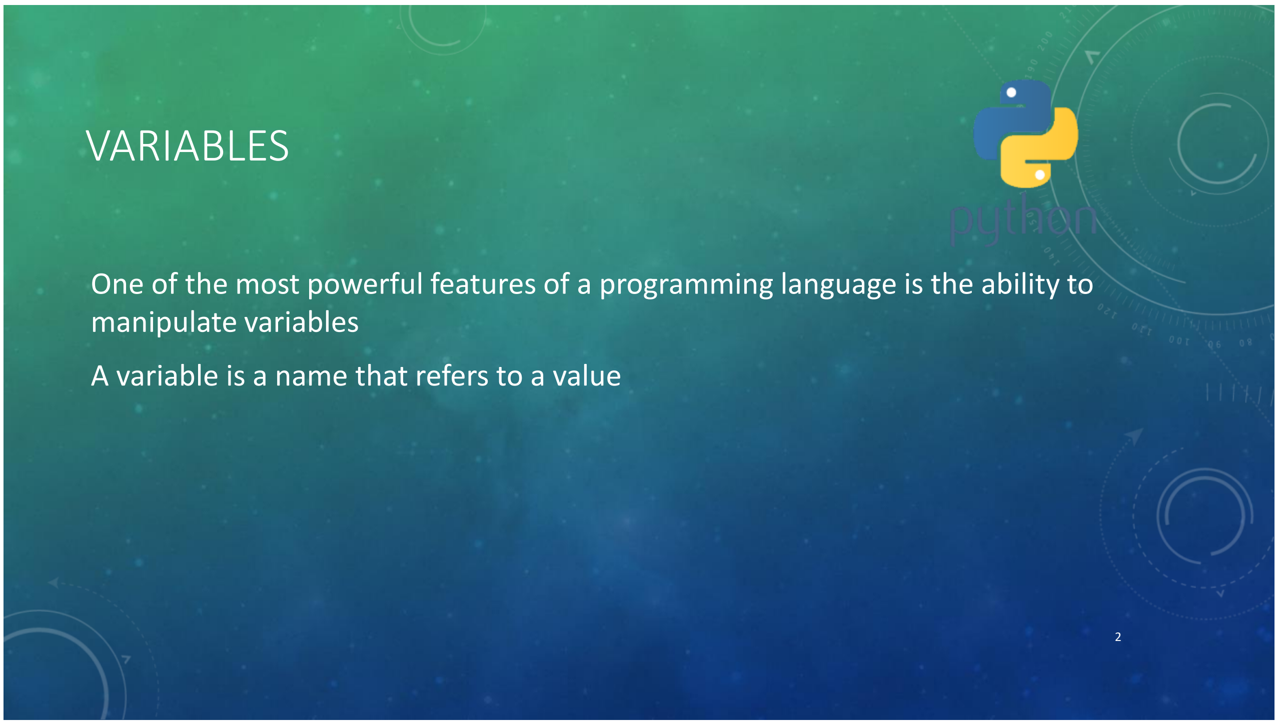
11



VARIABLES, EXPRESSIONS AND STATEMENTS

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



VARIABLES

One of the most powerful features of a programming language is the ability to manipulate variables

A variable is a name that refers to a value

2

ASSIGNMENT

An assignment creates a new variable and gives it a value

```
message = "What's Up Doc?"
x = 42
pi = 3.1415926
```

Unlike in maths, "=" in Python doesn't really mean equals. What it means is put the value on the right-hand side (e.g. 42) into the variable (e.g. x). This is what we mean by "assigning".

3

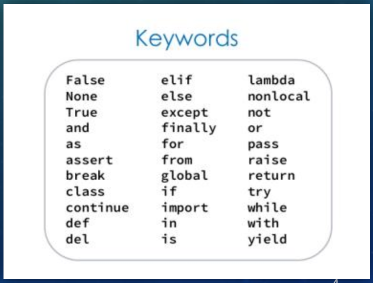
VARIABLE NAMES

Programmers choose variable names that are meaningful

- They document what the variable is used for

Some rules about variable names:

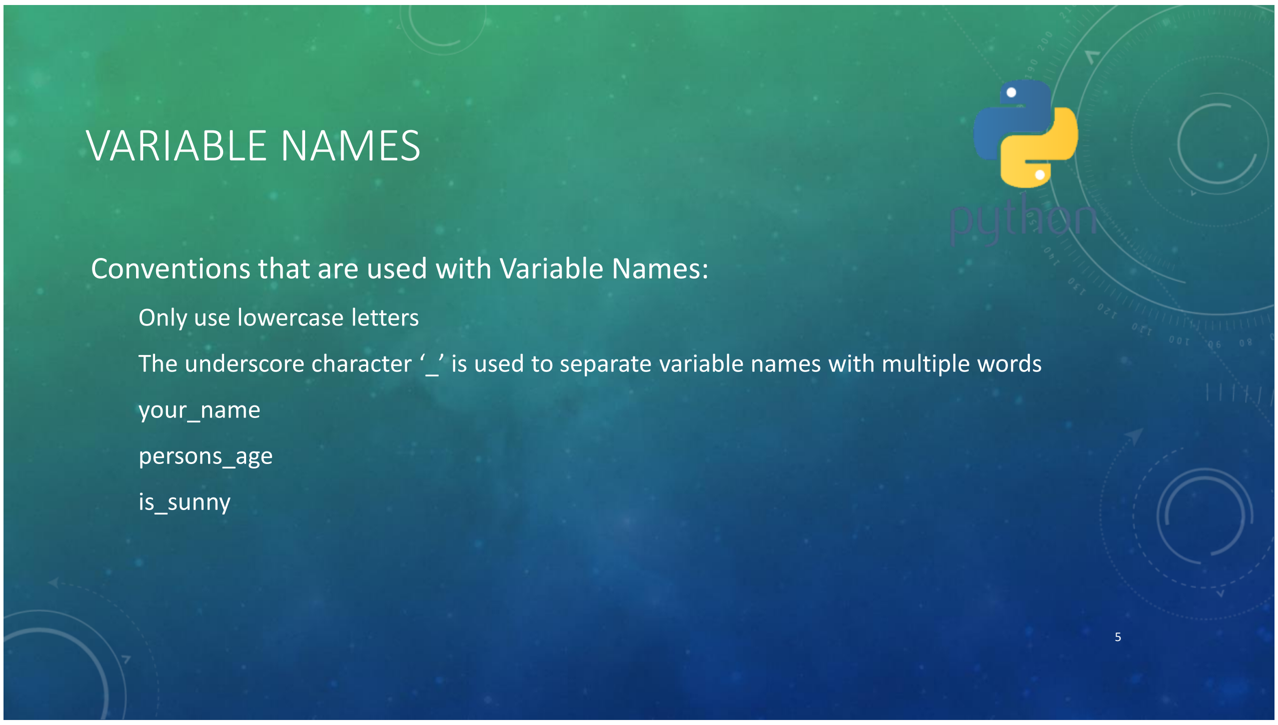
- As long as you like
- Contain both letters and numbers
- Can't begin with a number
- Can't be a Keyword



Keywords

False	elif	lambda
None	else	nonlocal
True	except	not
and	finally	or
as	for	pass
assert	from	raise
break	global	return
class	if	try
continue	import	while
def	in	with
del	is	yield

4

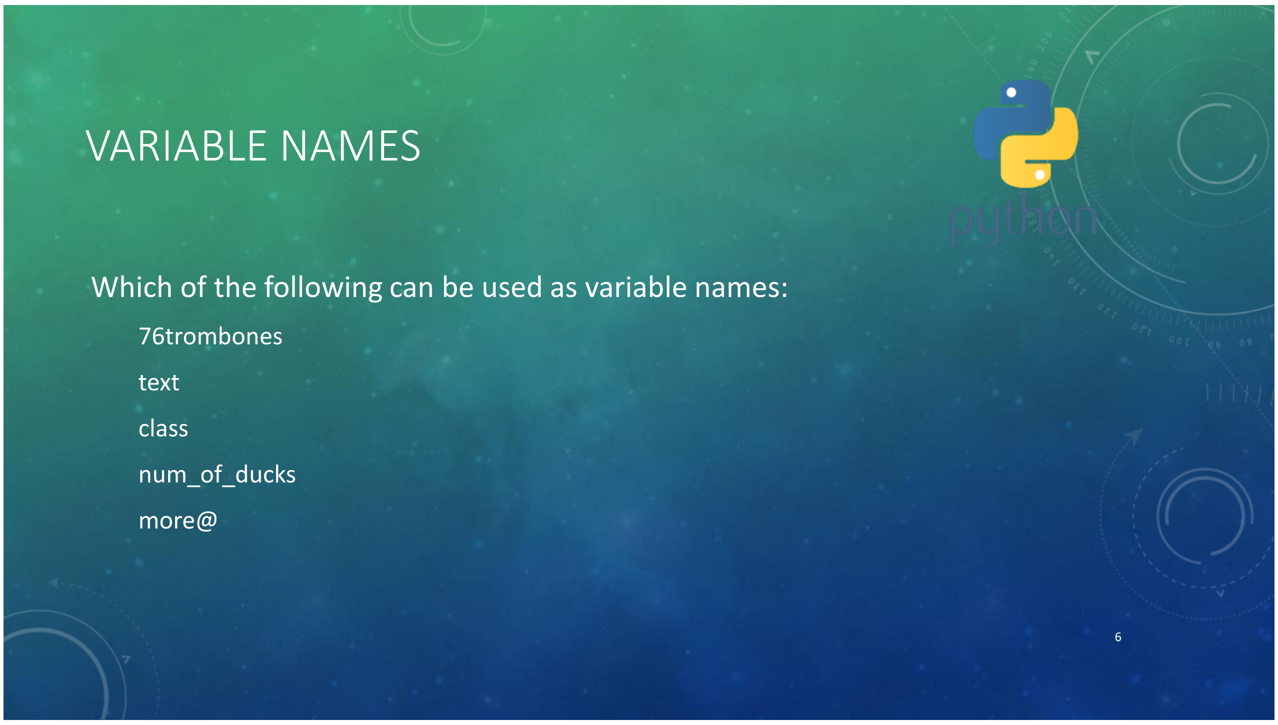


VARIABLE NAMES

Conventions that are used with Variable Names:

- Only use lowercase letters
- The underscore character '_' is used to separate variable names with multiple words
- your_name
- persons_age
- is_sunny

5

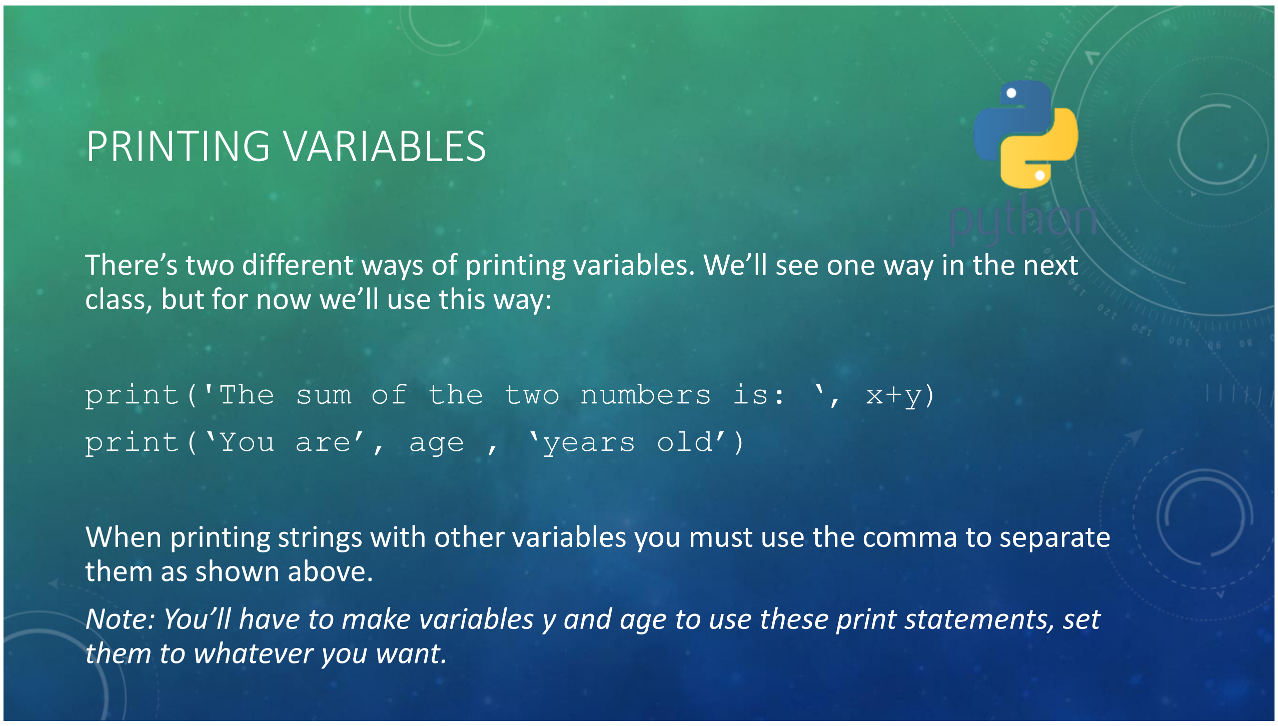


VARIABLE NAMES

Which of the following can be used as variable names:

- 76trombones
- text
- class
- num_of_ducks
- more@

6



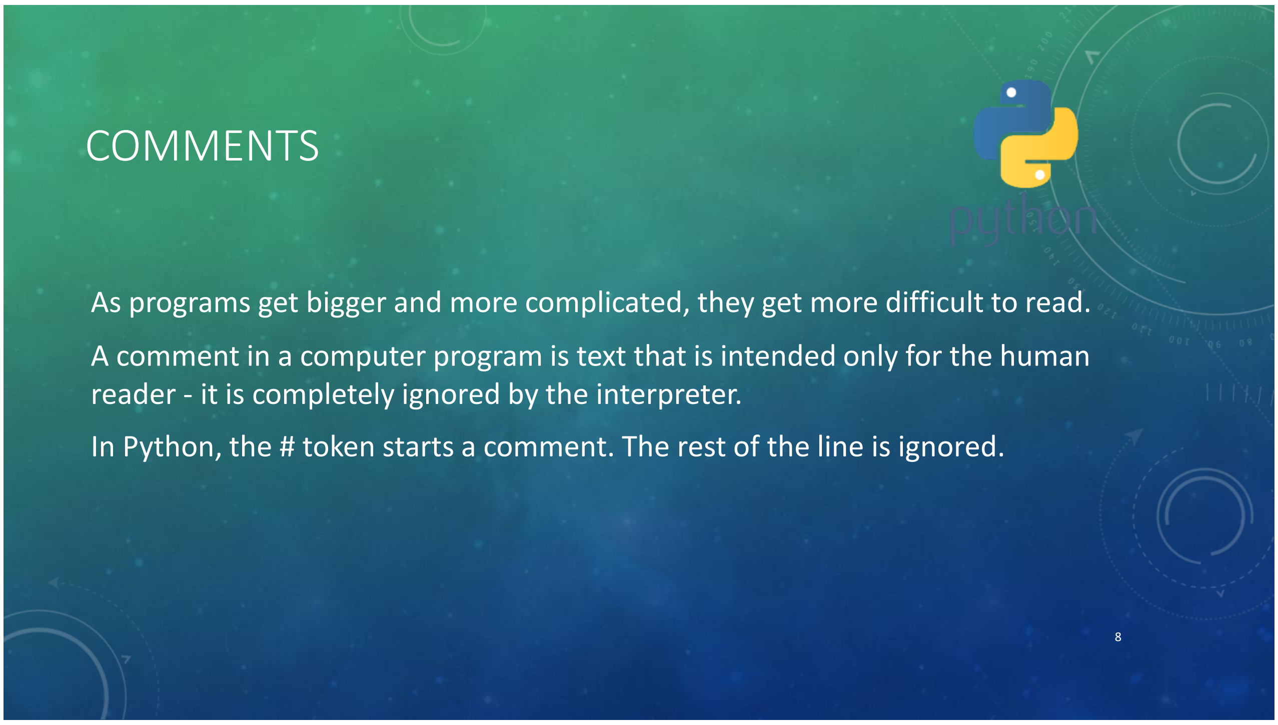
PRINTING VARIABLES

There’s two different ways of printing variables. We’ll see one way in the next class, but for now we’ll use this way:

```
print('The sum of the two numbers is: ‘, x+y)
print(‘You are’, age , ‘years old’)
```

When printing strings with other variables you must use the comma to separate them as shown above.

Note: You’ll have to make variables y and age to use these print statements, set them to whatever you want.



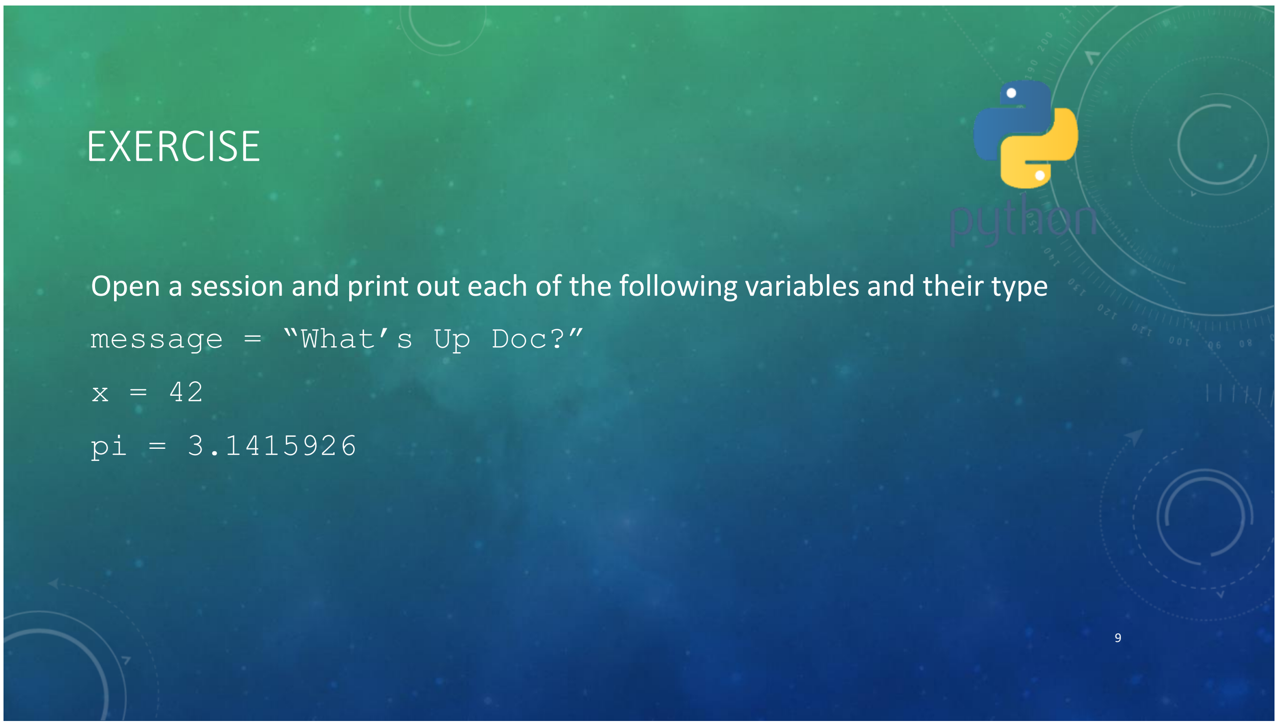
COMMENTS

As programs get bigger and more complicated, they get more difficult to read.

A comment in a computer program is text that is intended only for the human reader - it is completely ignored by the interpreter.

In Python, the # token starts a comment. The rest of the line is ignored.

8

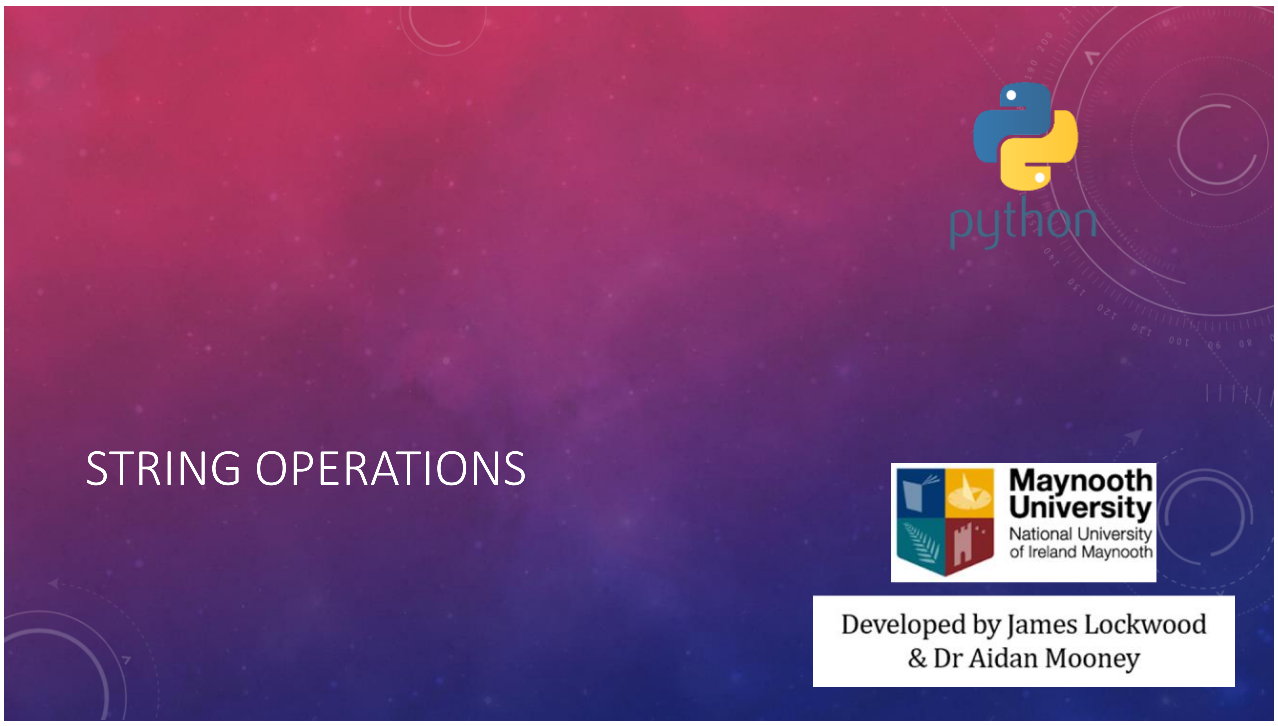


EXERCISE

Open a session and print out each of the following variables and their type

```
message = "What's Up Doc?"
x = 42
pi = 3.1415926
```

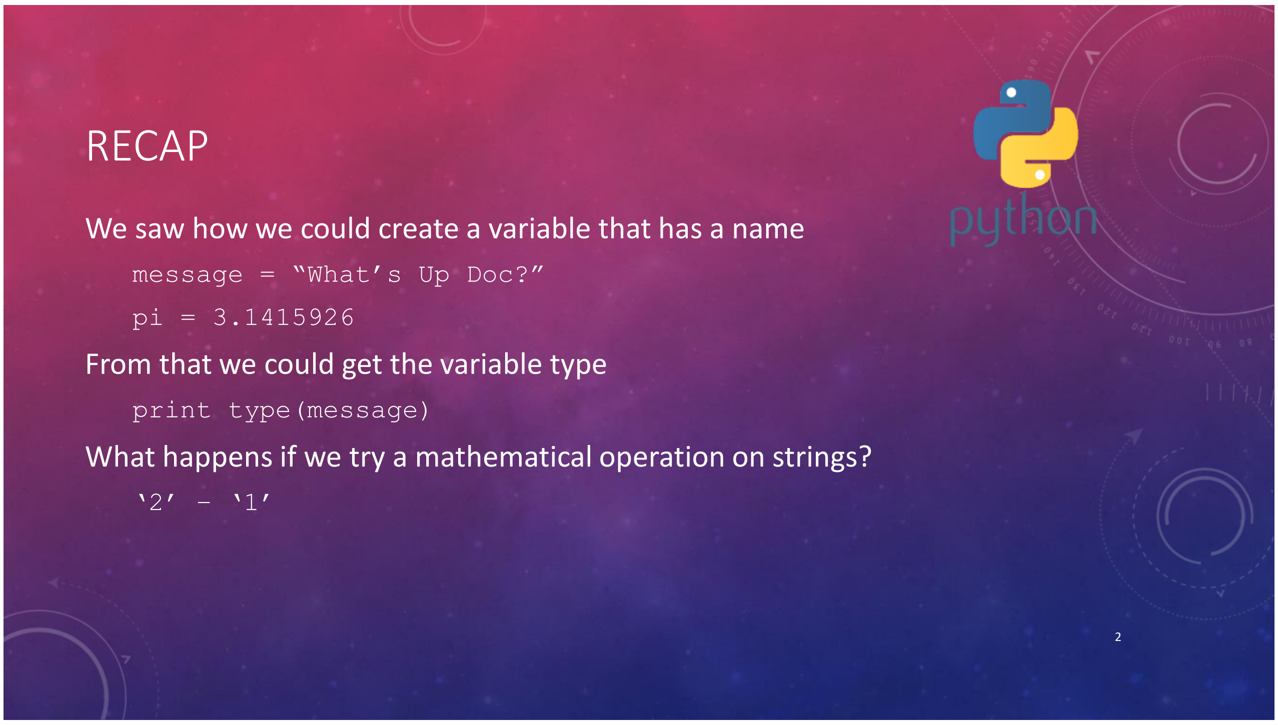
9



STRING OPERATIONS

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney



RECAP

We saw how we could create a variable that has a name

```
message = "What's Up Doc?"
pi = 3.1415926
```

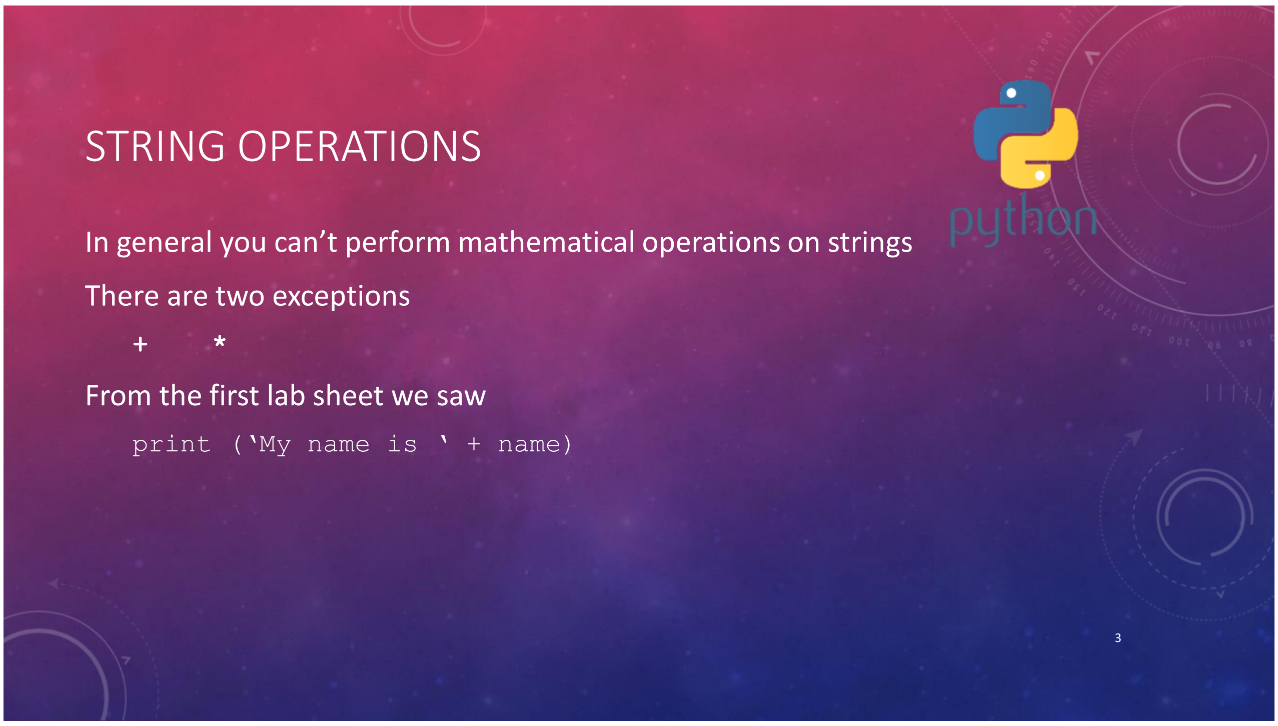
From that we could get the variable type

```
print type(message)
```

What happens if we try a mathematical operation on strings?

```
'2' - '1'
```

2



STRING OPERATIONS

In general you can't perform mathematical operations on strings

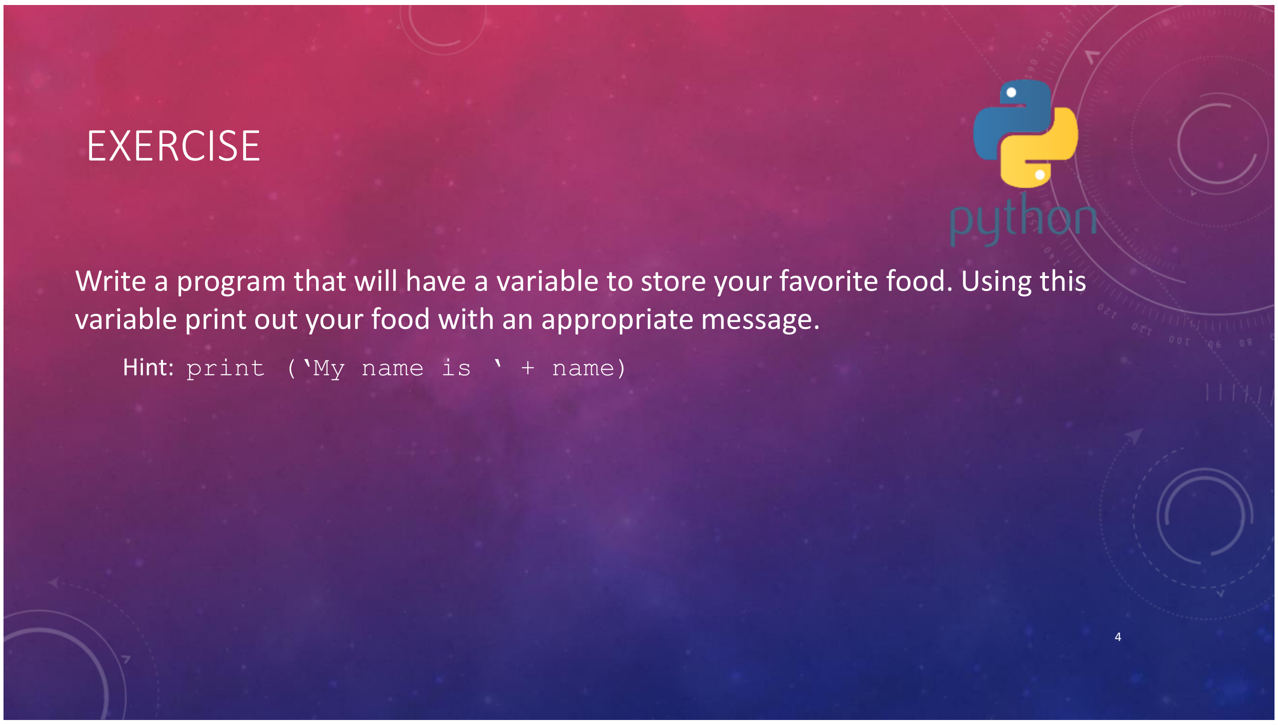
There are two exceptions

+ *

From the first lab sheet we saw

```
print ('My name is ' + name)
```

3

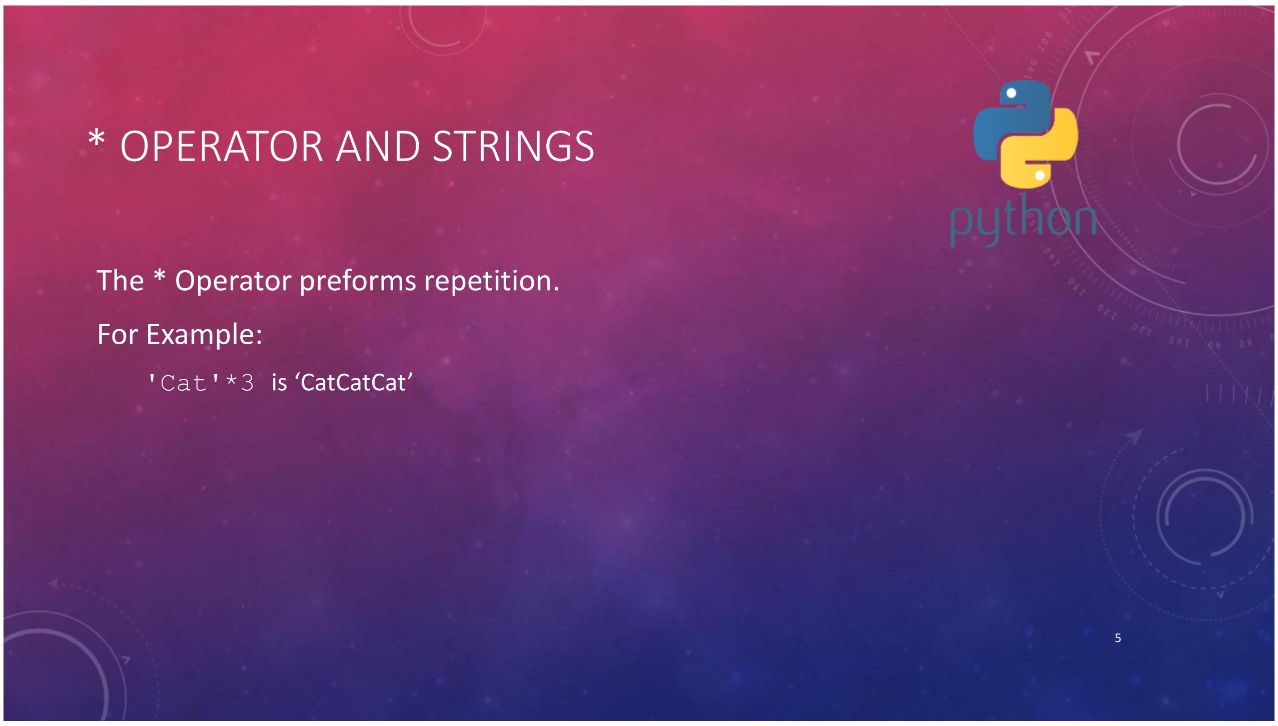


EXERCISE

Write a program that will have a variable to store your favorite food. Using this variable print out your food with an appropriate message.

Hint: `print ('My name is ' + name)`

4



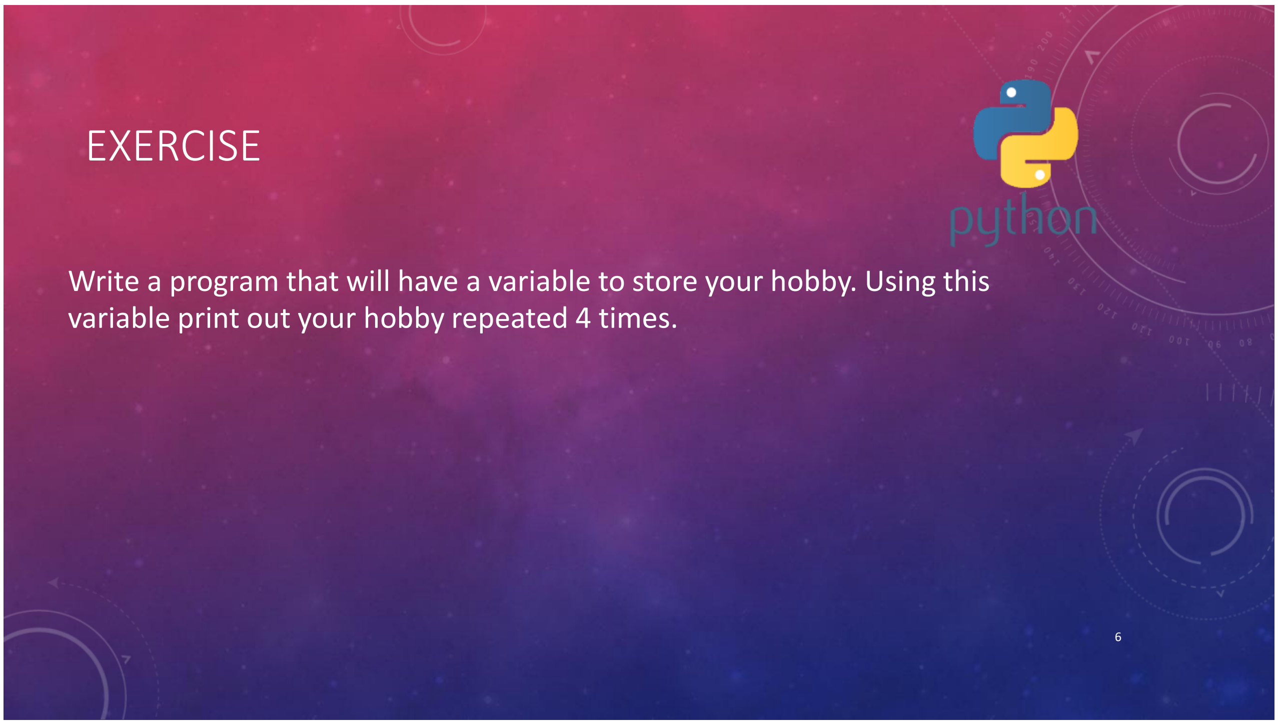
* OPERATOR AND STRINGS

The * Operator preforms repetition.

For Example:

`'Cat'*3` is 'CatCatCat'

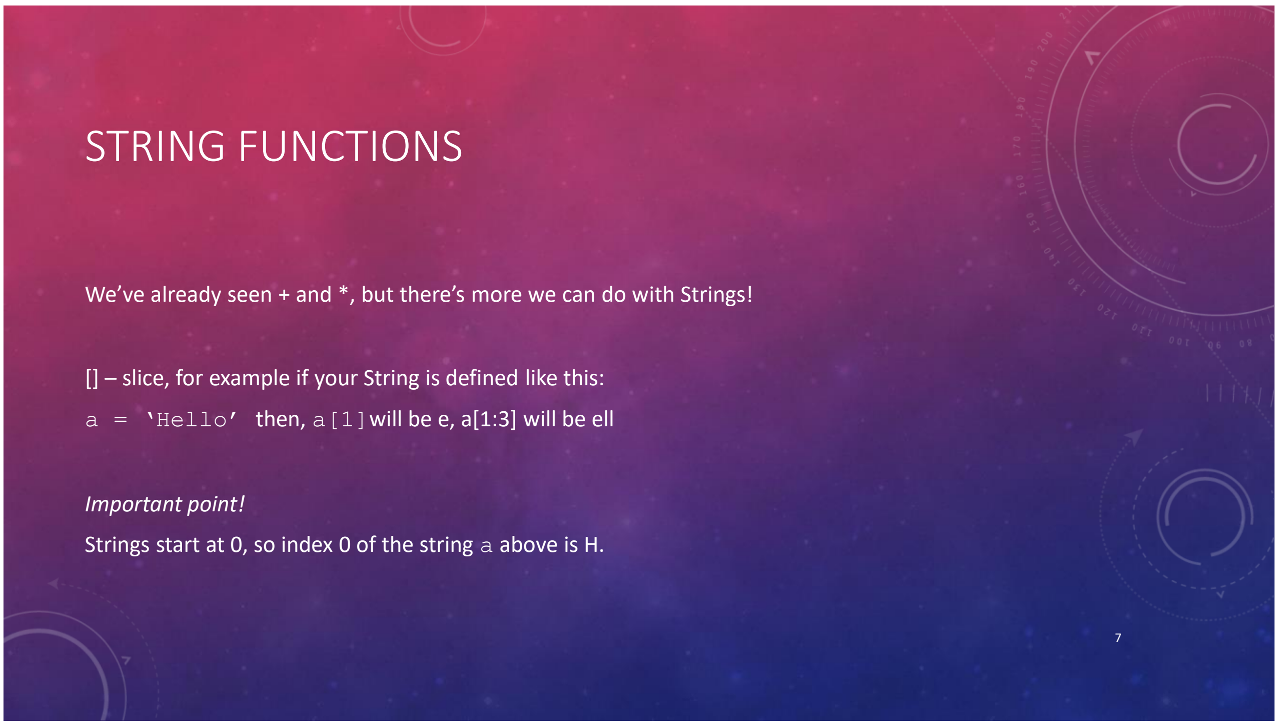
5



EXERCISE

Write a program that will have a variable to store your hobby. Using this variable print out your hobby repeated 4 times.

6



STRING FUNCTIONS

We've already seen + and *, but there's more we can do with Strings!

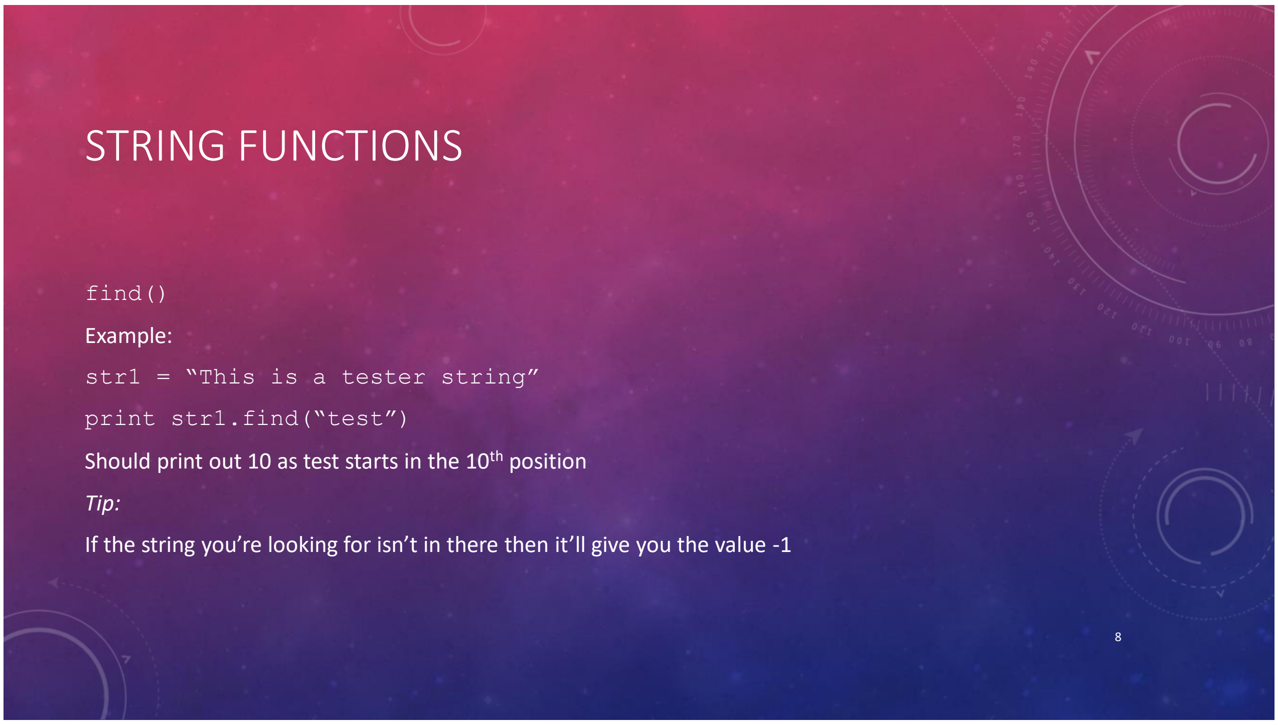
[] – slice, for example if your String is defined like this:

`a = 'Hello'` then, `a[1]` will be e, a[1:3] will be ell

Important point!

Strings start at 0, so index 0 of the string `a` above is H.

7



STRING FUNCTIONS

`find()`

Example:

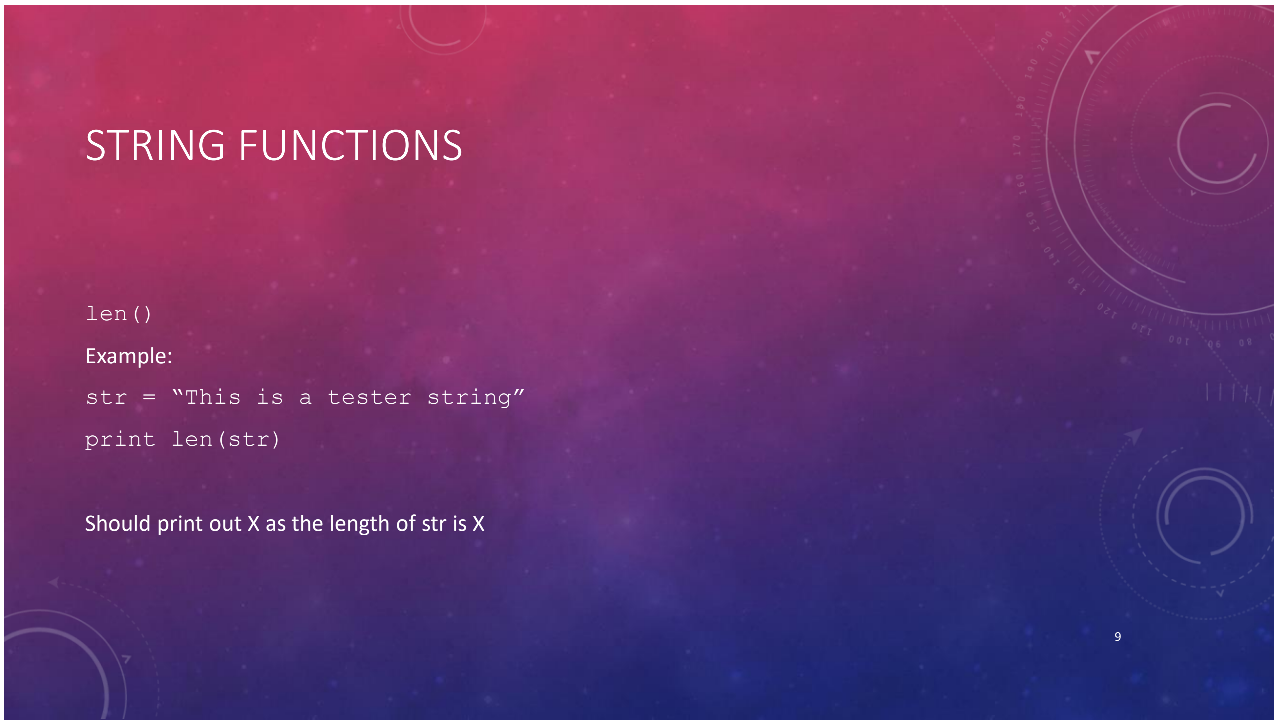
```
str1 = "This is a tester string"
print str1.find("test")
```

Should print out 10 as test starts in the 10th position

Tip:

If the string you're looking for isn't in there then it'll give you the value -1

8



STRING FUNCTIONS

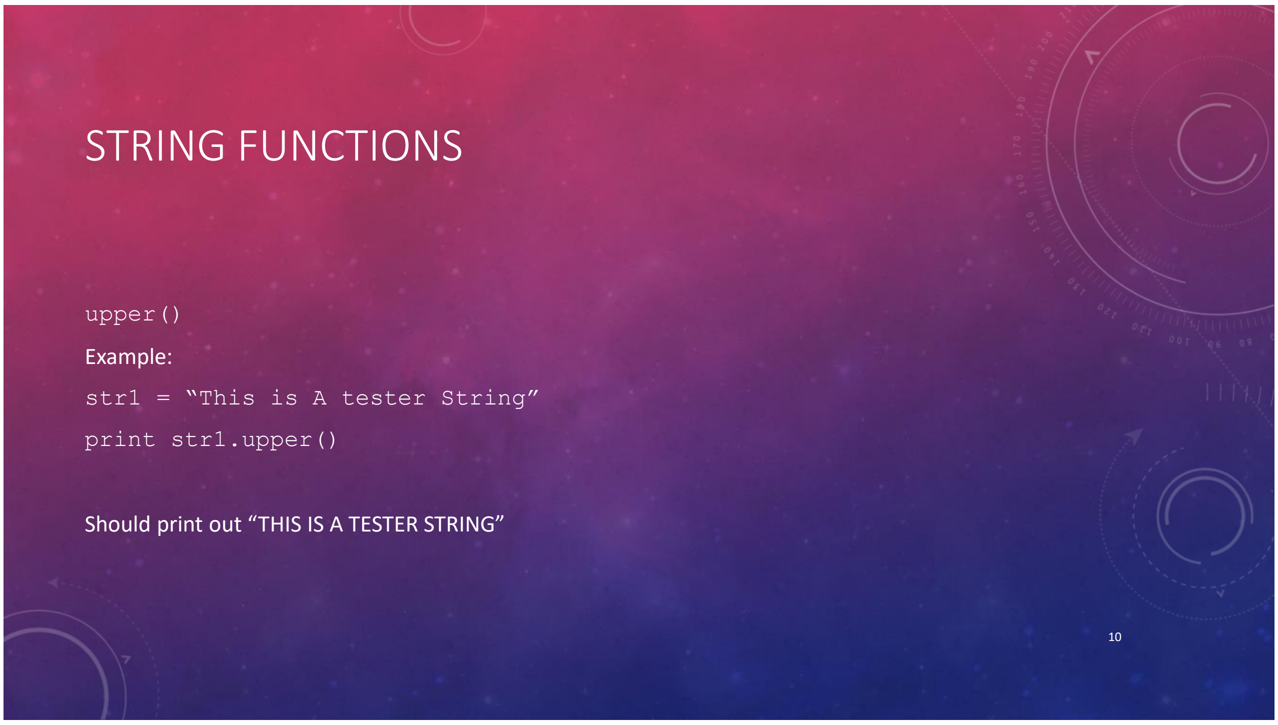
```
len()
```

Example:

```
str = "This is a tester string"
print len(str)
```

Should print out X as the length of str is X

9



STRING FUNCTIONS

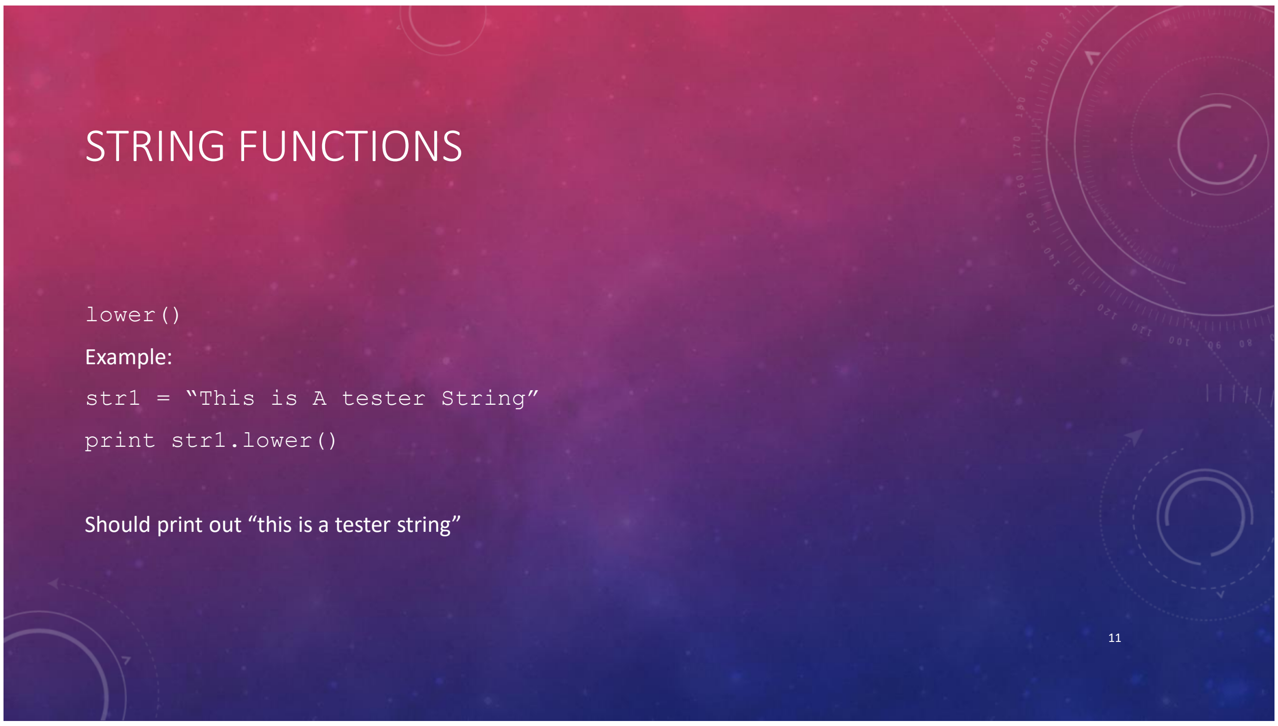
```
upper()
```

Example:

```
str1 = "This is A tester String"
print str1.upper()
```

Should print out "THIS IS A TESTER STRING"

10



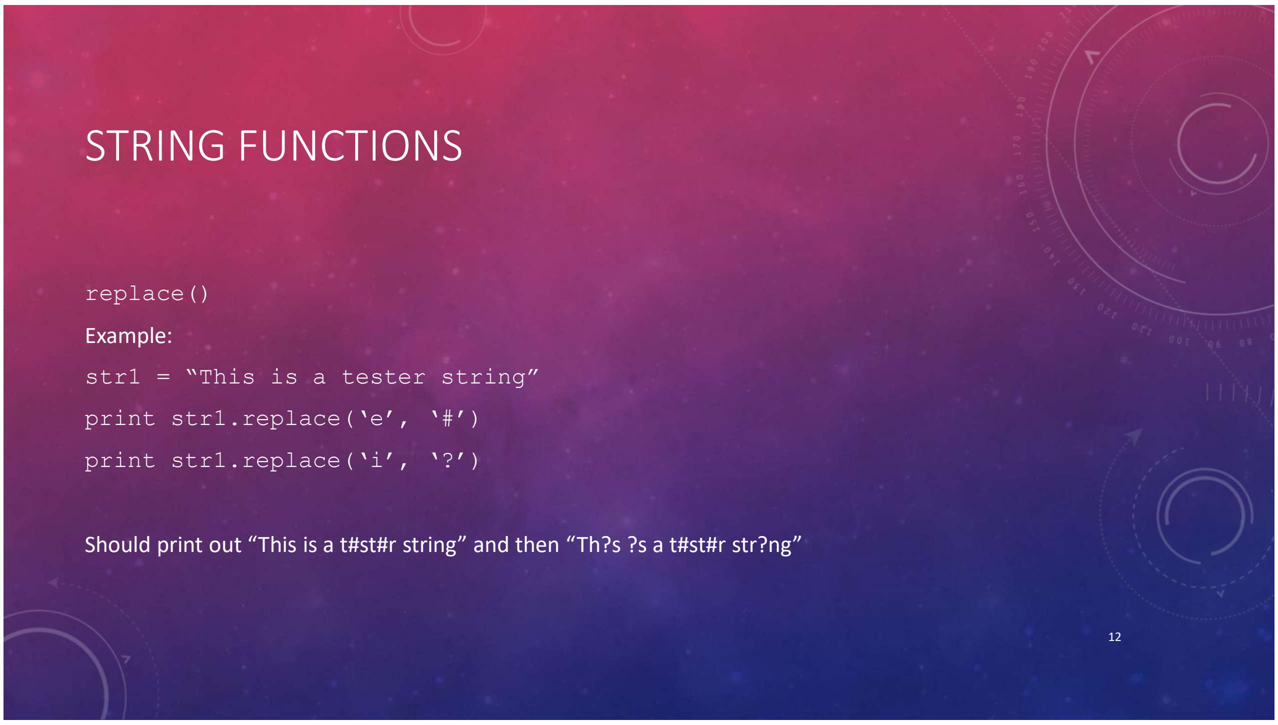
STRING FUNCTIONS

```
lower()
```

Example:

```
str1 = "This is A tester String"
print str1.lower()
```

Should print out “this is a tester string”



STRING FUNCTIONS

```
replace()
```

Example:

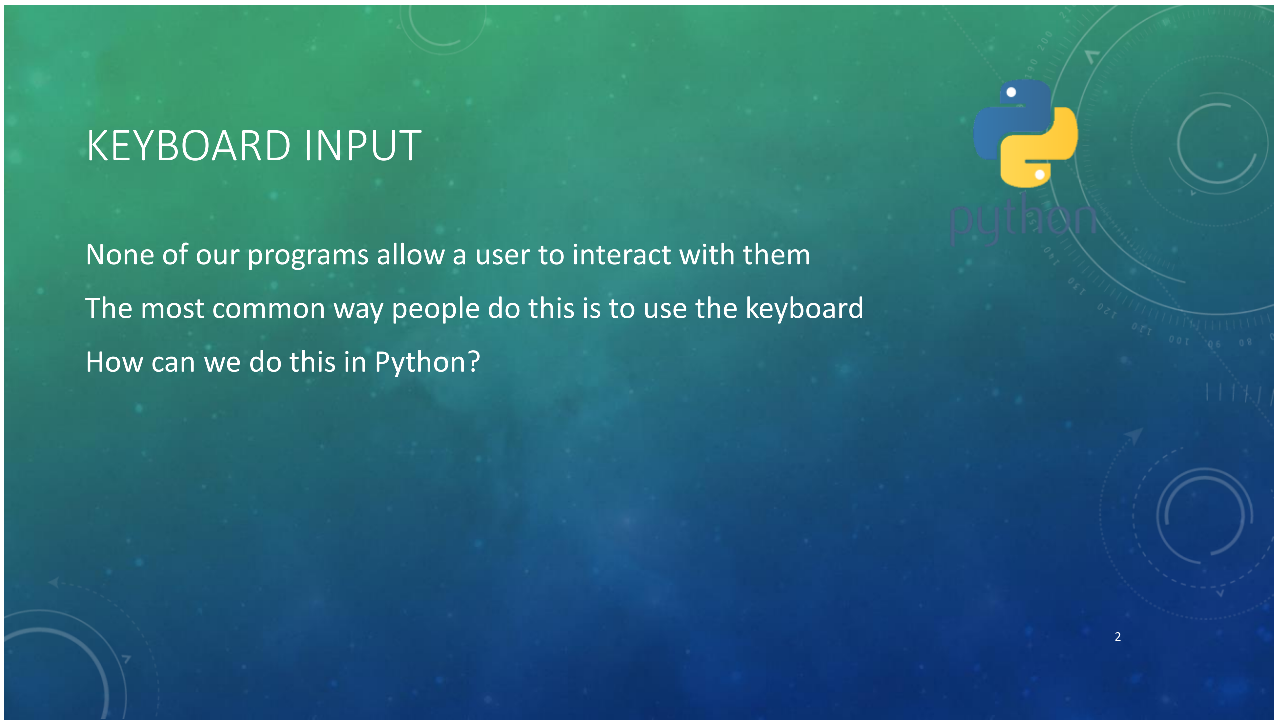
```
str1 = "This is a tester string"
print str1.replace('e', '#')
print str1.replace('i', '?')
```

Should print out “This is a t#st#r string” and then “Th?s ?s a t#st#r str?ng”

KEYBOARD INPUT

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



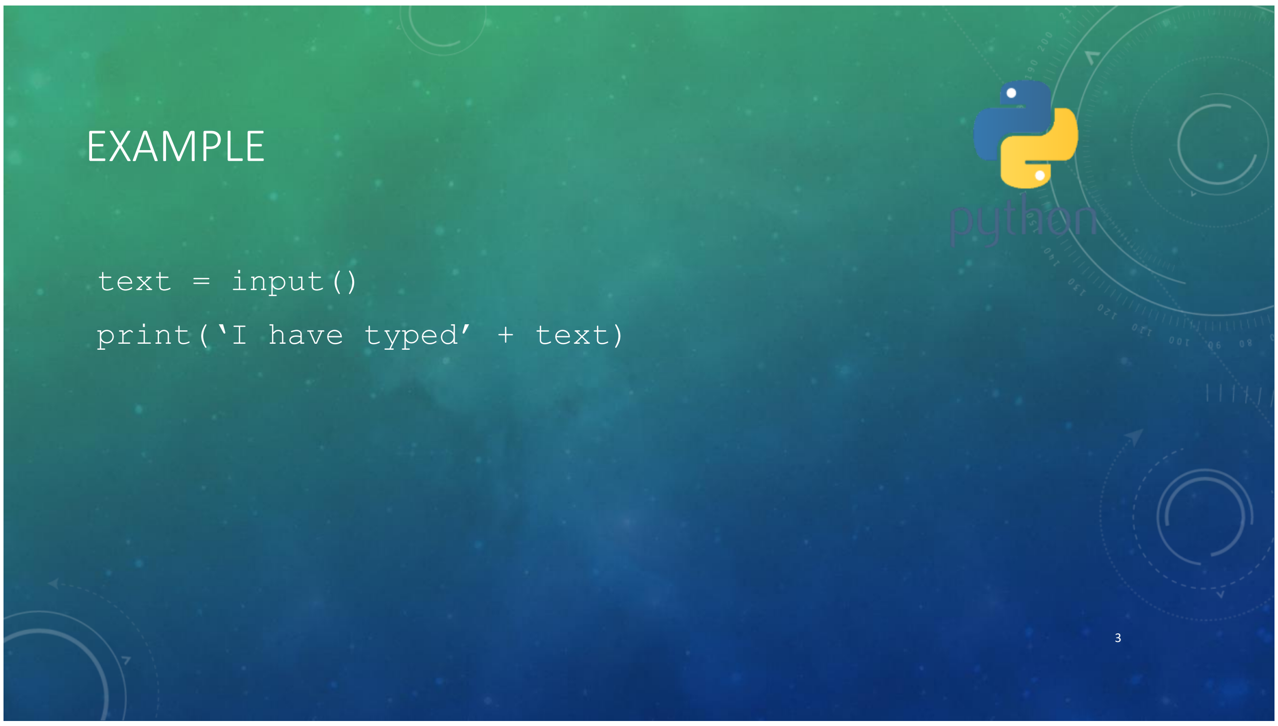
KEYBOARD INPUT

None of our programs allow a user to interact with them

The most common way people do this is to use the keyboard

How can we do this in Python?

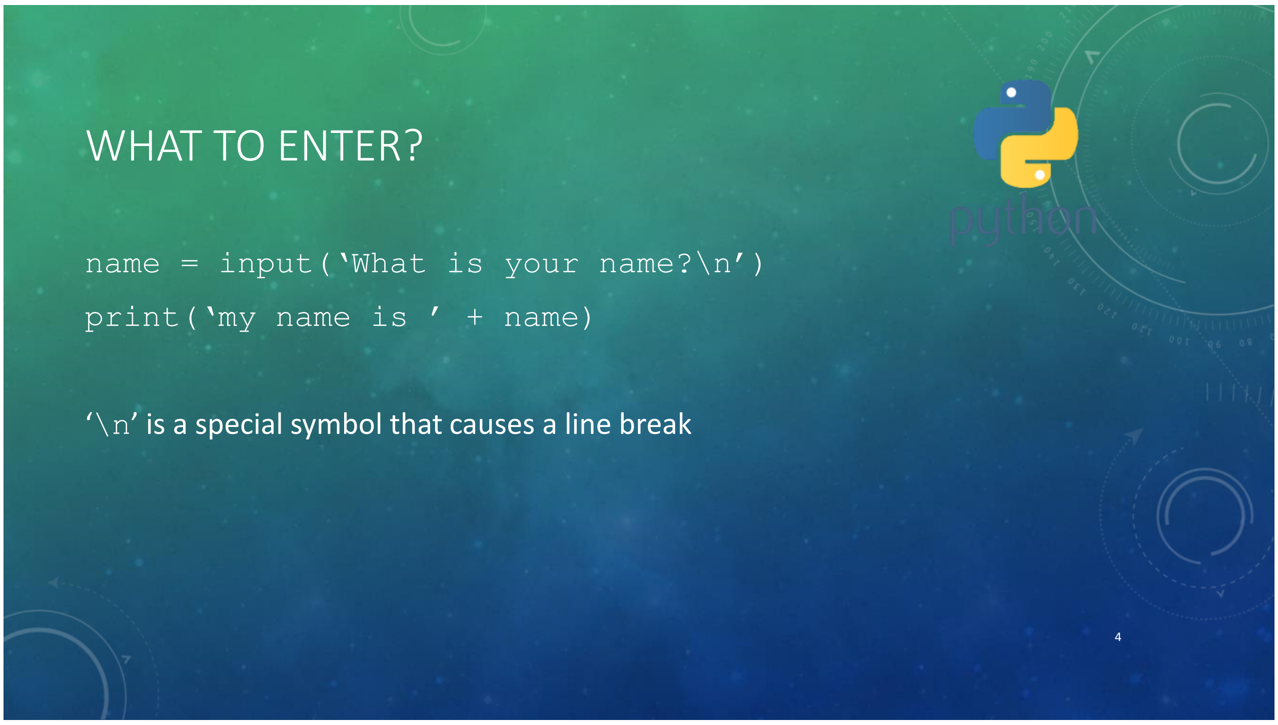
2



EXAMPLE

```
text = input()
print('I have typed' + text)
```

3

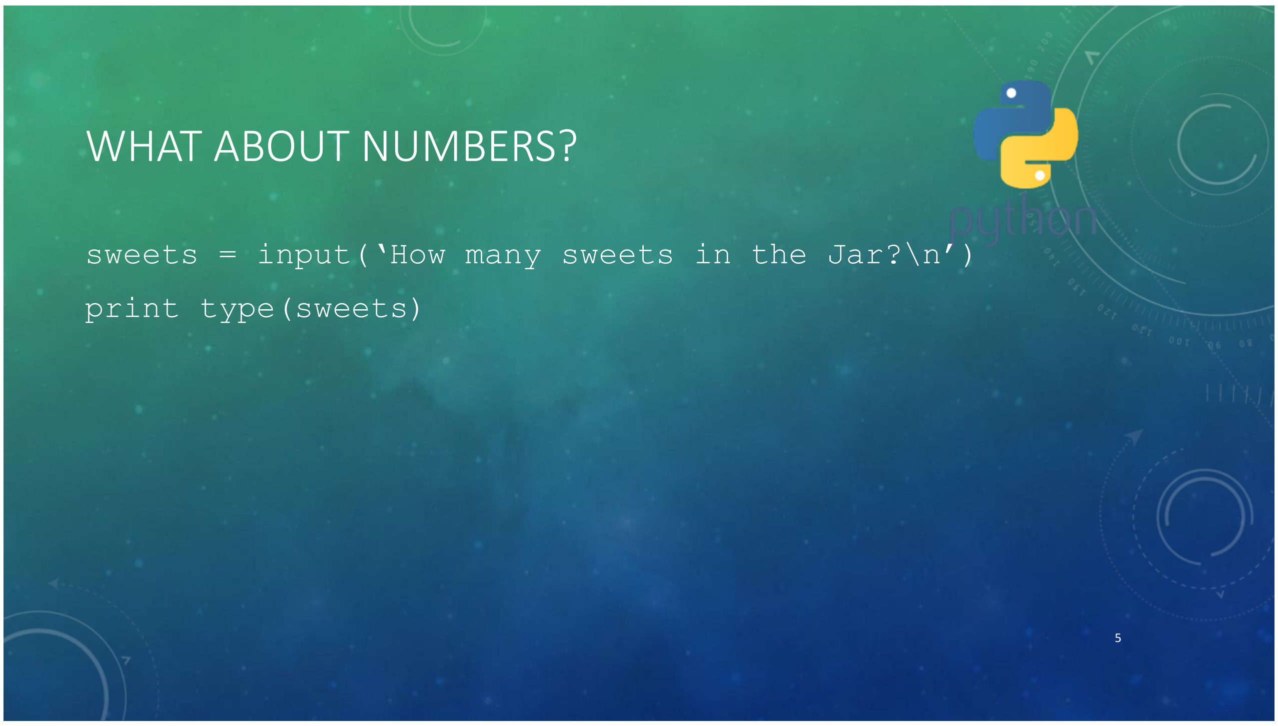


WHAT TO ENTER?

```
name = input('What is your name?\n')
print('my name is ' + name)
```

'`\n`' is a special symbol that causes a line break

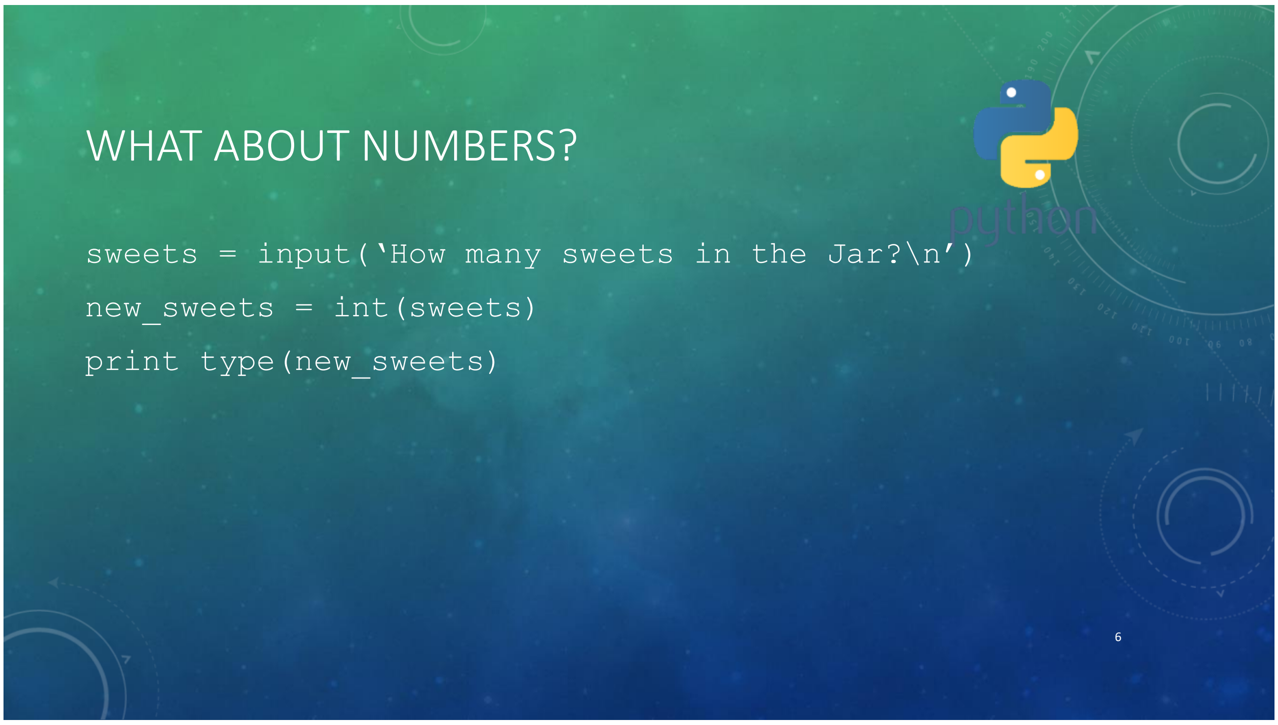
4



WHAT ABOUT NUMBERS?

```
sweets = input('How many sweets in the Jar?\n')
print type(sweets)
```

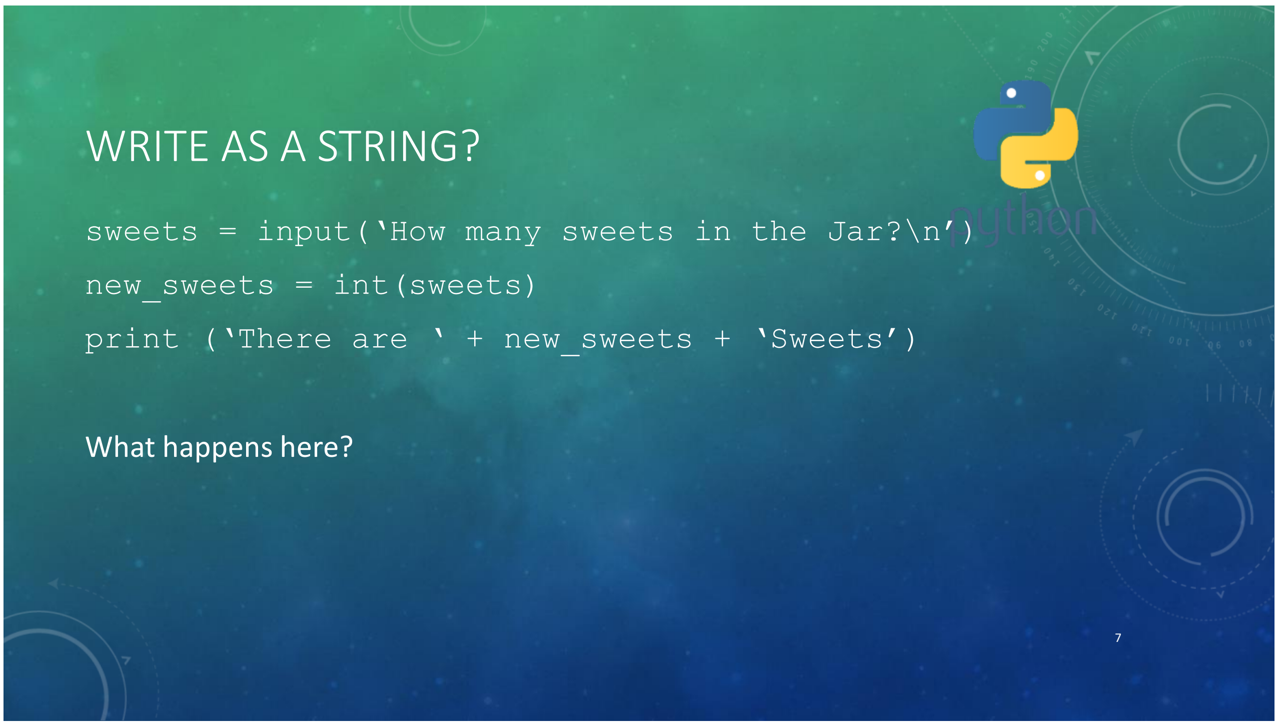
5



WHAT ABOUT NUMBERS?

```
sweets = input('How many sweets in the Jar?\n')
new_sweets = int(sweets)
print type(new_sweets)
```

6

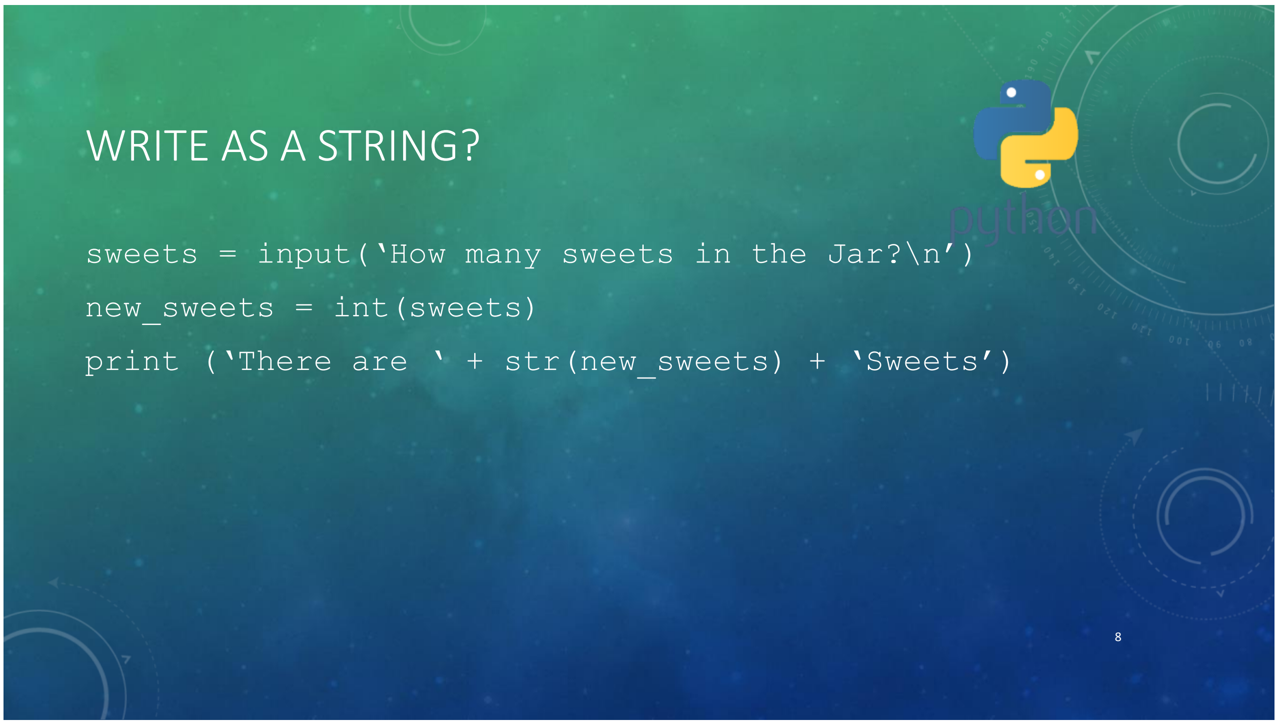


WRITE AS A STRING?

```
sweets = input('How many sweets in the Jar?\n')
new_sweets = int(sweets)
print ('There are ' + new_sweets + 'Sweets')
```

What happens here?

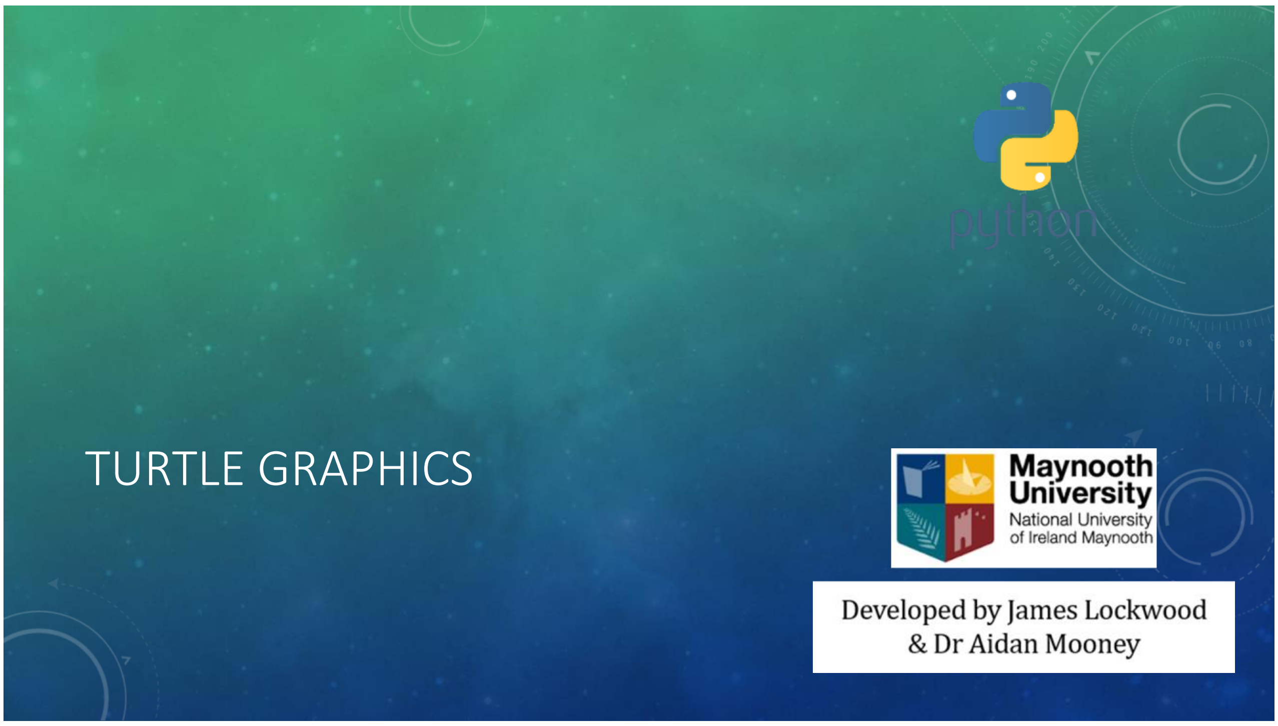
7



WRITE AS A STRING?

```
sweets = input('How many sweets in the Jar?\n')
new_sweets = int(sweets)
print ('There are ' + str(new_sweets) + 'Sweets')
```

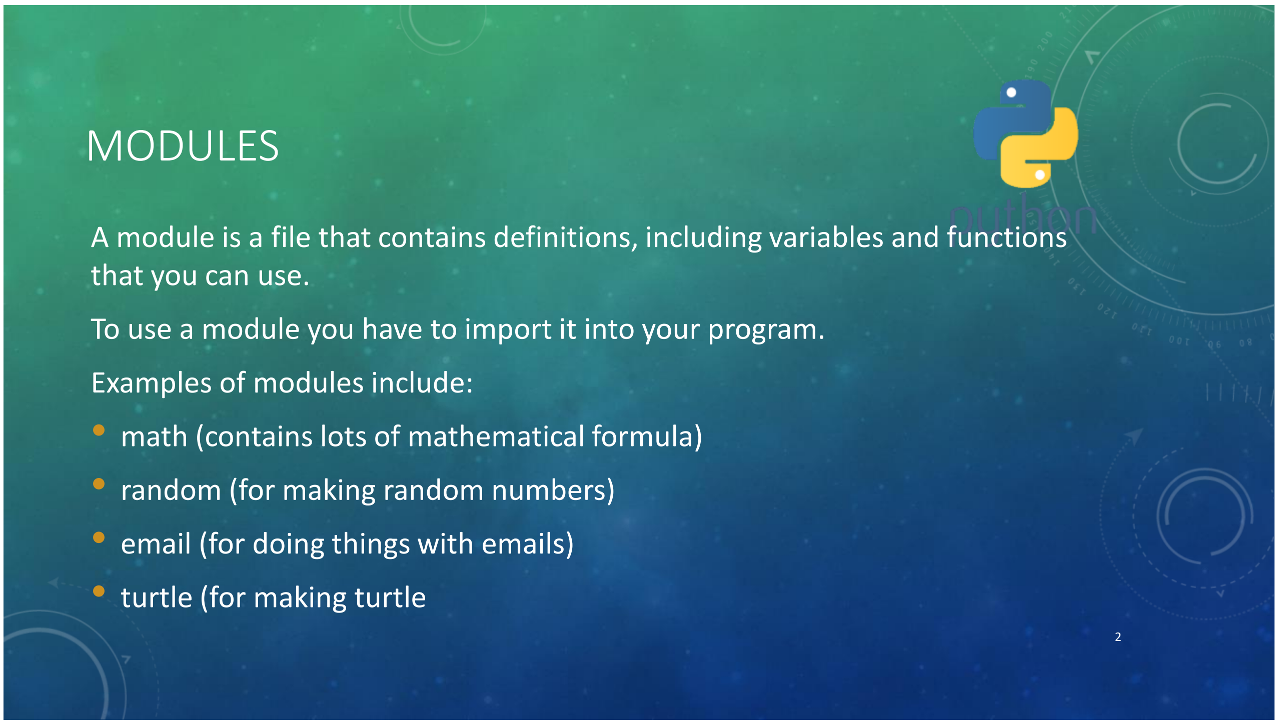
8



TURTLE GRAPHICS

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood & Dr Aidan Mooney



MODULES

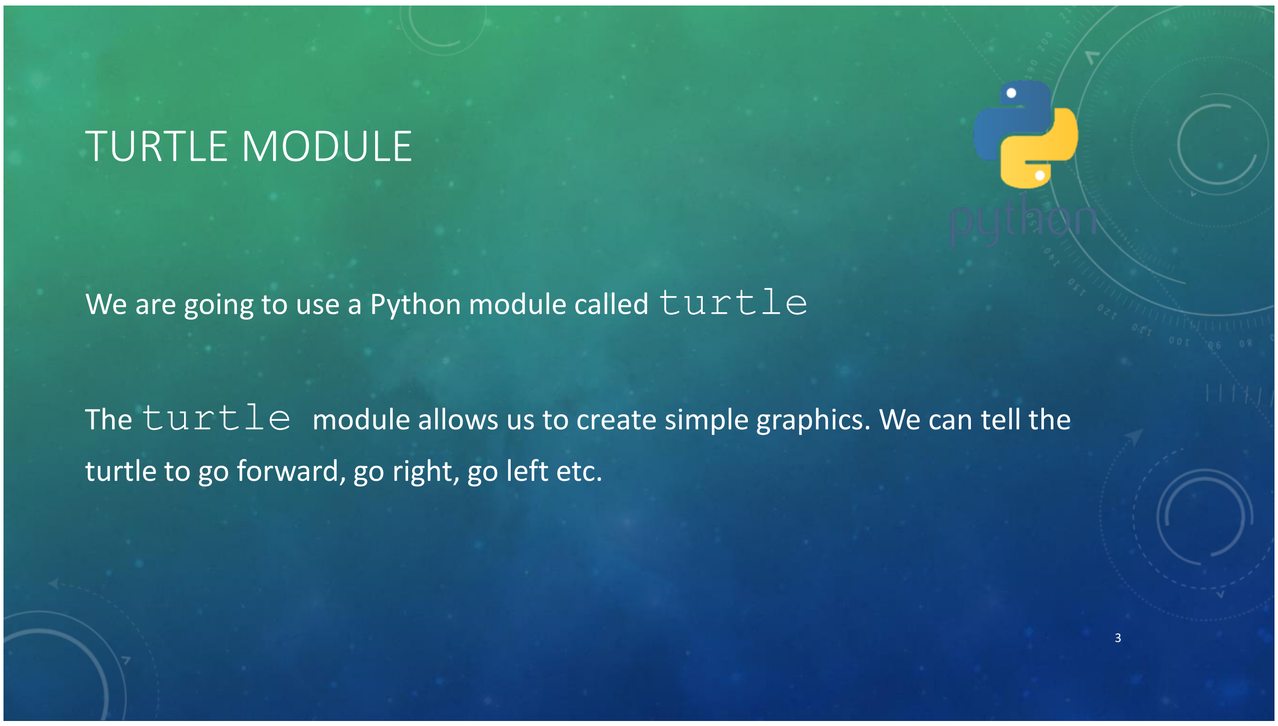
A module is a file that contains definitions, including variables and functions that you can use.

To use a module you have to import it into your program.

Examples of modules include:

- math (contains lots of mathematical formula)
- random (for making random numbers)
- email (for doing things with emails)
- turtle (for making turtle

2

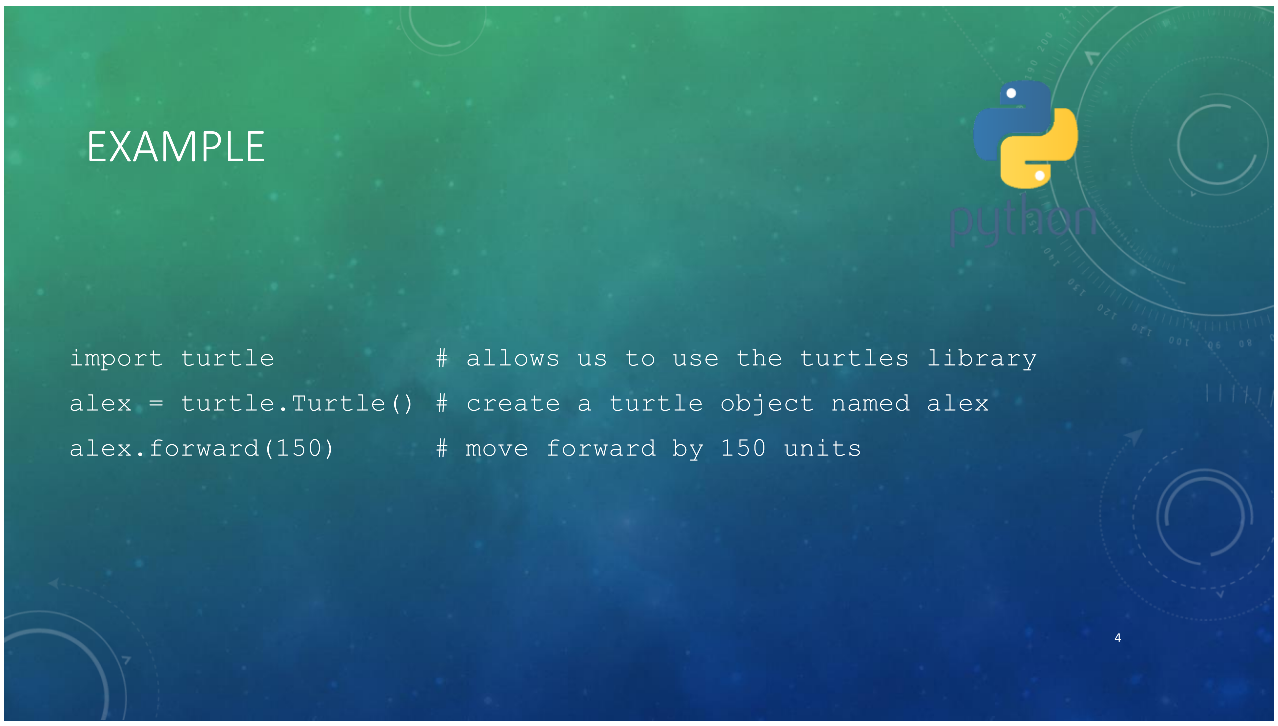


TURTLE MODULE

We are going to use a Python module called `turtle`

The `turtle` module allows us to create simple graphics. We can tell the turtle to go forward, go right, go left etc.

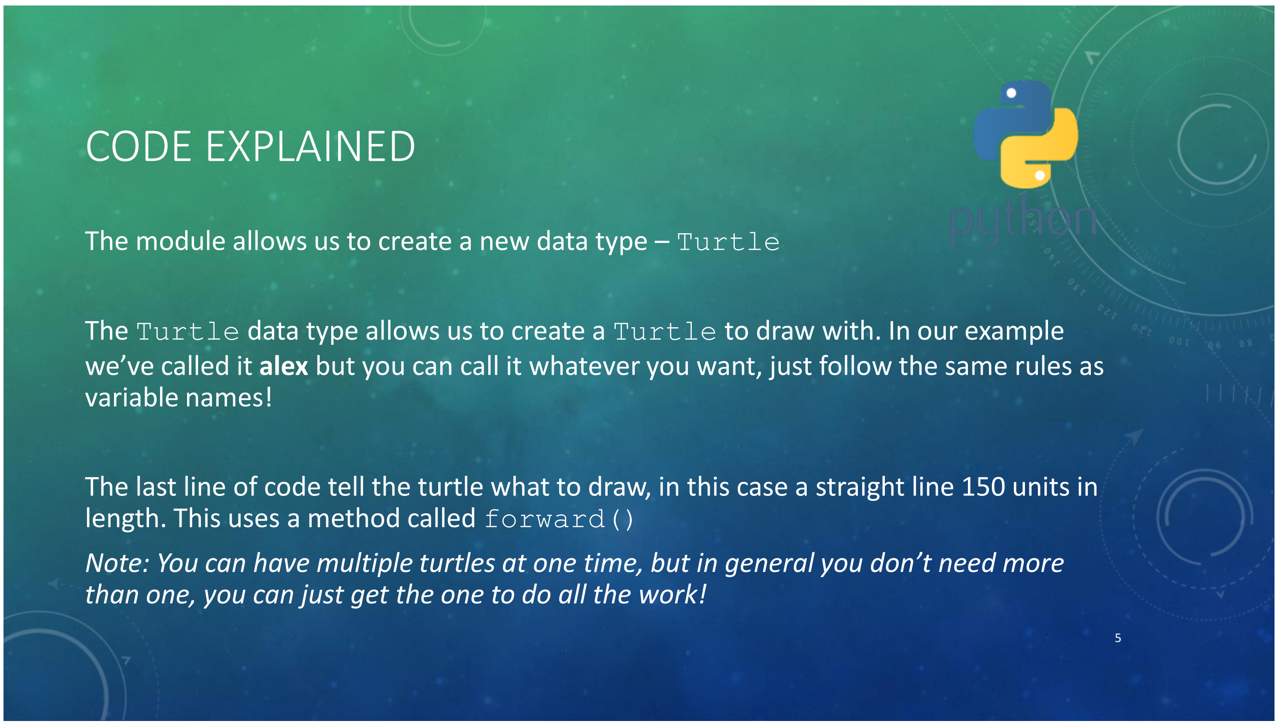
3



EXAMPLE

```
import turtle          # allows us to use the turtles library
alex = turtle.Turtle() # create a turtle object named alex
alex.forward(150)      # move forward by 150 units
```

4



CODE EXPLAINED

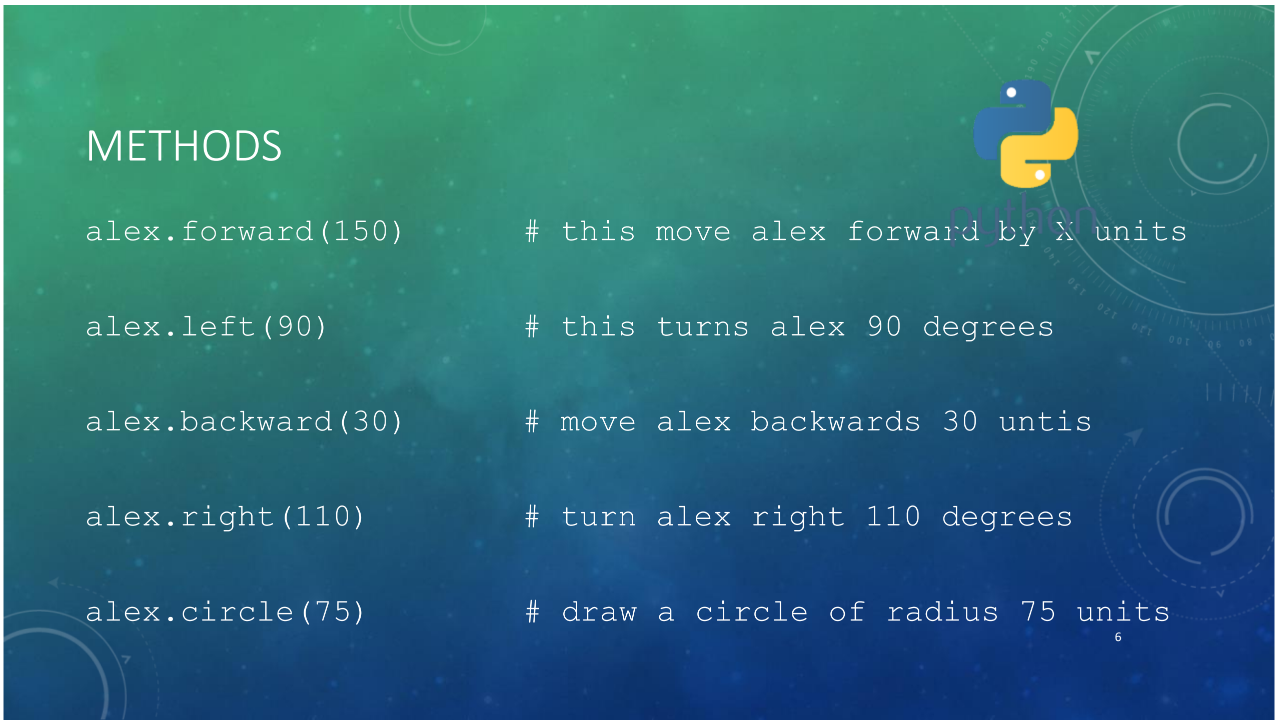
The module allows us to create a new data type – `Turtle`

The `Turtle` data type allows us to create a `Turtle` to draw with. In our example we've called it **alex** but you can call it whatever you want, just follow the same rules as variable names!

The last line of code tell the turtle what to draw, in this case a straight line 150 units in length. This uses a method called `forward()`

Note: You can have multiple turtles at one time, but in general you don't need more than one, you can just get the one to do all the work!

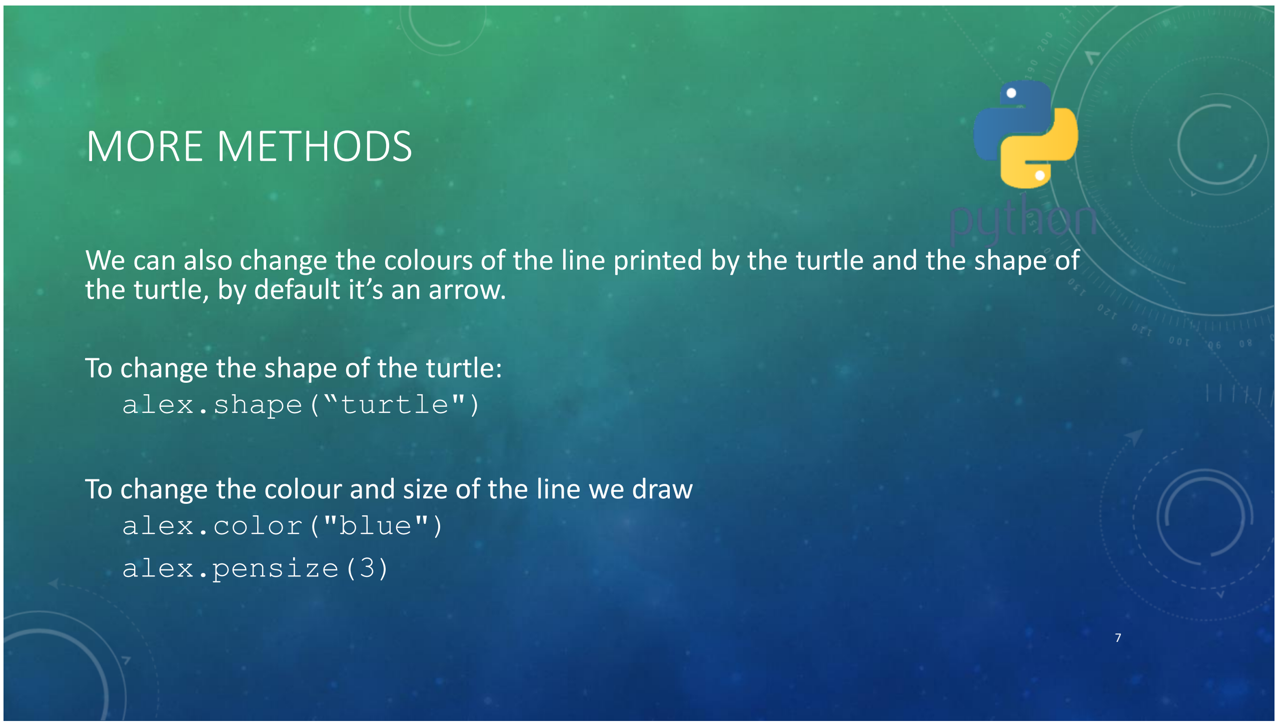
5



METHODS

```
alex.forward(150)      # this move alex forward by X units

alex.left(90)          # this turns alex 90 degrees

alex.backward(30)      # move alex backwards 30 untis

alex.right(110)        # turn alex right 110 degrees

alex.circle(75)        # draw a circle of radius 75 units
```

6



MORE METHODS

We can also change the colours of the line printed by the turtle and the shape of the turtle, by default it’s an arrow.

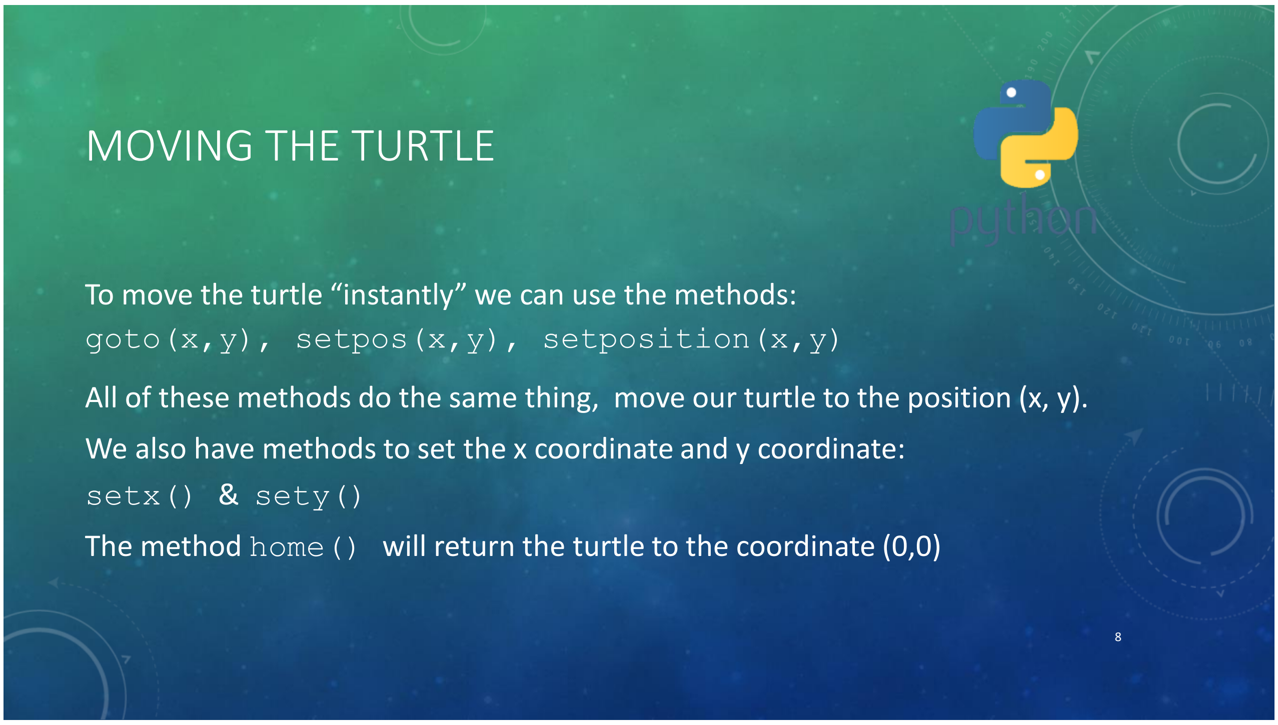
To change the shape of the turtle:

```
   alex.shape(“turtle")
```

To change the colour and size of the line we draw

```
   alex.color("blue")
   alex.pensize(3)
```

7



MOVING THE TURTLE

To move the turtle “instantly” we can use the methods:

`goto(x,y), setpos(x,y), setposition(x,y)`

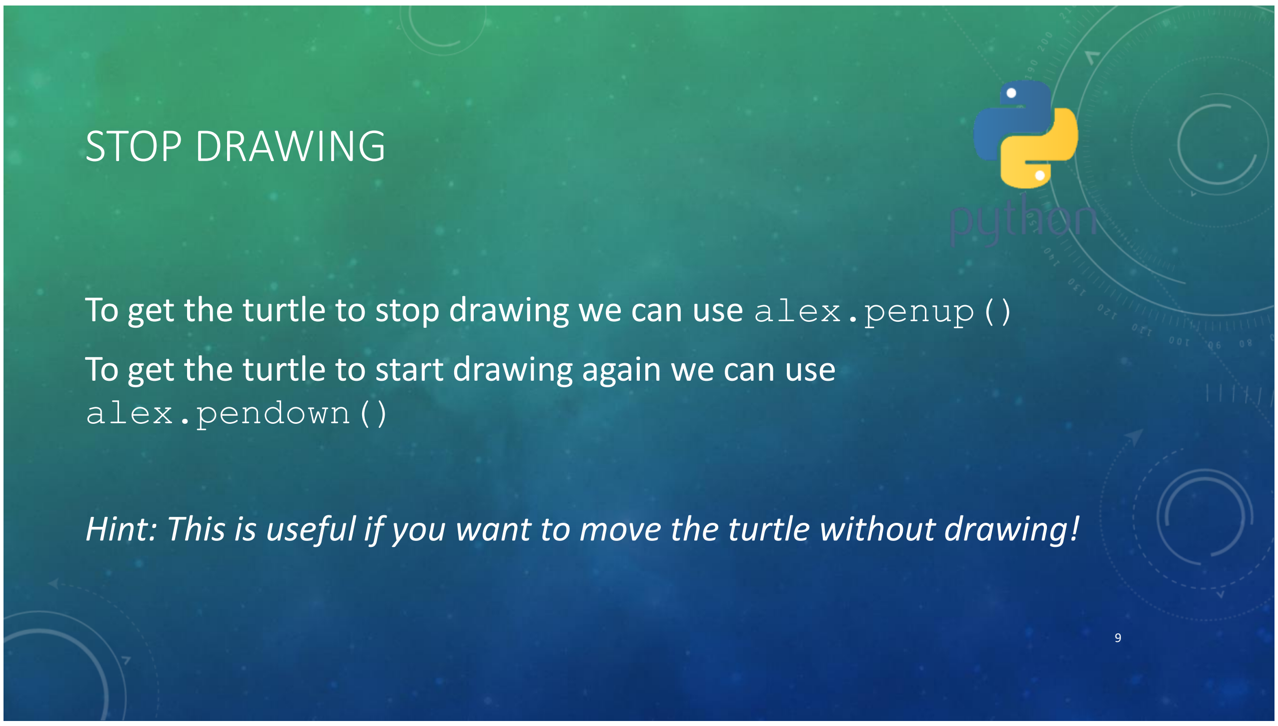
All of these methods do the same thing, move our turtle to the position (x, y).

We also have methods to set the x coordinate and y coordinate:

`setx()` & `sety()`

The method `home()` will return the turtle to the coordinate (0,0)

8



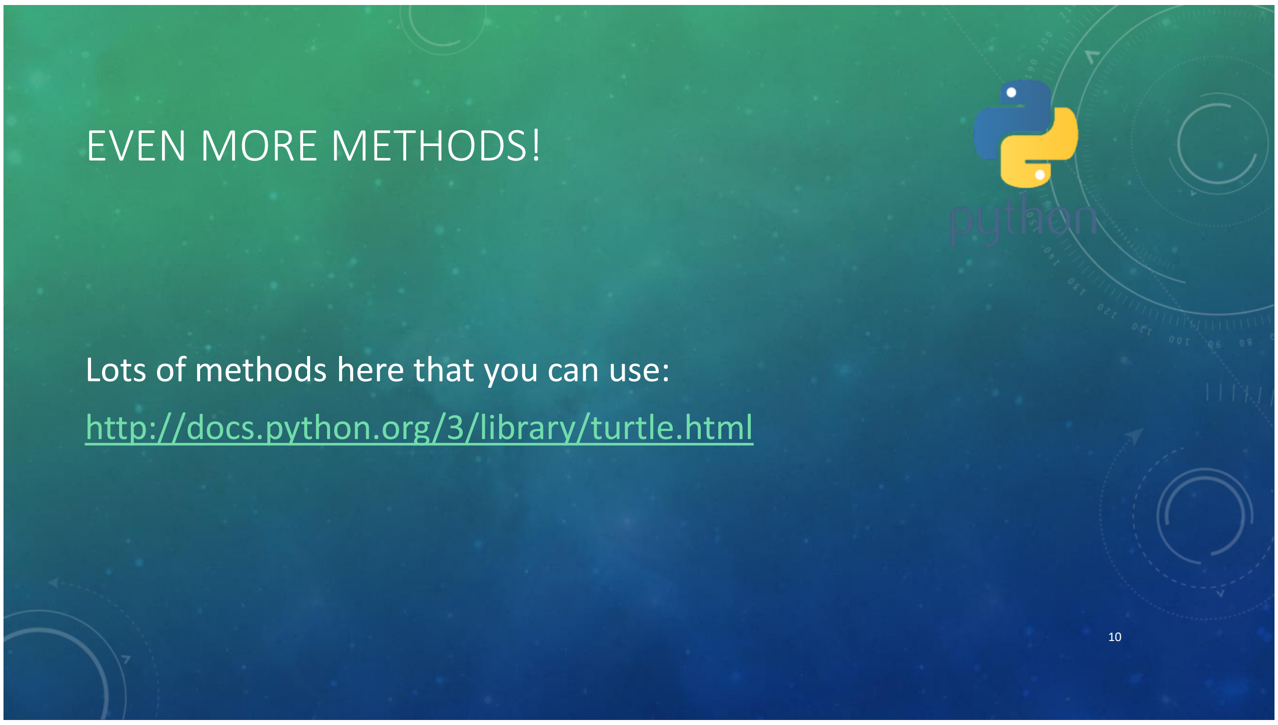
STOP DRAWING

To get the turtle to stop drawing we can use `alex.penup()`

To get the turtle to start drawing again we can use `alex.pendown()`

Hint: This is useful if you want to move the turtle without drawing!

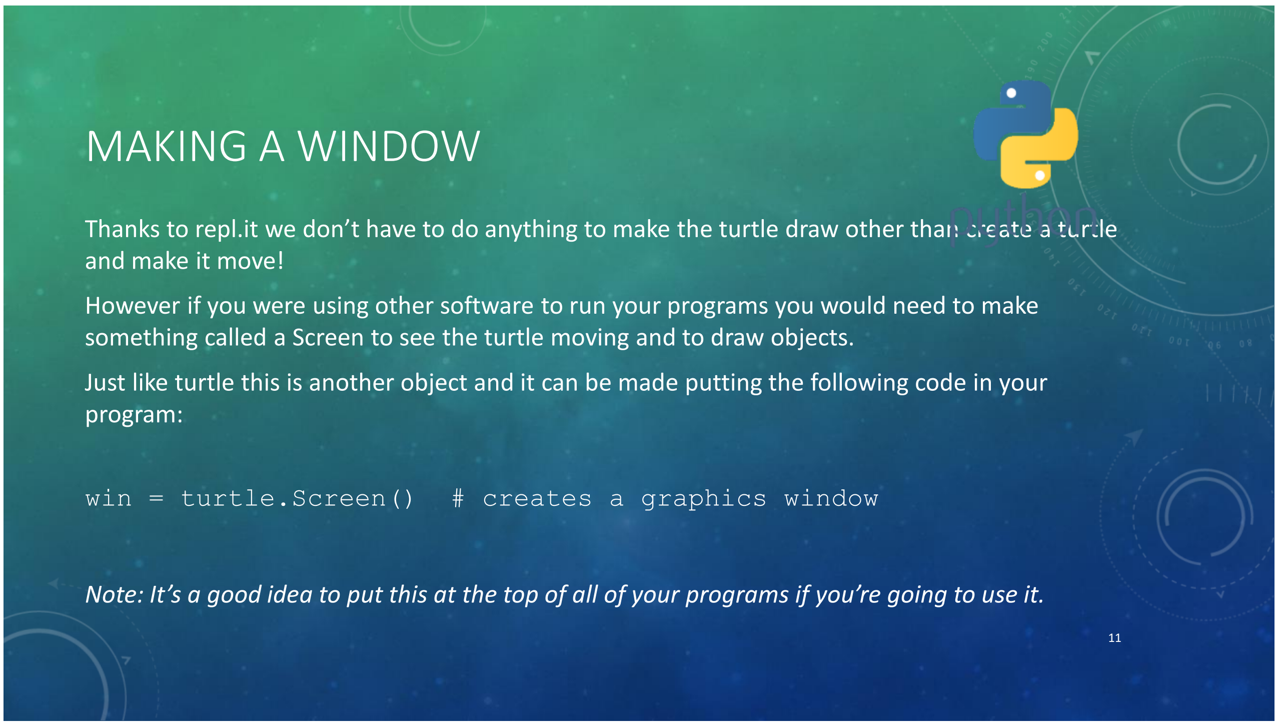
9



EVEN MORE METHODS!

Lots of methods here that you can use:
http://docs.python.org/3/library/turtle.html

10



MAKING A WINDOW

Thanks to repl.it we don't have to do anything to make the turtle draw other than create a turtle and make it move!

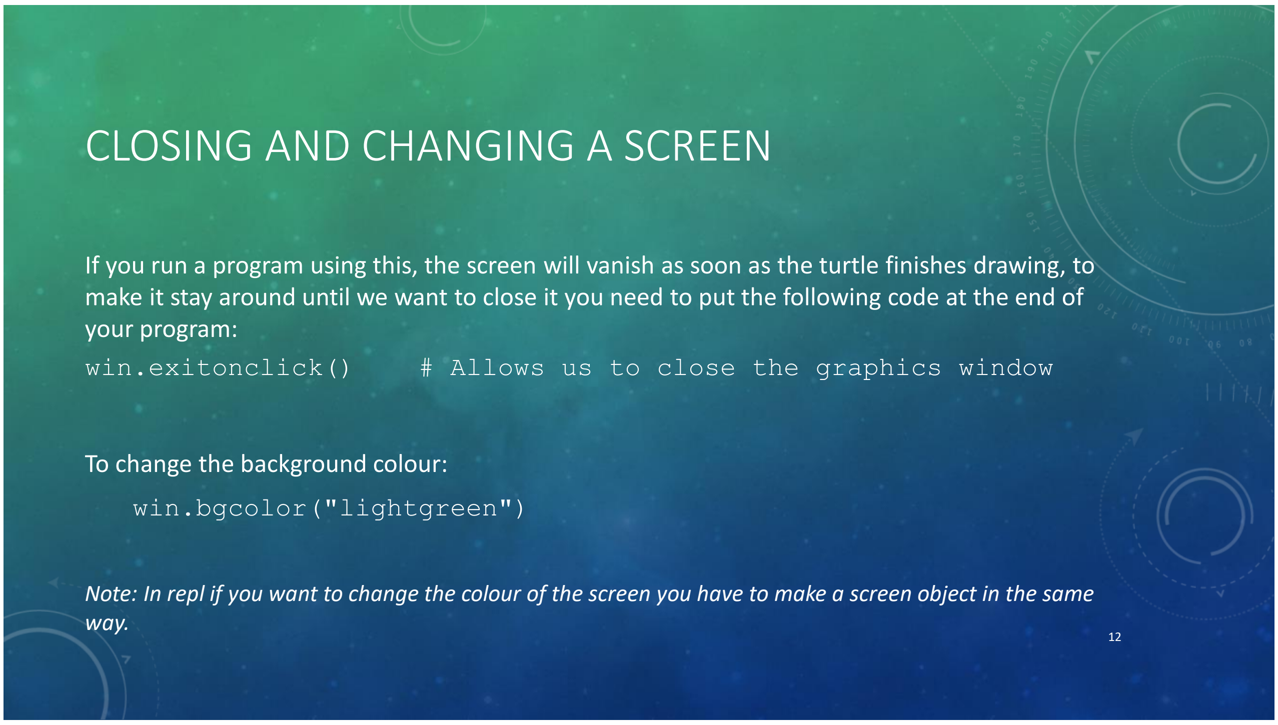
However if you were using other software to run your programs you would need to make something called a Screen to see the turtle moving and to draw objects.

Just like turtle this is another object and it can be made putting the following code in your program:

```
win = turtle.Screen()   # creates a graphics window
```

Note: It's a good idea to put this at the top of all of your programs if you're going to use it.

11



CLOSING AND CHANGING A SCREEN

If you run a program using this, the screen will vanish as soon as the turtle finishes drawing, to make it stay around until we want to close it you need to put the following code at the end of your program:

```
win.exitonclick()     # Allows us to close the graphics window
```

To change the background colour:

```
    win.bgcolor("lightgreen")
```

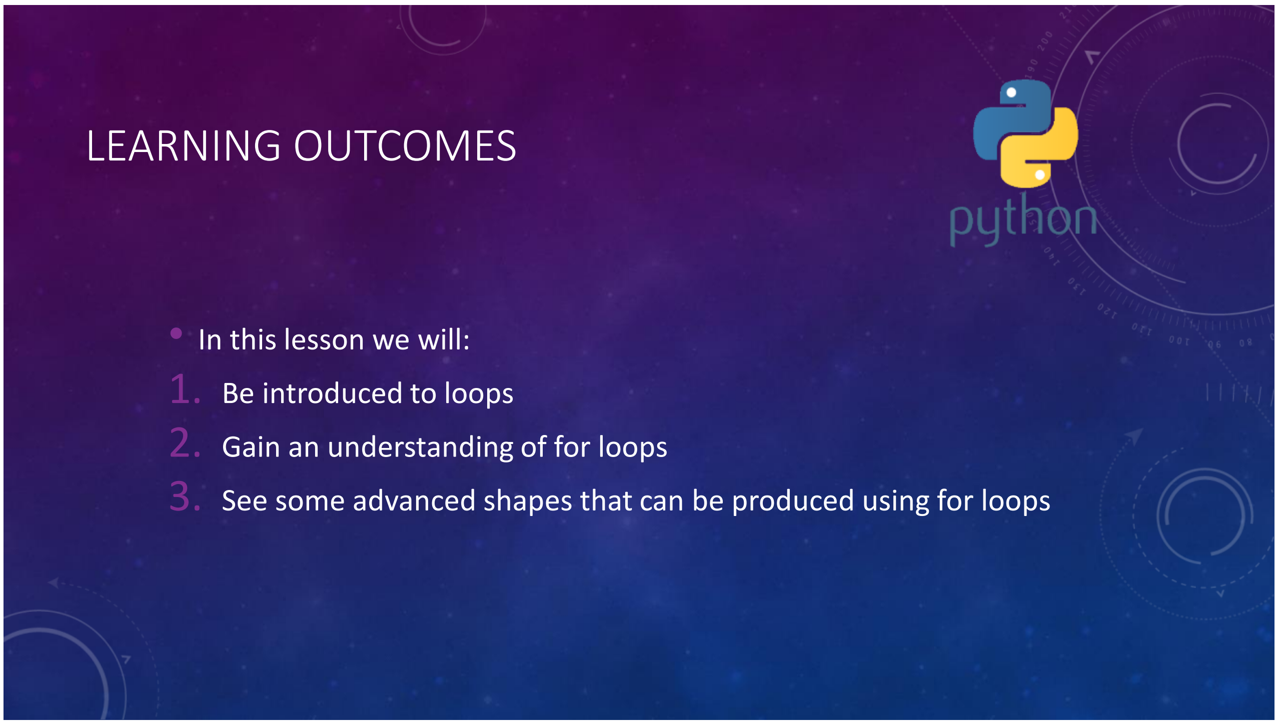
Note: In repl if you want to change the colour of the screen you have to make a screen object in the same way.

12

LOOPS

Maynooth University
National University of Ireland Maynooth

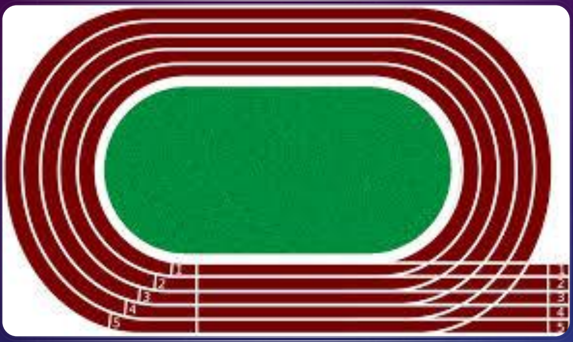
Developed by James Lockwood & Dr Aidan Mooney



LEARNING OUTCOMES

- In this lesson we will:

1. Be introduced to loops
2. Gain an understanding of for loops
3. See some advanced shapes that can be produced using for loops



LOOPS – WHAT ARE THEY?

Loops are used when you want to do the same thing multiple times.

Think of a running track that's 400m long, if you want to do 2km, you have to run around 5 times!

In programming there a few different types of loops, we're going to look at one called a *for loop*

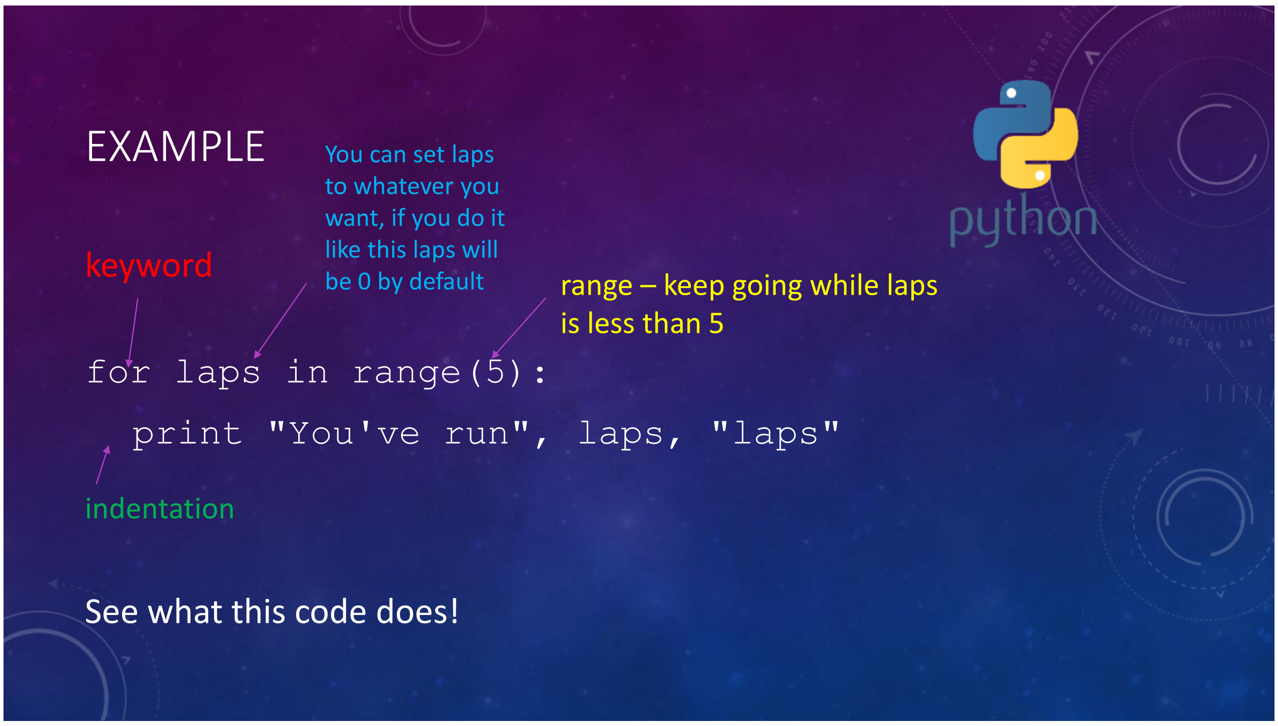
FOR LOOPS – WHAT DO YOU NEED

A keyword – in our case the key word is *for*

A range – e.g. keep running around the track until you've run 5 laps

An update – e.g. when you pass the start/finish line add 1 lap to your distance run, in Python this is done automatically

Indentation (only in Python) – one of the key things is everything *inside* the loop has to be indented 4 spaces (or one tab)



EXAMPLE

You can set laps to whatever you want, if you do it like this laps will be 0 by default

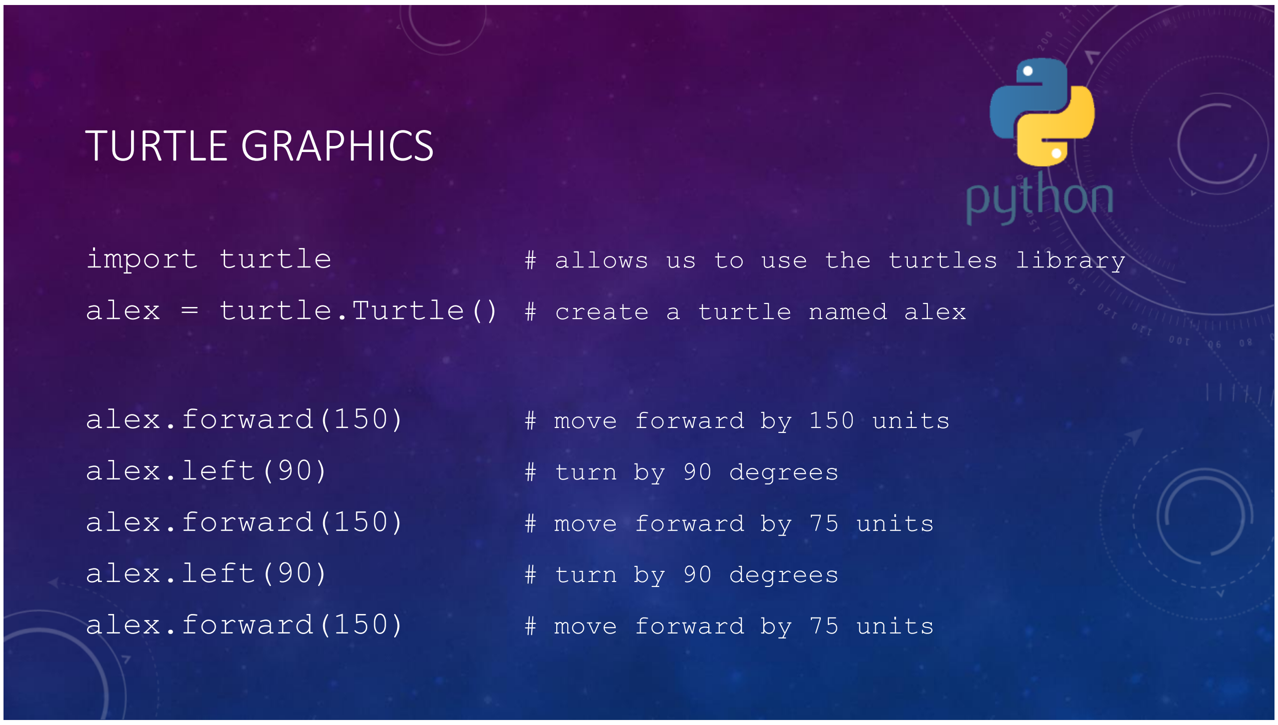
keyword

range – keep going while laps is less than 5

```
for laps in range(5):
  print "You've run", laps, "laps"
```

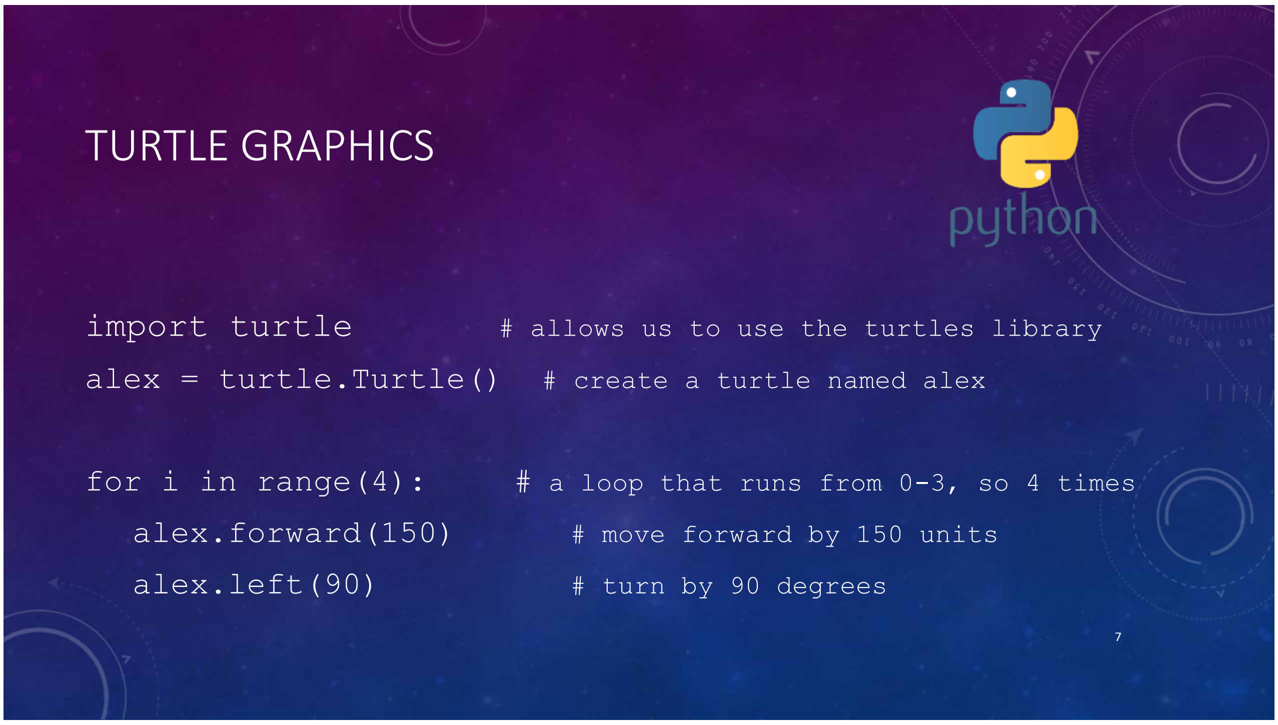
indentation

See what this code does!



TURTLE GRAPHICS

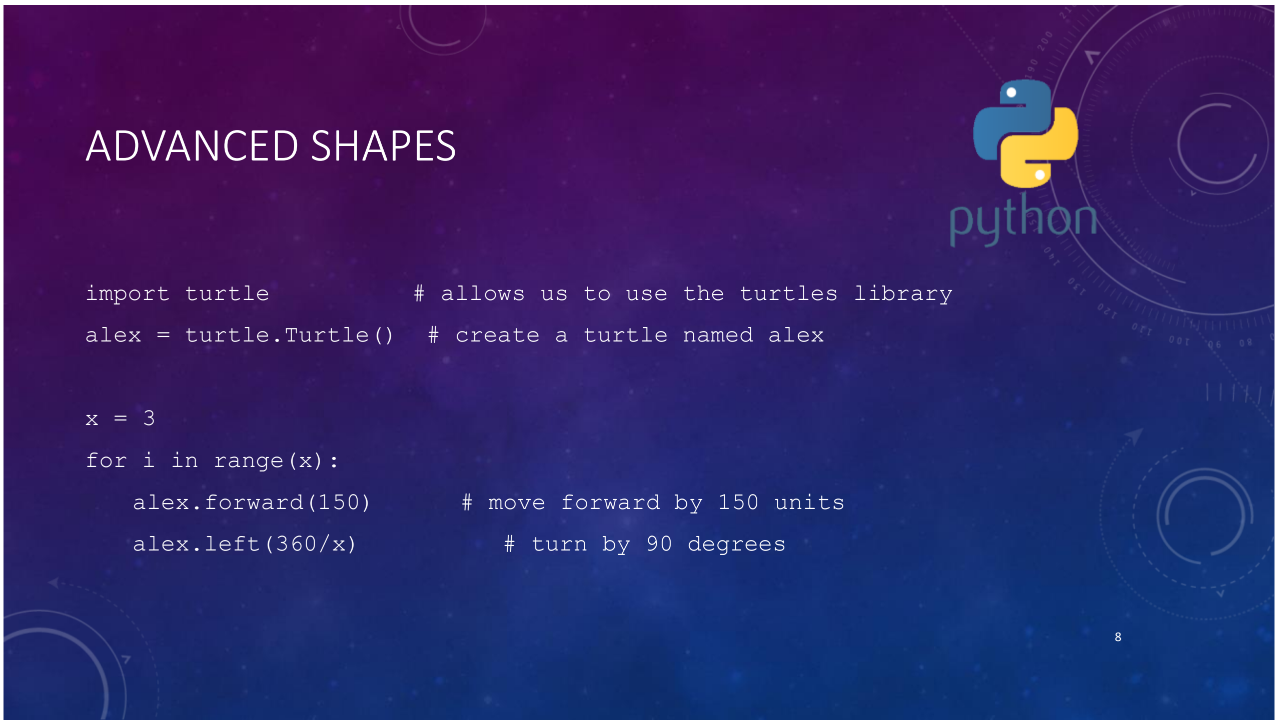
```
import turtle           # allows us to use the turtles library
alex = turtle.Turtle()  # create a turtle named alex

alex.forward(150)       # move forward by 150 units
alex.left(90)           # turn by 90 degrees
alex.forward(150)       # move forward by 75 units
alex.left(90)           # turn by 90 degrees
alex.forward(150)       # move forward by 75 units
```



TURTLE GRAPHICS

```
import turtle          # allows us to use the turtles library
alex = turtle.Turtle()   # create a turtle named alex

for i in range(4):       # a loop that runs from 0-3, so 4 times
    alex.forward(150)         # move forward by 150 units
    alex.left(90)             # turn by 90 degrees
```

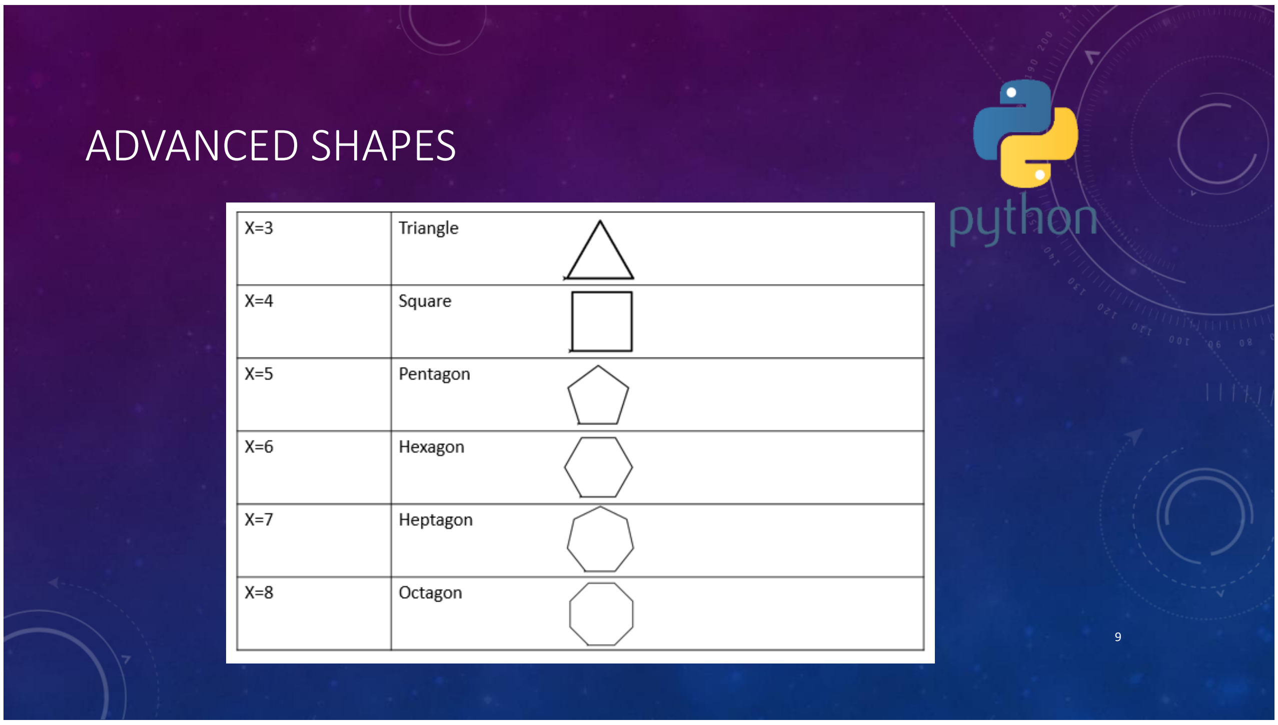
7



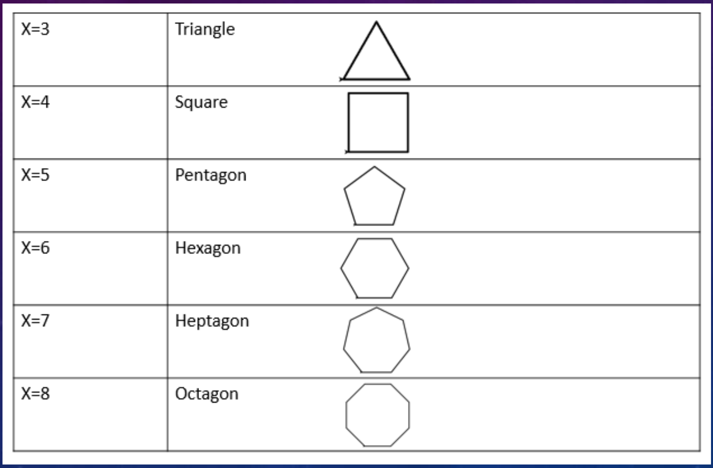
ADVANCED SHAPES

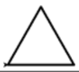
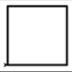
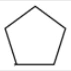
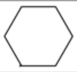
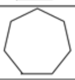
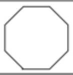
```
import turtle            # allows us to use the turtles library
alex = turtle.Turtle()   # create a turtle named alex

x = 3
for i in range(x):
    alex.forward(150)       # move forward by 150 units
    alex.left(360/x)           # turn by 90 degrees
```

8

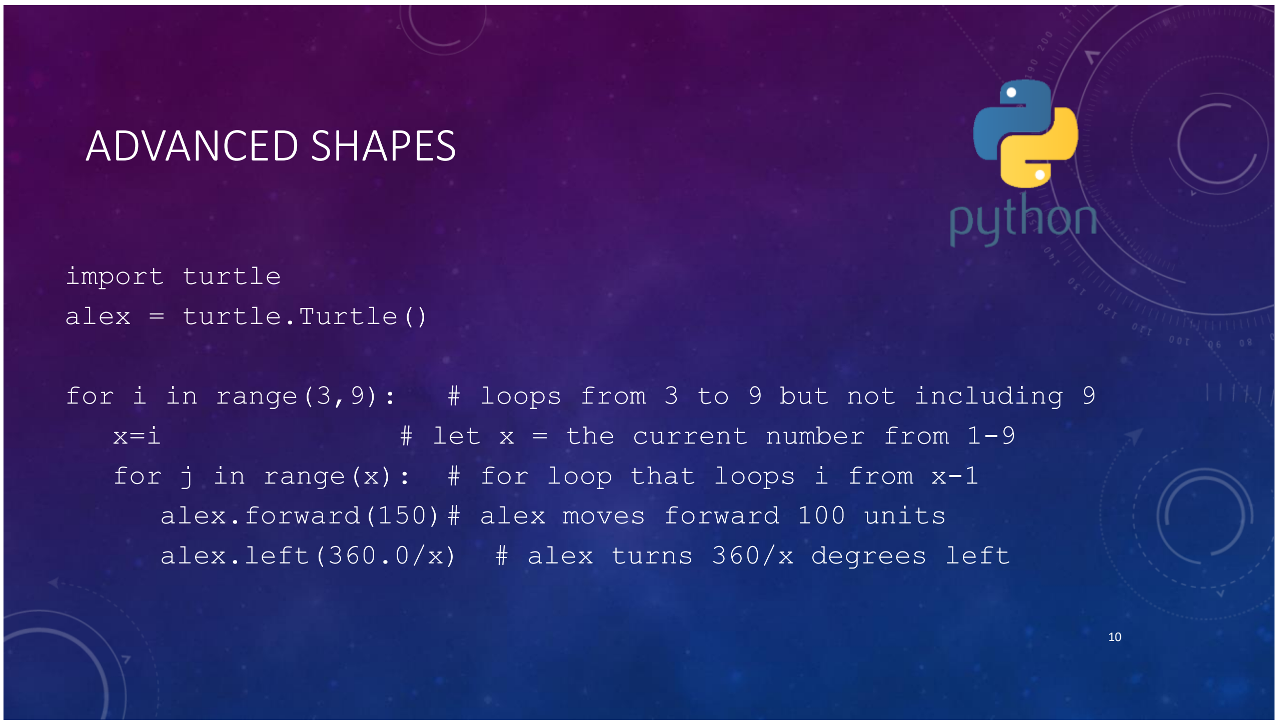


ADVANCED SHAPES



X=3	Triangle	
X=4	Square	
X=5	Pentagon	
X=6	Hexagon	
X=7	Heptagon	
X=8	Octagon	

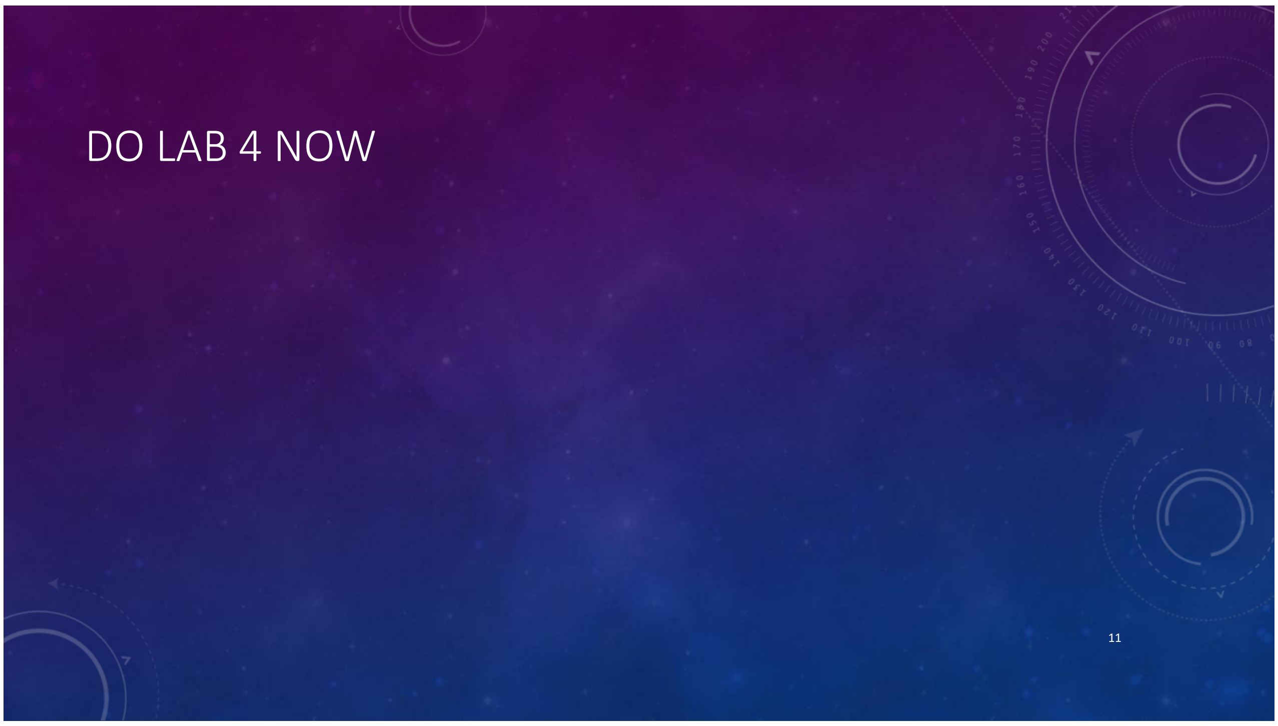
9



ADVANCED SHAPES

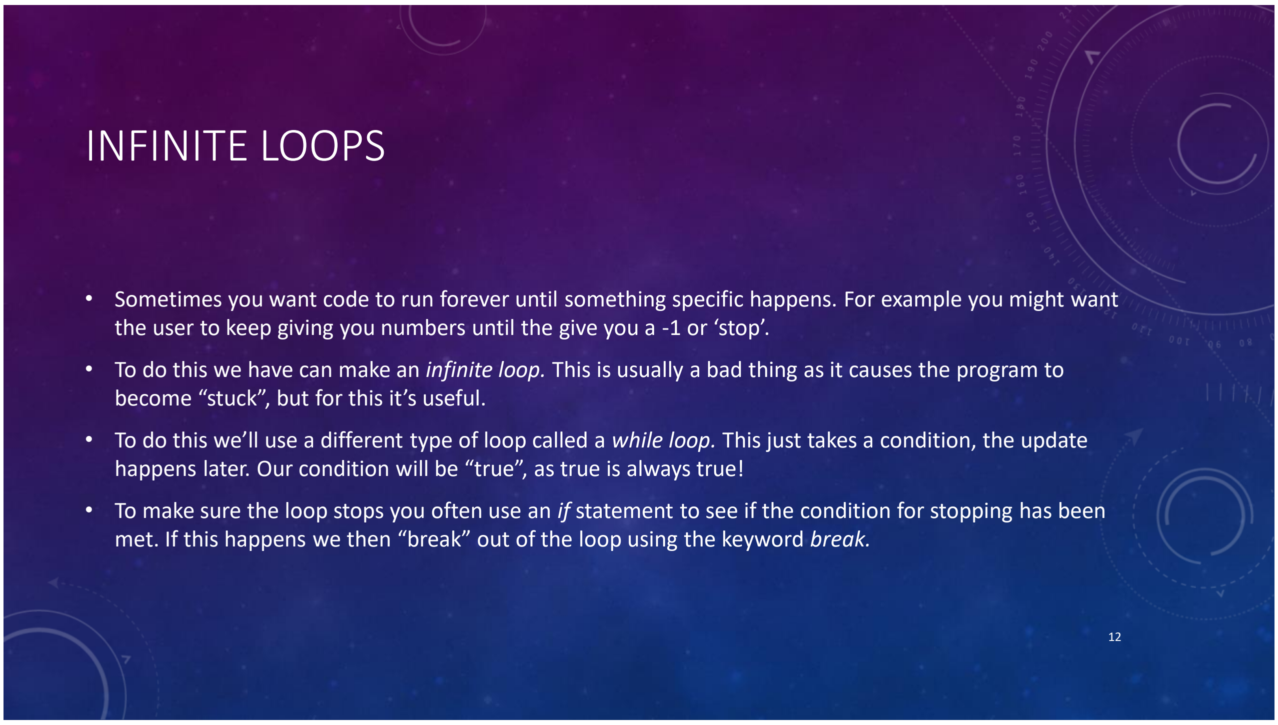
```
import turtle
alex = turtle.Turtle()

for i in range(3,9):    # loops from 3 to 9 but not including 9
   x=i                 # let x = the current number from 1-9
   for j in range(x):  # for loop that loops i from x-1
      alex.forward(150)# alex moves forward 100 units
      alex.left(360.0/x)  # alex turns 360/x degrees left
```

10



DO LAB 4 NOW

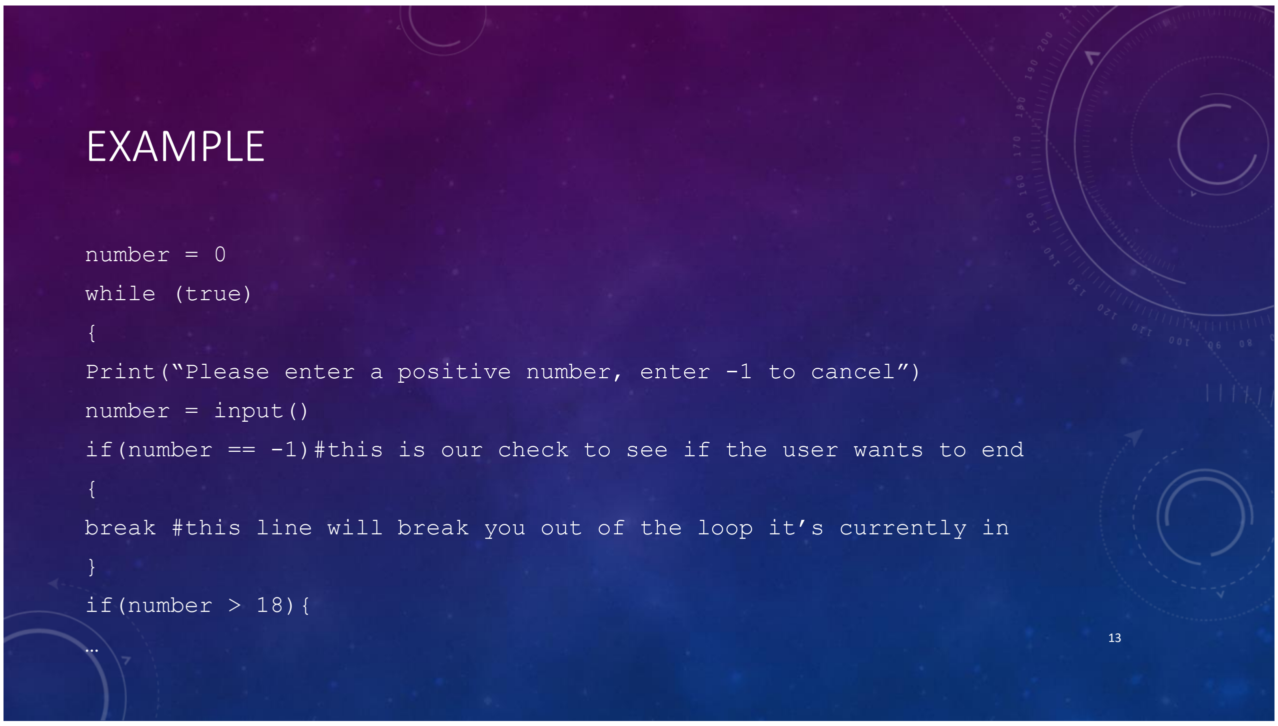
11



INFINITE LOOPS

- Sometimes you want code to run forever until something specific happens. For example you might want the user to keep giving you numbers until the give you a -1 or 'stop'.
- To do this we have can make an *infinite loop.* This is usually a bad thing as it causes the program to become "stuck", but for this it's useful.
- To do this we'll use a different type of loop called a *while loop.* This just takes a condition, the update happens later. Our condition will be "true", as true is always true!
- To make sure the loop stops you often use an *if* statement to see if the condition for stopping has been met. If this happens we then "break" out of the loop using the keyword *break.*

12



EXAMPLE

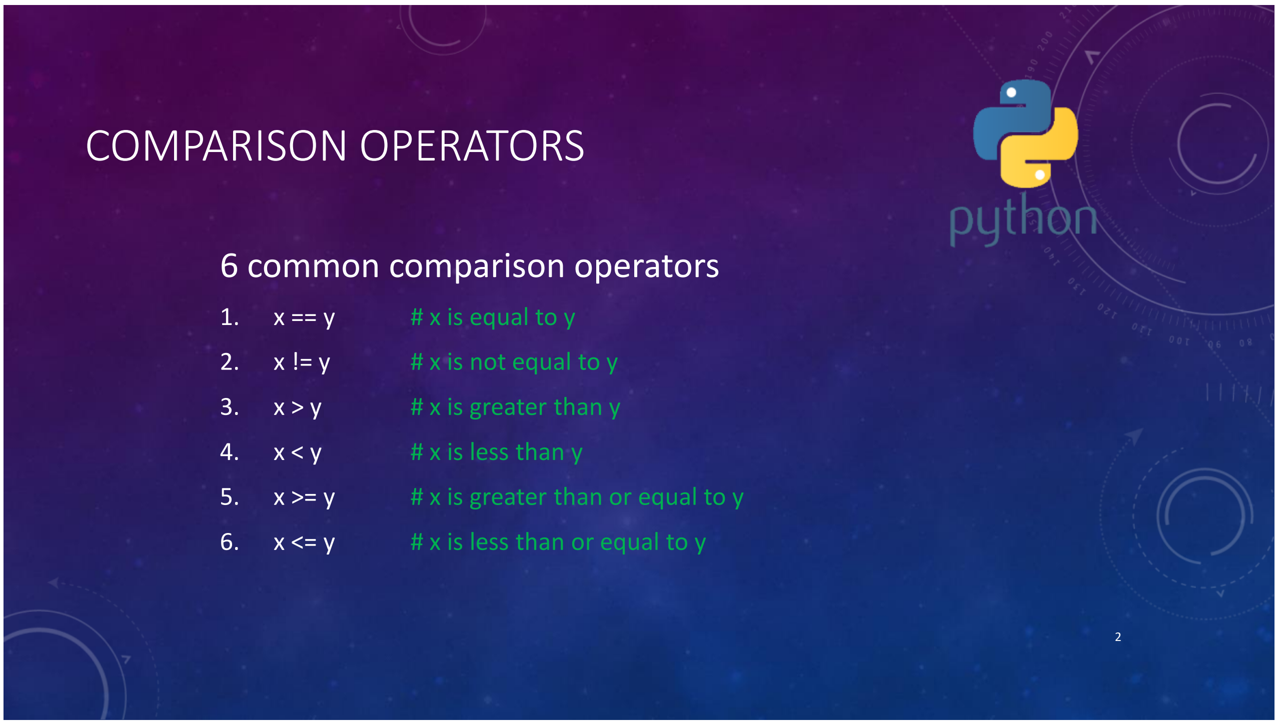
```
number = 0
while (true)
{
Print("Please enter a positive number, enter -1 to cancel")
number = input()
if(number == -1)#this is our check to see if the user wants to end
{
break #this line will break you out of the loop it's currently in
}
if(number > 18){
…
```

13

SELECTION STATEMENTS

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

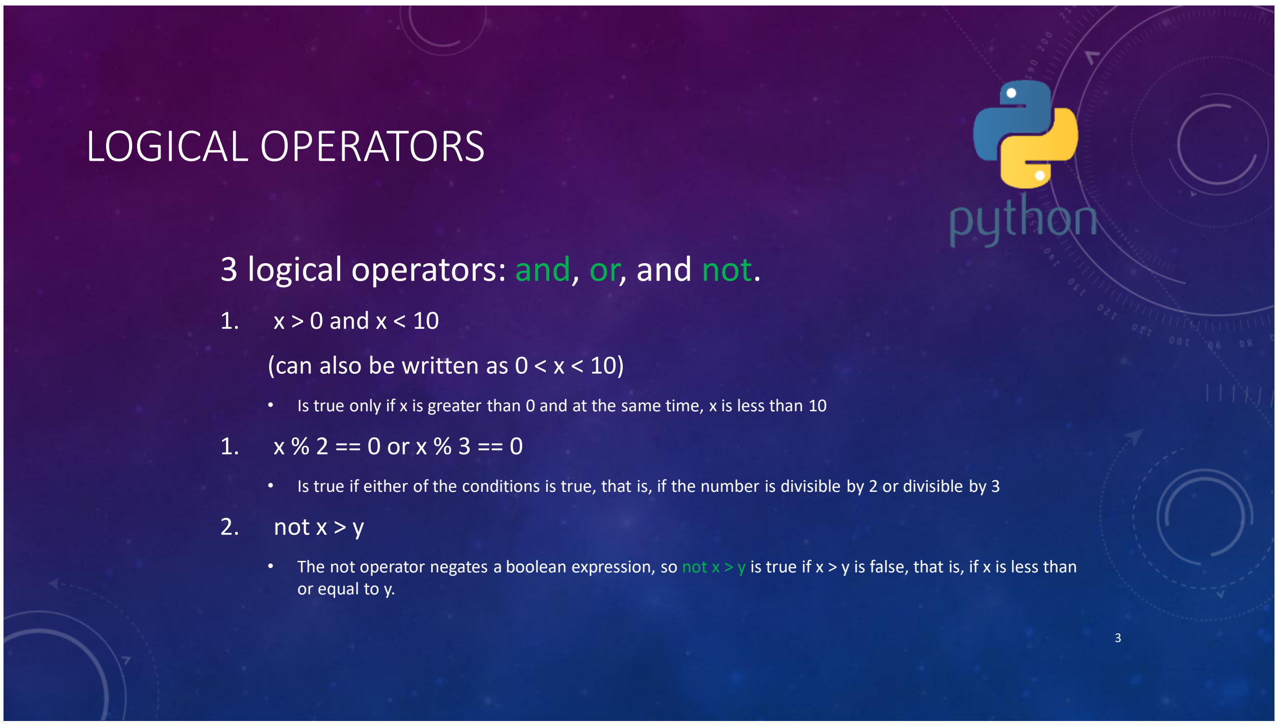


COMPARISON OPERATORS

6 common comparison operators

1.	x == y	# x is equal to y
2.	x != y	# x is not equal to y
3.	x > y	# x is greater than y
4.	x < y	# x is less than y
5.	x >= y	# x is greater than or equal to y
6.	x <= y	# x is less than or equal to y

2



LOGICAL OPERATORS

3 logical operators: and, or, and not.

1. x > 0 and x < 10

   (can also be written as 0 < x < 10)

   - Is true only if x is greater than 0 and at the same time, x is less than 10

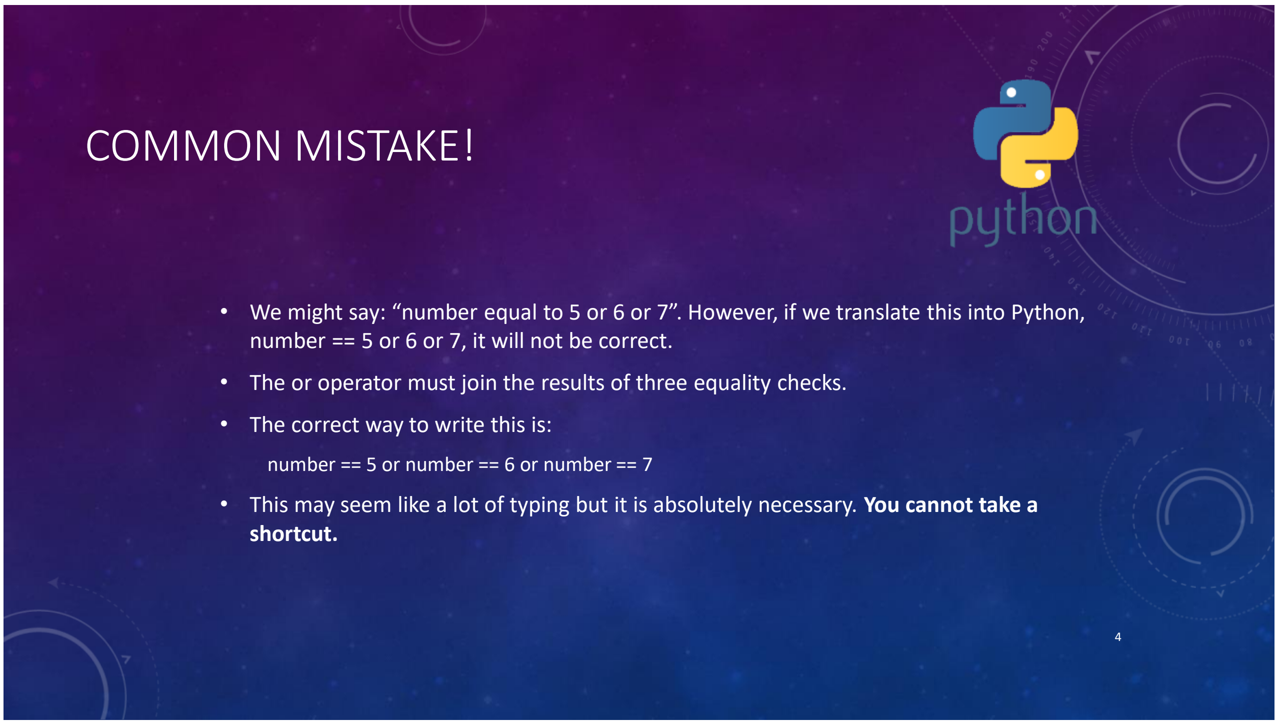
1. x % 2 == 0 or x % 3 == 0

   - Is true if either of the conditions is true, that is, if the number is divisible by 2 or divisible by 3

2. not x > y

   - The not operator negates a boolean expression, so not x > y is true if x > y is false, that is, if x is less than or equal to y.

3

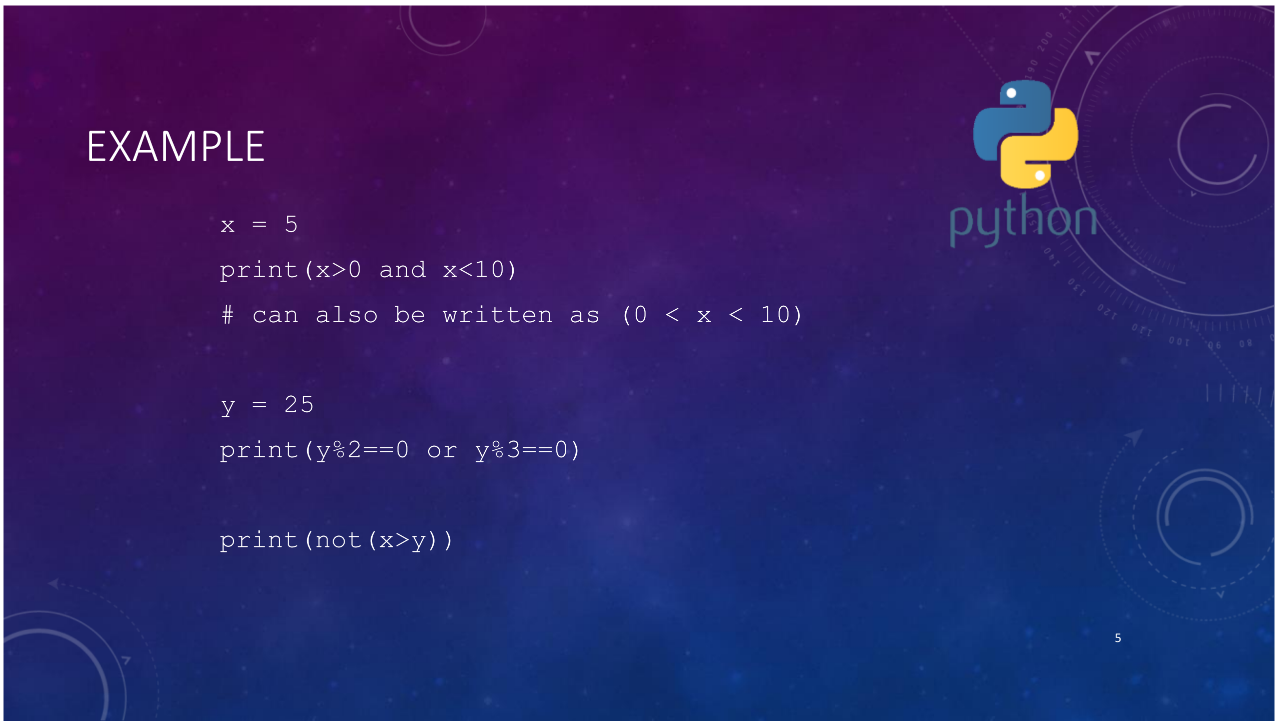


COMMON MISTAKE!

- We might say: “number equal to 5 or 6 or 7”. However, if we translate this into Python, number == 5 or 6 or 7, it will not be correct.
- The or operator must join the results of three equality checks.
- The correct way to write this is:

  number == 5 or number == 6 or number == 7
- This may seem like a lot of typing but it is absolutely necessary. **You cannot take a shortcut.**

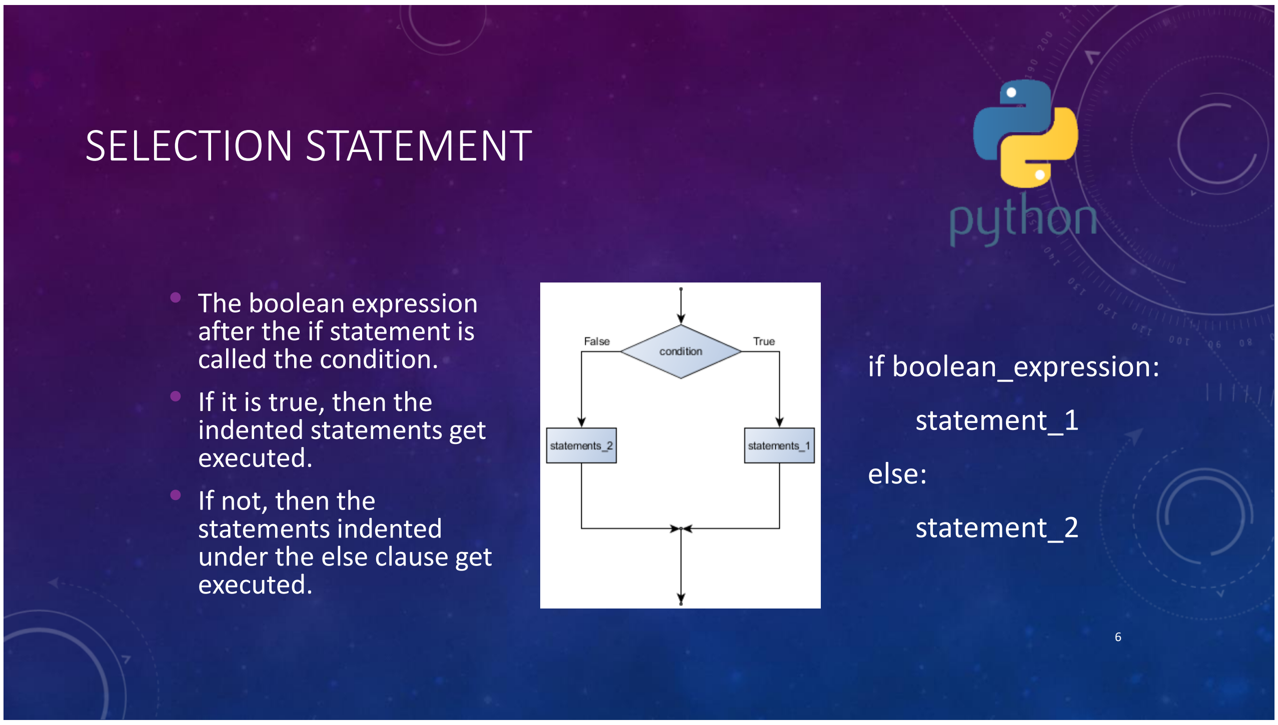
4



EXAMPLE

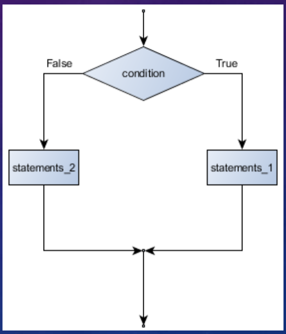
```
x = 5
print(x>0 and x<10)
# can also be written as (0 < x < 10)

y = 25
print(y%2==0 or y%3==0)

print(not(x>y))
```

5



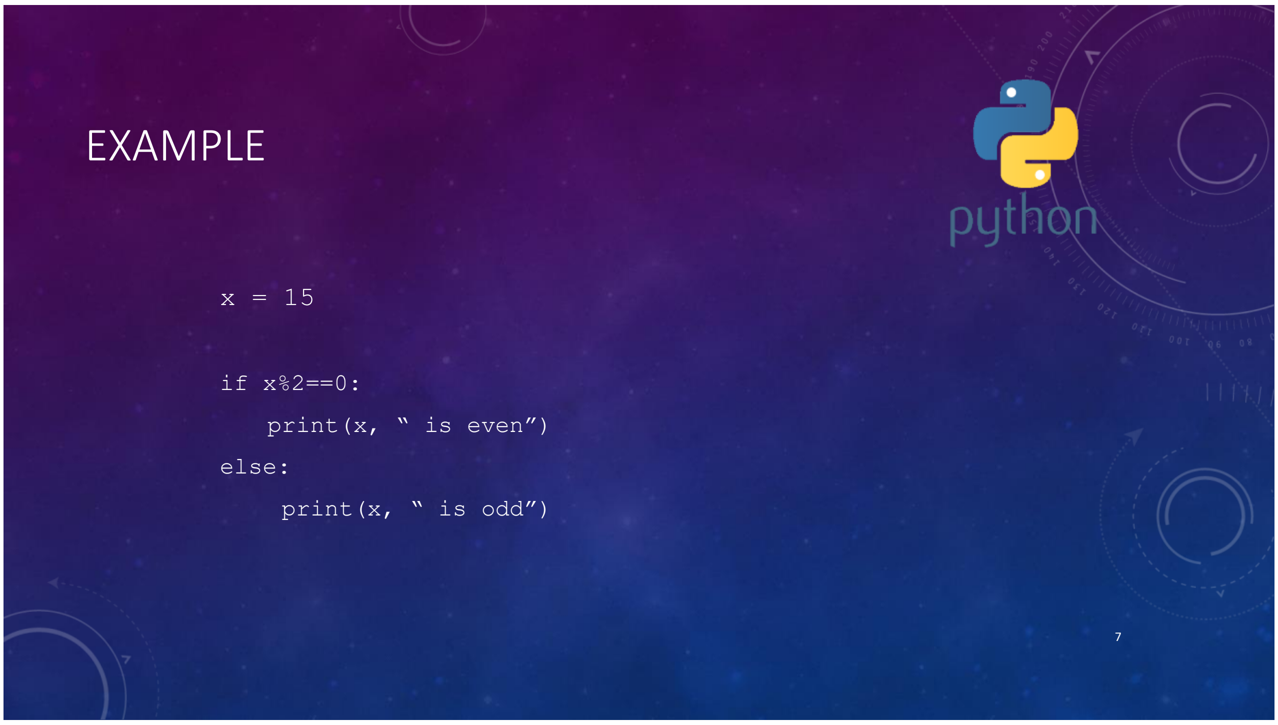
SELECTION STATEMENT

- The boolean expression after the if statement is called the condition.
- If it is true, then the indented statements get executed.
- If not, then the statements indented under the else clause get executed.



```
if boolean_expression:
    statement_1
else:
    statement_2
```

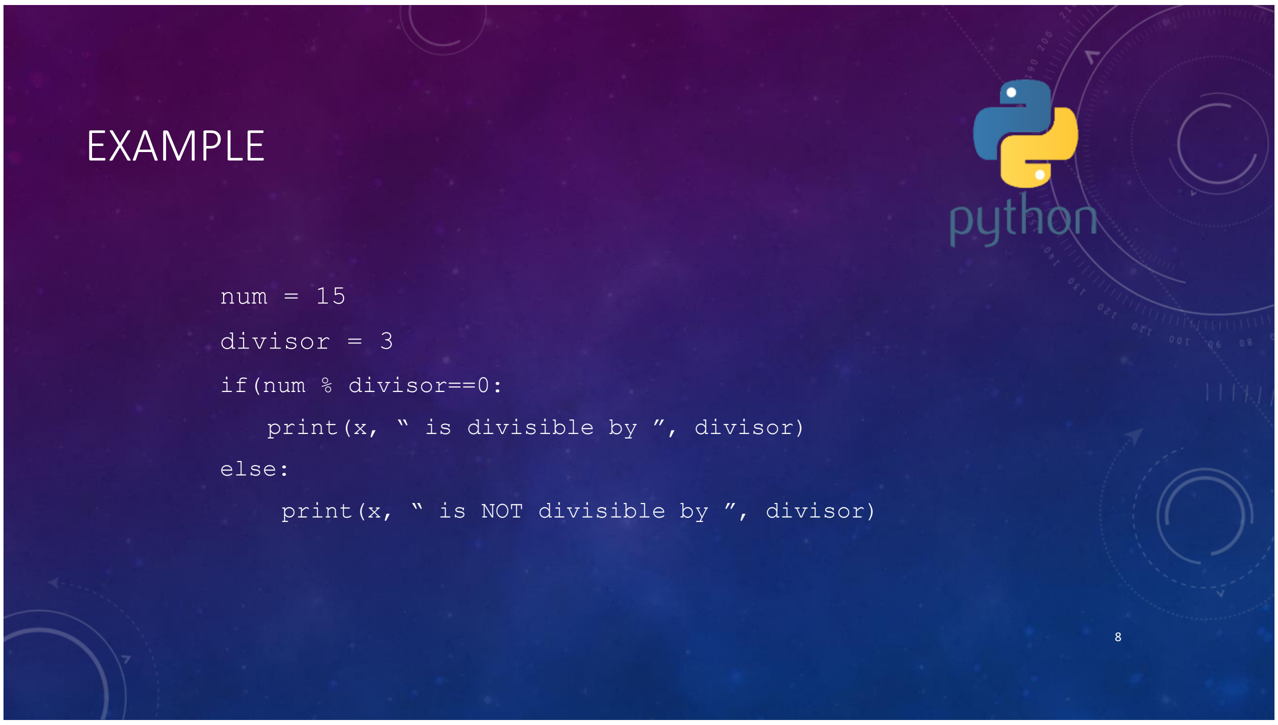
6



EXAMPLE

```
x = 15

if x%2==0:
   print(x, " is even")
else:
    print(x, " is odd")
```

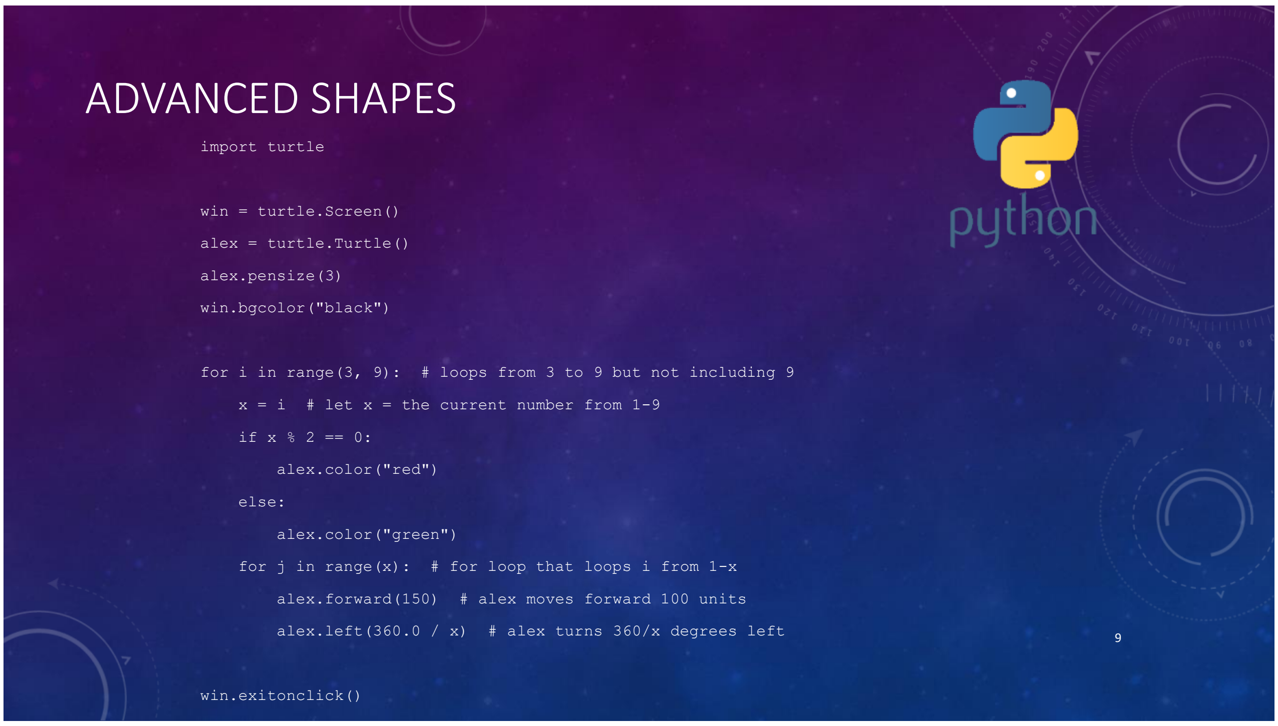
7



EXAMPLE

```
num = 15
divisor = 3
if(num % divisor==0:
   print(x, " is divisible by ", divisor)
else:
    print(x, " is NOT divisible by ", divisor)
```

8



ADVANCED SHAPES

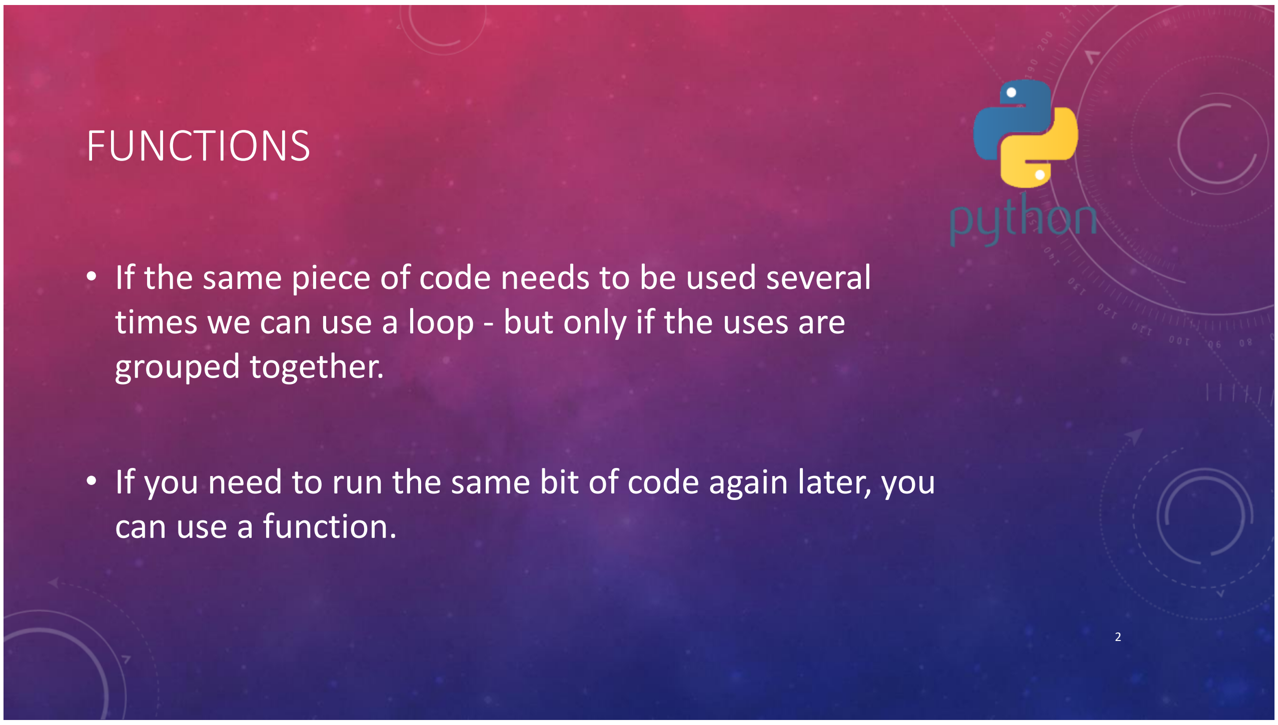
```
import turtle

win = turtle.Screen()
alex = turtle.Turtle()
alex.pensize(3)
win.bgcolor("black")

for i in range(3, 9):  # loops from 3 to 9 but not including 9
    x = i  # let x = the current number from 1-9
    if x % 2 == 0:
        alex.color("red")
    else:
        alex.color("green")
    for j in range(x):  # for loop that loops i from 1-x
        alex.forward(150)  # alex moves forward 100 units
        alex.left(360.0 / x)  # alex turns 360/x degrees left

win.exitonclick()
```

9

FUNCTIONS

Maynooth University
National University of Ireland Maynooth

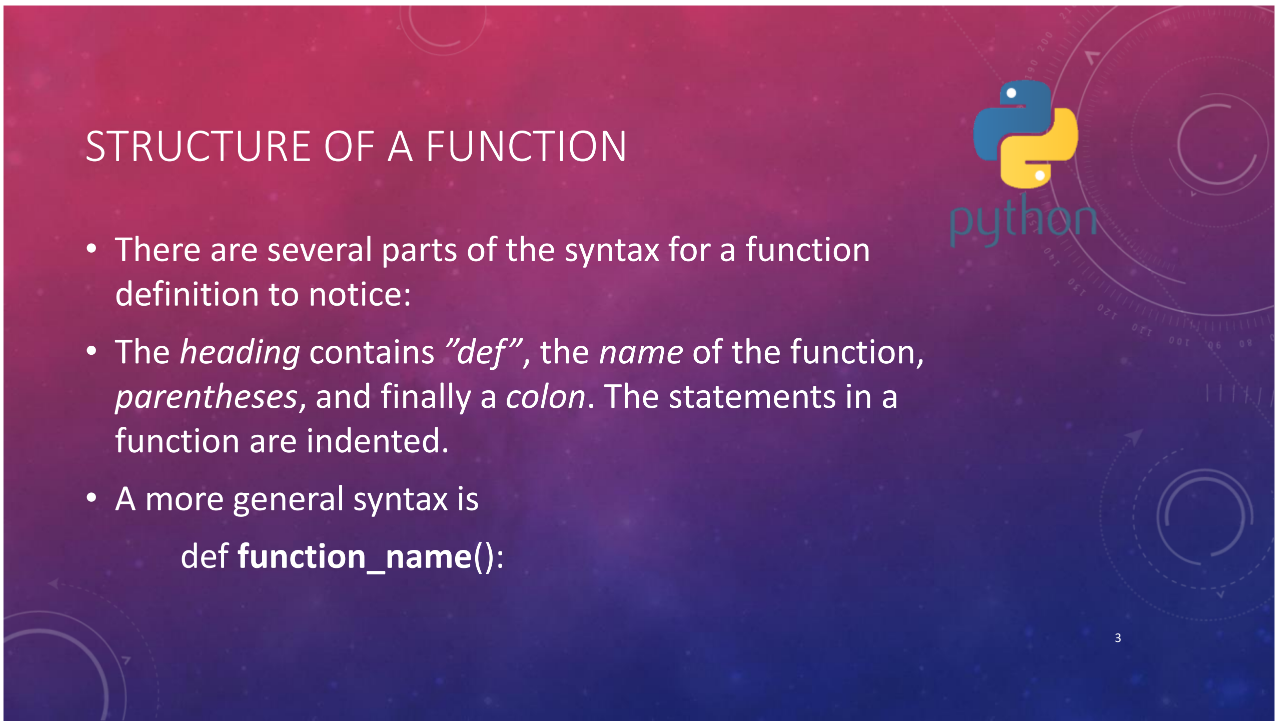
Developed by James Lockwood & Dr Aidan Mooney



FUNCTIONS

- If the same piece of code needs to be used several times we can use a loop - but only if the uses are grouped together.
- If you need to run the same bit of code again later, you can use a function.

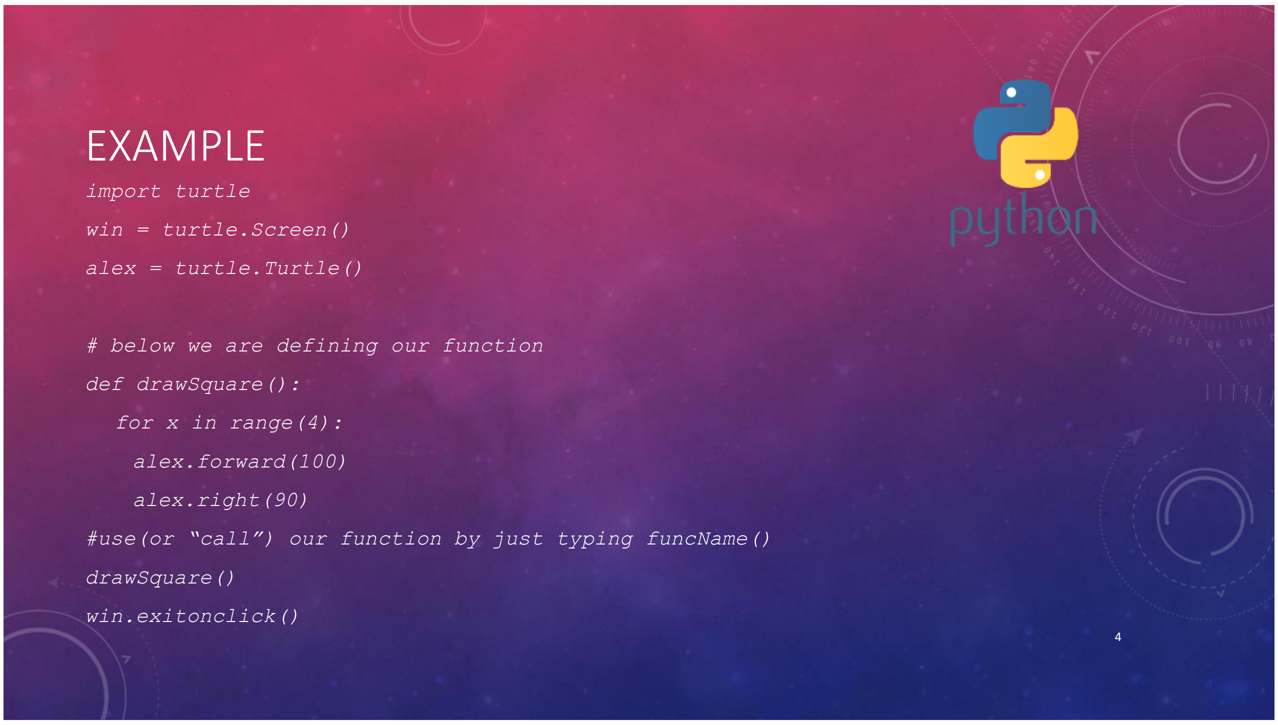
2



STRUCTURE OF A FUNCTION

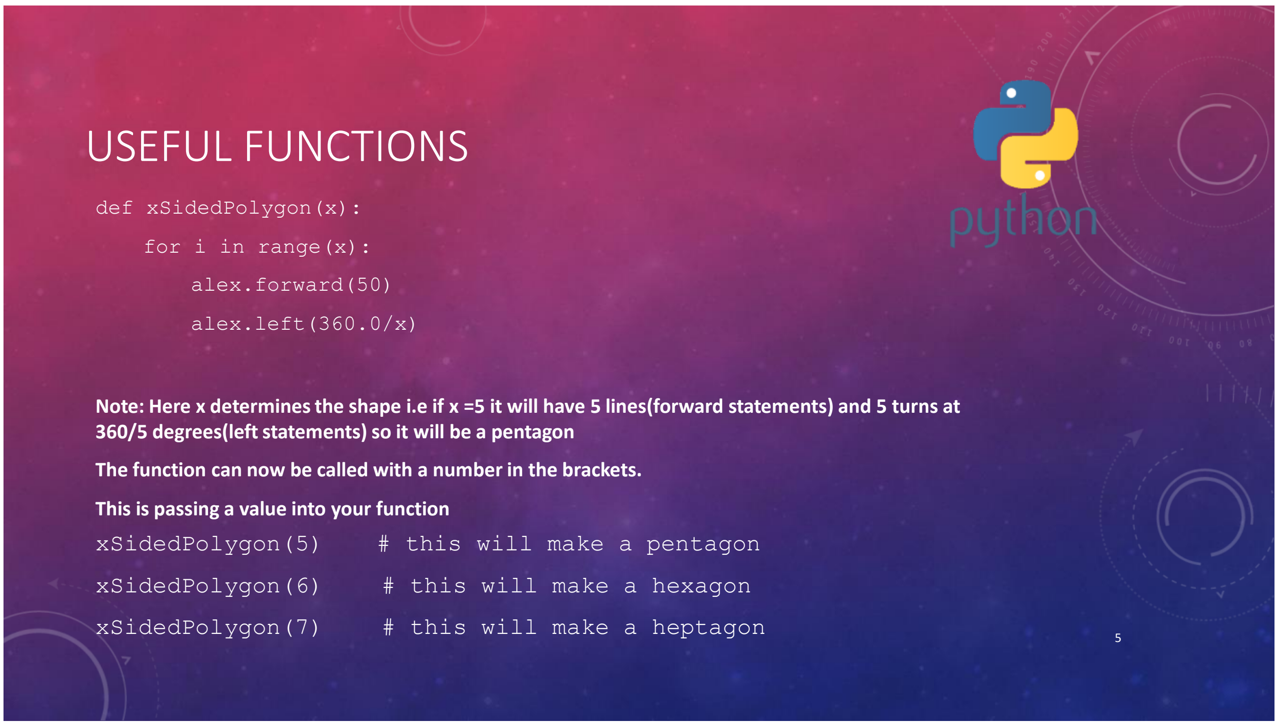
- There are several parts of the syntax for a function definition to notice:
- The *heading* contains *"def"*, the *name* of the function, *parentheses*, and finally a *colon*. The statements in a function are indented.
- A more general syntax is

  def **function_name**():



EXAMPLE

```
import turtle
win = turtle.Screen()
alex = turtle.Turtle()

# below we are defining our function
def drawSquare():
  for x in range(4):
    alex.forward(100)
    alex.right(90)
#use(or "call") our function by just typing funcName()
drawSquare()
win.exitonclick()
```



USEFUL FUNCTIONS

```
def xSidedPolygon(x):
    for i in range(x):
        alex.forward(50)
        alex.left(360.0/x)
```

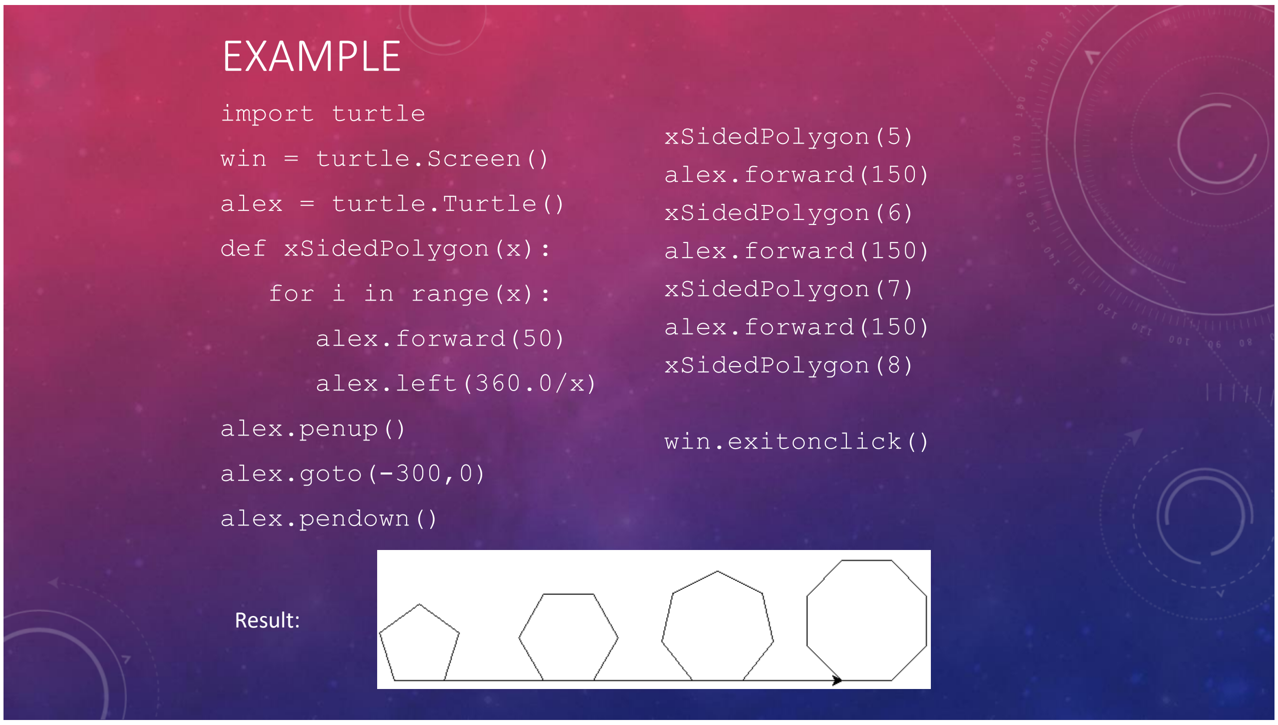
Note: Here x determines the shape i.e if x =5 it will have 5 lines(forward statements) and 5 turns at 360/5 degrees(left statements) so it will be a pentagon

The function can now be called with a number in the brackets.

This is passing a value into your function

```
xSidedPolygon(5)    # this will make a pentagon
xSidedPolygon(6)    # this will make a hexagon
xSidedPolygon(7)    # this will make a heptagon
```

5

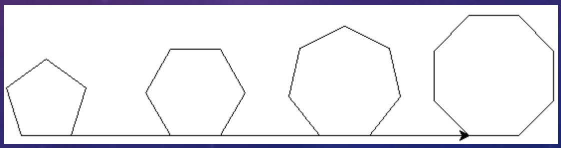


EXAMPLE

```
import turtle
win = turtle.Screen()
alex = turtle.Turtle()
def xSidedPolygon(x):
    for i in range(x):
        alex.forward(50)
        alex.left(360.0/x)
alex.penup()
alex.goto(-300,0)
alex.pendown()
```

```
xSidedPolygon(5)
alex.forward(150)
xSidedPolygon(6)
alex.forward(150)
xSidedPolygon(7)
alex.forward(150)
xSidedPolygon(8)

win.exitonclick()
```

Result:



F

WEB DEVELOPMENT LESSONS

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Making a basic website

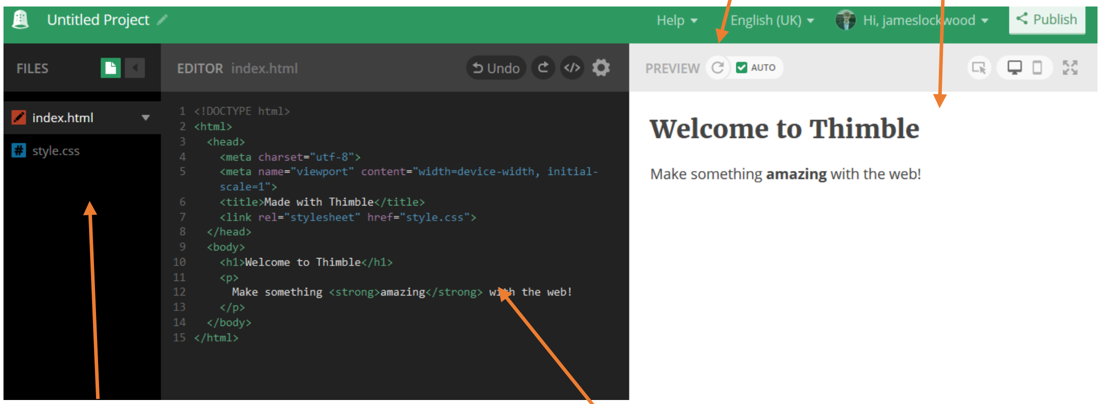
Our basic website will be made up of two things, HTML and CSS. HTML stands for Hypertext Markup Language. HTML is used to structure content on our website, to make sure it's in the place we want it to be and displayed how we want it. CSS stands for Cascading Style Sheets and CSS is used to style the content, for example changing the colour of text, giving different borders to areas etc.

We'll start with HTML and add some content. The way we use HTML is using HTML tags. The tags tell your browser how the information should be structured. HTML tags are keywords surrounded by angled brackets such as <html> and <p>.

When you open Thimble, you should have an untitled project open up like this below:

To rename your project go to the top left and give it a name like "My Website".

This is a preview of the web page; it will automatically update so long as this box is ticked.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
    scale=1">
    <title>Made with Thimble</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Welcome to Thimble</h1>
    <p>
      Make something <strong>amazing</strong> with the web!
    </p>
  </body>
</html>
```

Welcome to Thimble

Make something **amazing** with the web!

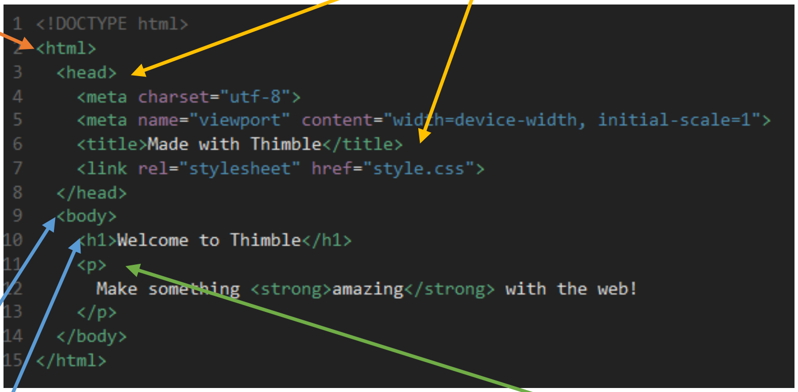
On the left of the screen is the list of files in our project. One is our HTML file, the other our styling file (CSS). We can add other files (including videos, images, JavaScript) to our project.

This is your HTML code; it's made up of a variety of tags which we'll go through below!

If you look at the HTML code on the left you'll see at the top there's a <!DOCTYPE html>, this to tell the browser that the following lines are HTML.

We then have a <html> tag and you'll see down the bottom a closing tag </html>. All of your html tags and the content should be in between these two tags.

We then have a <head> tag. Inside this and the closing </head> tag is information about the page. One line to note for now inside the header is the <title> tags. Change the title (the words in between) if you want!



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Made with Thimble</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Welcome to Thimble</h1>
    <p>
      Make something <strong>amazing</strong> with the web!
    </p>
  </body>
</html>
```

The <body> tag further down the page contains the content.

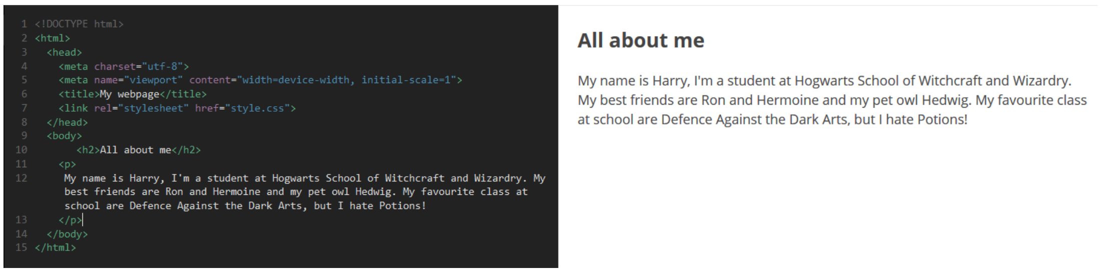
Inside the <body> tags you'll see a <h1> tag. In HTML, there's six types of headers with <h1> being the largest font and <h6> being the smallest. Change the text in here and see how it changes in the page on the right!

We then have a <p> tag which signifies a paragraph, usually made up of text. Again, change this and see how it changes!

Exercise:

Add in another header and a paragraph, title the header "All about X", replace X with whatever you want your webpage to be about. In the paragraph, write a few sentences about a hobby, your favourite book, food etc. If you're writing anything about yourself don't put anything too specific you should never put your address or email or mobile number on this or any website). You can delete the first header and paragraph if you want.

Hopefully your code and webpage look similar to the picture below. Almost every website you’ve ever been on or that you’ll ever make uses the tags we’ve looked at so far and you’ve already got a working webpage! However, it doesn’t look very good so let’s make it look at bit more colourful!



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>My webpage</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
        <h2>All about me</h2>
    <p>
     My name is Harry, I'm a student at Hogwarts School of Witchcraft and Wizardry. My
     best friends are Ron and Hermoine and my pet owl Hedwig. My favourite class at
     school are Defence Against the Dark Arts, but I hate Potions!
    </p>
  </body>
</html>
```

All about me

My name is Harry, I'm a student at Hogwarts School of Witchcraft and Wizardry. My best friends are Ron and Hermoine and my pet owl Hedwig. My favourite class at school are Defence Against the Dark Arts, but I hate Potions!

CSS is the way we style our webpage, it determines where things appear and how they look. There’s a couple of ways you can do styling and we’ll look at two.

Inline Styling

The first is what’s called inline styling. This is good if you want a specific paragraph/heading/image/table etc. to look a certain way.

To use inline styling, you have to put the keyword “style” into the tag so that it looks like this:

```
<h2 style="">Your heading</h2>
```

There are many elements you can style, we’ll just change the colour and the font of the heading to start. To do this you need to place the following code inside the quotes so it looks like this:

```
<h2 style="color:blue; font-family:courier;">Your heading</h2>
```

Note the US spelling of color

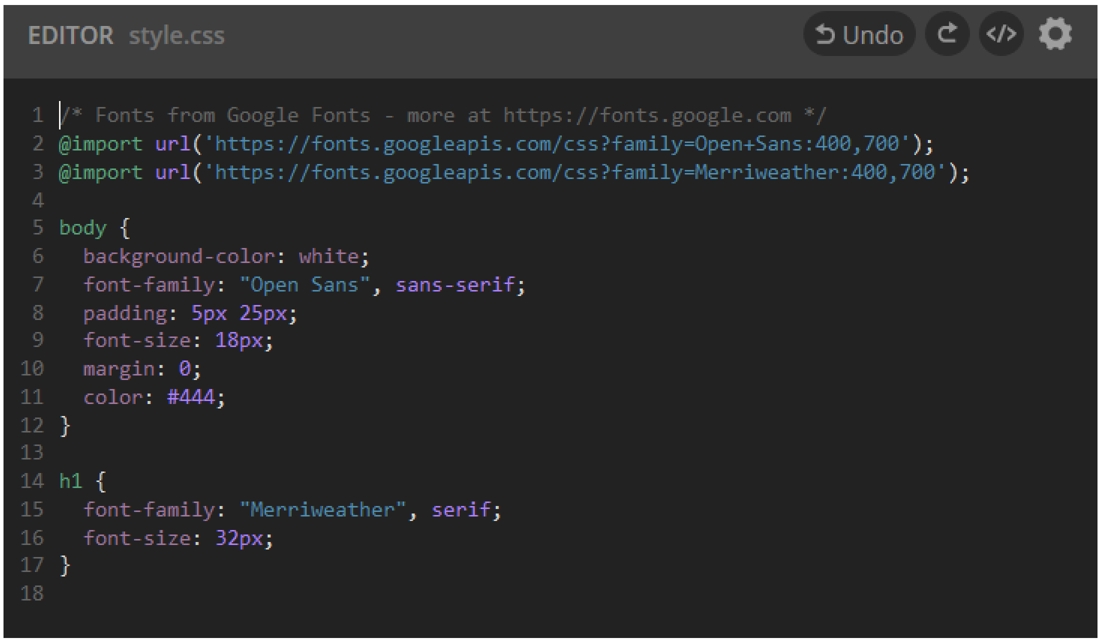
We use a semicolon ‘;’ to split the different styling elements

You can choose whatever colour you want and whatever font you want, for a list of commonly used fonts see this webpage https://www.w3schools.com/cssref/css_websafe_fonts.asp.

For a list of the different ways you can style text through CSS look at the list at the bottom of this webpage: https://www.w3schools.com/css/css_text.asp.

Using a stylesheet

You might have noticed on the far left of the screen is a files section. In this is our html file (called index.html) and another file called style.css. Click on this file and you should see something like this:



```css
/* Fonts from Google Fonts - more at https://fonts.google.com */
@import url('https://fonts.googleapis.com/css?family=Open+Sans:400,700');
@import url('https://fonts.googleapis.com/css?family=Merriweather:400,700');

body {
  background-color: white;
  font-family: "Open Sans", sans-serif;
  padding: 5px 25px;
  font-size: 18px;
  margin: 0;
  color: #444;
}

h1 {
  font-family: "Merriweather", serif;
  font-size: 32px;
}

```

This is a style sheet. At the top, you'll see two import lines, these are used to import various things, in this case font styles.

We then have two sections which have a tag name (body and h1) followed by a set of curly brackets. In between these lines are then lines of CSS to style the tag. This is the best way of styling your webpage as it ensures all sections have a common style and saves you having to type lines of code each time you make a new paragraph, heading etc.

We're going to style our paragraphs so that they look the same! To do this add in this code at the bottom of the style file. The p is the selector, and it tells the computer which tags should be styled in a specific way. Most selectors have many different attributes you can style. You can see some of these in the code that's already there. For our paragraphs (<p>) we'll just change the font size and the font type.

```
p{
   font-family: “Sans-serif”, Veranda;
   font-size: 24px;
}
```

You've now styled your webpage using both inline styling and using a style sheet! One last thing to note is the order of precedence, this basically just means if there is styling for a paragraph in the stylesheet and inline, which is used? The answer is the inline styling takes precedence and so will be used instead of the stylesheet.

Feel free to change any of the styling to whatever you want, be as creative as you like!

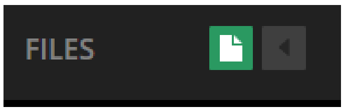
Let’s make is colourful!

Our webpage is looking very empty and dull at the moment, so now we’re going to add some more content to it!

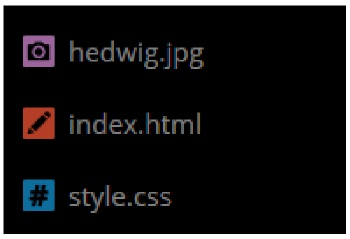
Adding an image

First we’ll add an image. To do this make sure you’ve got an image you want to include, make sure it’s free to use or your own picture! This is a really good website for finding copyright-free images: https://pixabay.com/ or https://www.pexels.com/royalty-free-images/.

To add an image to our webpage, click on this symbol in the top left hand corner. Then select Upload a file from the menu. Click on “From your computer” and navigate to wherever your image is (probably downloads if you’ve just got it!) and click on it.



It should now appear in the files pane.



To get the image to appear on the page, go to your index.html page and insert the following line of code wherever you want your picture to appear. You might notice that we don’t have a closing tag, just a ‘/’ at the end. Although most tags have an opening and closing tag img is one that doesn’t!

```
<img src = “yourfile”/>
```

For my website it would look like this: `<img src= “hedwig.jpg”/>`

In this code src stands for source, which is just saying where to find the image. You can use a file you’ve uploaded, or you could put the url (web address) of a specific image on another website! It’s usually a good idea, if you can, to upload any images you want to use.

Styling our image

Hopefully your image has appeared on your webpage. Chances are the image is much bigger than you want it to be! To change this, we’ll add some styling to our picture!

Navigate to ‘style.css’. Instead of styling a tag type, this time we’ll use a ID selector. You can give images, paragraphs, and pretty much anything an ID. This helps to group things together, for example content on a specific topic or that needs to go in a particular place on your website could all share the same ID.

Using the code below will give all content of id my-image the same styling.

```
#my-image{
      height: 200px;
      width: 100px;
}
```

Then go back to index.html and edit the image tag to look like this:

```
<img id= "my-image" src= "yourimage"/>
```

Check that this has worked and resized your picture. If you aren't happy with the size, then edit the width and height attributes.

Embedding a video or a game

HTML has one tag for embedding content from other websites into your webpage called *iframe*. Iframe has lots of different attributes including height, width and style, allowfullscreen etc.

Below is an example code block you can use to add a video or game to your website. Simply replace the PASTE YOUR LINK HERE part with the url of the video (perhaps from www.youtube.com) or the game (perhaps from www.miniclip.com) you want.

```
<iframe width="560" height="315" src="PASTE YOUR LINK HERE"></iframe>
```

If the above line doesn't work or causes you issues, both YouTube and most gaming websites have a section under the video/game which allows you to share it.

On YouTube, click the share button (on the right below the video) and select embed. This will provide a HTML code snippet which you can copy and paste onto your website. Miniclip has a similar function, below the game it has a button that says "Add to your website". Click here to get a HTML code snippet.

Publish your site

The great thing about Thimble is it allows you to publish your websites onto the internet so others can see it. To do this, go to the top right of the screen and hit the publish button. Hit publish and it'll give you a link to your website. Copy this and paste it into another tab to see your design in all its glory! You can also send this on to others if you want to share it. Make sure you update the published version whenever you've added something and are happy for it to go live.

Hopefully now you have a webpage looks a bit better, but it'll take much more work! Once you're finished, feel free to be creative and add more content, more styling and even more webpages and functionality to your website!

Developed by James Lockwood
& Dr Aidan Mooney

Adding a list

One thing you might want to add to a webpage is a list. There are a number of types of list you can use, these are unordered (which uses bullet points by default), ordered (which uses numbers by default) and definition list (which is used for a list of definitions).

We'll make an unordered list which, in another tutorial, you can convert into a weekly diary. To do this we'll use the tags `<ul></ul>`, an ordered list would be `<ol></ol>` and a definition list is `<dl></dl>`.

To put something into an unordered or ordered list we use a list item tag which is `<li></li>`. For a definition list, you use the `<dt>` tag which defines the term (e.g. football) and `<dd>` which describes the term (e.g. popular sport played with a round ball).

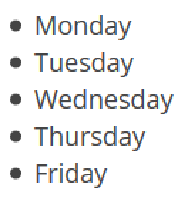
An example of a list could be the days of the week (which you could turn into a diary), your favourite foods, subjects, places you want to visit on holiday, whatever you want!

Below is an example using the days of the week:

```
<ul class= "diary" >

      <li class = "diary" > Monday</li>

      <li class = "diary" >Tuesday</li>

      <li class = "diary" > Wednesday</li>

      <li class = "diary" >Thursday</li>

      <li class = "diary" >Friday</li>

</ul>
```

The reason we've put `class = "diary"` in both our `<ul>` and `<li>` tags is because in this case we only want this list to look like this. You don't have to do this, it just makes it easier for some of the other tutorials! If you copy & paste the code above you'll probably need to change the quotation marks around diary, so delete them and just put in new ones.

Hopefully your code has added something to your webpage that looks something like this:



- Monday
- Tuesday
- Wednesday
- Thursday
- Friday

Styling our list

The list we have is pretty boring at the moment and deosn't look very good. We'll style our list so that it looks a bit better! Click into 'style.css' on the left of the screen. We'll use the `li` selector with a class `diary` to make all the list items have the same style. As with the other styling we've seen, list items can have many different attributes. In the code below we make the text bold, remove the bullet points, choose to display them in one line and give them a margin. We then get the mouse cursor to change to a pointer.

```
li.diary {
        font-weight: bold;
        list-style: none;
        display: inline-block;
        margin:5px;
        cursor: pointer;
   }
```

One useful thing you can do with a list is to then change each item to be uniquely styled. To do this we'll use nth-child(). This is used to style specific list items that are referenced by the order in which they appear in the list. For example, if you wanted to make all the first items in a list a specific colour you would use this:

```
li:nth-child(1){
     color: orangered;
}
```

Give it a go yourself and change the colour of each item! See an example below if you're stuck.

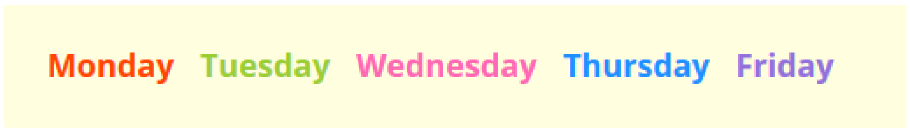
```
li:nth-child(2){
     color: yellowgreen;
```

```
    }
    li:nth-child(3){
          color: hotpink;
    }
    li:nth-child(4){
          color: dodgerblue;
    }
    li:nth-child(5){
          color: mediumpurple;
    }
```

Finally, we can also style the whole list itself. The code below adds padding around the list so that nothing goes on top of it. It the sets a border and changes the background colour. Add this code to your stylesheet!

```
ul.diary{
   padding:20px;
   background: lightyellow;
   border-radius: 5px;
}
```

Hopefully your webpage has something that looks like this now:



That’s how to add lists to your webpage and to style them! With this you can add menus, add different information or make something like a diary! If you want to make a menu or add a more interactive diary see the other tutorials!

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Add a menu

This tutorial will show you how to add a menu to your website. Menus are vital in any website as they allow people to move around from one webpage to another. It also means you can link things you want together on different pages or submenus.

Menu's on most websites are just lists of links. We'll have a look first at how to put a link to another webpage on our site.

Adding a link

Some theory about web addresses

Links are the web's way of addressing certain webpages. They are used to move from one web page to another and are made up of what's called a *URL*. A *URL* looks like the example below, you've probably seen hundreds before!

http://example.ie

URL's are made up of two components, the *protocol identifier* which in our example is HTTP which stands for *Hypertext Transfer Protocol*. This is the way in which the information is sent.

The second component is the *resource name*, for us this would be example.ie.

This resource name is the complete address that you're going to and it can look different depending on the protocol used.

Putting in links using HTML

HTML has a tag which is used for links. The example below shows you how to use it, put this example into your website and check it works!

```
<a href= "www.google.ie"> Click here to go to Google </a>
```

Inside the opening <a> tag we have the word *href* followed by a web address in quotes. You can replace the link with any other address on the web. In between the opening and closing tags we have some text. This is what appears on the website and is known as the *link text*. However, it doesn't have to be text and could be an image for example.

So now you've put in one link, try add a few more to your page!

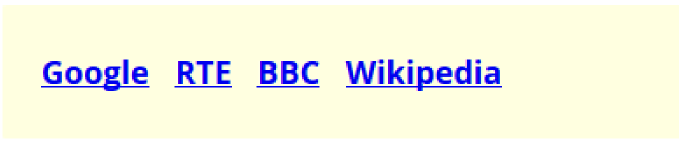
Putting our links into a list

A menu is really just a list of links. We'll take our newly added links and add them into a list. This is exactly like any other list, just instead of text we'll use the <a> tag. You probably want to use an unordered list for now.

Hopefully your code will look like this:

```
<ul class = "menu">

<li class = "menu"><a href= "www.google.ie">Google</a></li>

<li class = "menu" ><a href= "www.rte.ie">RTE</a></li>

<li class = "menu" ><a href= "www.bbc.co.uk">BBC</a></li>

<li class = "menu" ><a href= "www.wikipedia.org">Wikipedia</a></li>

</ul>
```

Now depending on how you've changed your stylesheet then this will appear differently on your webpage. This is what it looks like if you use the same stylesheet as in the Basic Webpage tutorial. But don't worry if it looks different as we're going to style it on its own!



Styling our menu

Open up the CSS of your webpage. We're going to style our menu so that it has its own style, this is just one example of how you could do it so feel free to change and customise it! This styling comes from https://www.w3schools.com/css/css_navbar.asp but is altered slightly.

```
ul.menu {

  list-style-type: none;

  margin: 0;

  padding: 0;

  overflow: hidden;

  background-color: #333;

}
```

This first section styles unordered lists of class menu. We want to remove the bullet points and that's what the first line does. We then set the `padding` and `margin` to 0, this has to do with how much space is around the list. We then hide any `overflow`, this says what should happen if the content was to overflow, i.e. you had too many links in the menu. Finally, we set the menu background colour.

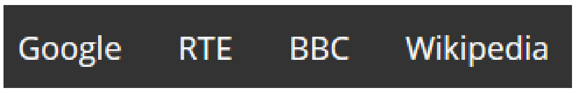
```
li.menu {

  float: left;

}
```

This next section styles all list items of class menu. This `float` command floats the list items over to the left. This `float` command allows you to wrap text around things as well as position things on the page.

```
li a {

  display: block;

  color: white;

  text-align: center;

  padding: 14px 16px;

  text-decoration: none;

}
```

We next style all list items that are inside our link, `<a>`, tags. We want them to be displayed in a block, this puts them alongside one another. We then colour the text and align it to the center. We then give them some `padding` so that they don't overrun onto each other. The `text-decoration: none;` line removes the line underneath the links which usually appears on websites.

Hopefully you now have something that looks like this:



```
li a:hover {

  background-color: #111;

}
```

This last styling block allows us to change the colour of the link items if the mouse is hovering over it. We just change the background colour if the built in hover selector is true.

Setting an Active menu item

Another thing you might want to do, is alter the appearance of the menu so that the user knows which page they're on. To do this, we'll add something to the `<li>` of the menu on the page we're currently at. Go to your index.html file and add another link to your menu which links to "index.html".

Inside the `<a>` tag we'll set it to the class = "active". Now navigate to our CSS file and place the following code:

```
.active {
    background-color: #4CAF50;
}
```

This code will change the colour of the background of the link you assign to active. This won't do anything yet, but if you move onto the next step it'll turn the current page green on the menu.

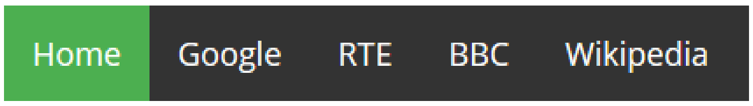
Adding your own webpages to the menu

We now have a "working" menu, but all our menu does is redirect people to other websites, this is no good! Also, with the way Thimble works your links to other sites might not actually work! We want our menu to display the rest of our website.

Firstly, we'll add a link to our Home page. The home page is the first webpage that people will land on, and we can use the one we've been working on. The only difference between adding this link to the list is the web address, it should look like this:

```
<li class="menu"><a href="index.html" class ="active">Home</a></li>
```

Add this line to the top of the list. The address index.html is the name of our file, and we've added the class active so that when people are on the index page the menu should look like this:

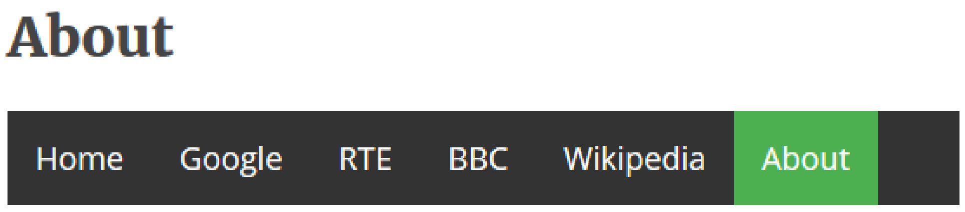


We'll add another link in exactly the same way, but we'll add another of our pages to the menu. If you haven't already, we need to make another webpage. To do this, go to the top left of Thimble and click the New File button. Select Add HTML file. This will make a brand-new html file that it will probably called index-2.html. We're going to turn this into an About page, this is a common page on websites which explains the purpose of the website, company or service that is provided.

To rename the file click on the drop-down arrow on the index-2.html file or right-click on it, select rename and call it about.
We'll also briefly change the content so that it's not the default Thimble page. Change the title and heading to About and put something in the paragraph that describes the purpose of the website.

Now we want to add this to our menu. Navigate back to your index.html file and then add a link to our new page. A hint is that in the `<a>` tag you should have the following code: `href = "about.html"`.

Make sure you publish your changes and check it works. Also, make sure you make the About link active on the About page so that it looks like this:



You've successfully made a menu! Make sure when you make more webpages to add the link to the menu, and also make sure to add the menu to each page so people can easily navigate around your site!

Developed by James Lockwood
& Dr Aidan Mooney

Introduction to Databases Lesson Plan

Strand: Web Development   Duration of Lesson: 40-1+ hour

Learning Outcomes

Students will understand the idea of Databases, what they are and why they might be used.
Students will begin to learn how to construct databases.
Students will learn how to query a database including interpreting queries and writing their own.

Curriculum links

Leaving Certificate Computer Science Specification

1.2 explain how the power of computing enables different solutions to difficult problems

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

3.2 create a basic relational database to store and retrieve a variety of forms of data types

3.7 use algorithms to analyse and interpret data in a way that informs decision-making

Computational Thinking Skills used

Algorithm
Pattern matching

Computer Science concepts/topics involved

Databases
Query languages

Resources / Materials

Pen, paper etc.
Printed Exercise sheets.
Printed sample database.
A way of constructing a database (either a big sheet of paper, whiteboard or Google sheets or equivalent online version.

The Class

Discussion – Introduction – 5-10 minutes

In pairs or whole class, discuss what students initially think of when they hear databases. What do they think it means, why might you use one, can they give any examples?

Britannica.com defines a database as *"any collection of data, or information, that is specially organized for rapid search and retrieval by a computer."* Now if you aren't going to do the lessons on searching and sorting or data then you could discuss why you might want/need to be able to search and sort data. If you have done either of those lessons, remind the students that being able to sort data is vital for many things in life, and being able to search it quickly is also hugely beneficial.
Some examples of large databases include Google, YouTube, Amazon and the CIA. The largest database in the world is the World Data Center for Climate operated by the Max Planck Institute for Meteorology and German Climate Computing Center.
Websites are usually an interface to interact with a database, for example the google search engine allows you to access their database of webpages. Daft.ie allows you to access a database of available houses to rent or buy. Even things like ebay.com and aib.ie are just interfaces to access databases.

Activity – Introduction – 10-20 minutes

Either individually, or in pairs, we're going to get the students to create a database of the class. Assign each person or pair at least one of the below categories. The students then need to go around and ask all of the students to obtain the necessary data to fill our student database. If it suits it might make sense for you to tell the students the categories the lesson before so they can come up with answers. You can then do one of a number of things to create the database.

- You could write it all up on a whiteboard.
- You could take in all of the sheets and then compile the data yourself and show them perhaps next week.
- Create a collaborative spreadsheet (using Google Sheets or similar). This is the method I'd recommend if possible.

Categories:

Name	Date of Birth	Ice cream or chocolate?	What they want to do now (job-wise)
Nationality	Favourite film	Broken a bone before	What they wanted to be when they were 7 (job-wise)
Hair colour	Favourite colour	No. of brothers and sisters	Do you love or hate rollercoasters?
Address	Favourite subject	Favourite sports team	No. of countries visited
Favourite hobby	Eye colour	Favourite TV show	What city would you most like to live in?
Height	Favourite music artist	Favourite day of the year	If you were a superhero, what power would you have?
Age	Favourite drink	Left or right-handed	If you suddenly became a master at woodworking, what would you make?
Favourite food	Dogs or cats?	Favourite book	What pets, if any do you have?

A sample database can be found on the schoology site if you'd prefer to use that in the next section of the activity, this may save time. It could also mean you use the next part as work in another class or as revision/homework. The sample has two sheets, one contains errors to discuss in the following discussion, the next has no such errors to work with the activity below.

Discussion – Development – 5-10 minutes

Once you have a database, we're going to look at the main use of databases, querying them.
Show your class the complete database, if you can have it on the screen or else maybe print off copies of it.
Explain that the main reason databases exist is to allow people to query them. What this means is to retrieve data from the database. You usually want just a subset of the data, especially if you have lots of data.
Imagine having all of the websites on the internet showing up whenever you searched on google! This is why sites have search functions. Another example is a shopping website, such as Amazon. When you enter a search term such as "Harry Potter", that query is sent to the database and all the information linked to the term "Harry Potter" is returned.

Our database is quite simple so we will look at some simple queries you could use with it. Before we do that though, get the students to see if they can see any problems that might be caused by the data. If your students have all answered the questions perfectly and in the same way then there might not be any, but have a look at the sample one if this is the case. Some errors that occur are numbers being written in digits or words, dates written in different formats, invalid answers to questions etc.
Using the sample database as an example, some errors would occur if asking for people whose favourite days are in the first half of the year, or those under 180cm of height. See if the students can come up with other possible issues that could have happened.

Activity - Development – 20-30 minutes

Now you've talked about queries we're going to have a go at making our own. Now these won't work on the spreadsheet or in any database system as it is a query language I have designed for this purpose. It is similar to the popular Database query language SQL.
Our language for querying has a number of commands:

SELECT
This is the actual command that will return the information, this usually goes at the beginning of the query.

ALL
Returns all of the data in the table that fit any conditions that might have been used.

WHERE
This allows you to set constraints.
AND
Exactly as you imagine, allows you to ask for two or more things or check for two or more conditions

OR
Exactly as you imagine, allows you to check for multiple conditions

( )
Use to combine elements of a query

=, >, <, >=, <=
Used to signify equals, greater than, less than, greater than or equal to and less than or equal to.

All the examples below are for the sample database, but they could be adapted to suit your own needs.

The general syntax of a statement is FIND x WHERE y, with x being the column's you want to return the information from and y being the value of specific columns.
Quotation marks must go around the values you're looking for e.g. FIND ALL WHERE Name = "Harry", FIND ALL WHERE Age = "15"

You can return multiple columns by separating them with a comma e.g. SELECT Name, Age

You can specify multiple conditions by separating them with "AND" and "OR", you can group these using brackets e.g. SELECT Name WHERE Age = "16" AND (Height < "160" OR Height > "180")

Exercise 1

1. If we wanted to display all of the information from our database, we would use this command
   SELECT ALL
   This will return all information from the database.

2. If we wanted to return all information about all 15 year olds we would use
   SELECT ALL WHERE Age = "15"
   This will return all the information in rows 2 and 8 (Adam and Harry).

3. If we wanted to return all information on all of those whose favourite TV show is Friends we would use:
   SELECT ALL WHERE Fav TV show = "Friends"
   This will return all the information in row 2 (George).

4. If we wanted to just return the names of those who like Pizza, we would use:
   SELECT Name WHERE Fav food = "Pizza"
   This will return Adam Anderson and Harry.

5. If we wanted to return the ages, favourite foods and favourite hobbies of everyone we would use:

SELECT Name, Age, Fav food, Fav TV show

This will return all the rows names, ages, food and favourite TV shows e.g. Adam, 15, Pizza and The IT Crowd.

6. If we wanted to return all information about people who are above 170cm and like Dogs, we would use:
   SELECT Name WHERE Height> "170"AND Dogs or cats = "Dogs"
   This will return Adam, George and Harry.

7. If we wanted to return the names of people who like playing video games or watching films, we would use:
   SELECT Name WHERE Fav Hobby = "Video games" OR Fav Hobby = "Watching films"
   This will return Adam, Fiona and Harry.
   Note: You must give the column you're picking from each time

Have students look through these examples and write out the information that is returned for each of them on the answer sheets (answers below each question). For those where it asks for ALL just get them to write the names of the rows returned but write in their answer that it should be all columns.

Exercise 2

Now we're going to give writing our own commands a go. These could be done in pairs as well or individually.

Return all information of students who are 17.

SELECT ALL WHERE Age = "17"

Return all information of students who are over 15.

SELECT ALL WHERE Age > "15"

Return student's names who are under 17 and over 160cm tall.

SELECT Name WHERE Age < "17" AND Height > "160"

Return names, ages and heights of those students who like Dogs or Cats.

SELECT Name, Age, Height WHERE Dogs or cats = "Dog" OR Dogs or cats = "Cats"

Return names, favourite hobbies and superhero powers of students who have visited more than 5 countries.

SELECT Name, Fav hobby, Superhero powers WHERE Countries visited > "5"

Return all information of those students who like Dogs and Rollercoasters or who like neither Dogs or Cats and have never been on a rollercoaster.

SELECT ALL WHERE (Dogs or cats = “Dogs” AND Rollercoasters = “Love”) OR (Dogs or cats = “Neither” AND Rollercoasters = “Never been on one”)

Feedback – Conclusion – 5 minutes

Please get your students to fill out the feedback form.

Additional Activities

Activity – Development – 10-30 minutes

As the examples are using just a small set of the columns, there is potential to have much more complex or specific queries. If there seems to be interest from the students after completing the above examples you could pair them up and get them to come up with queries or requests to swap between them.

This would involve them either writing using the query language and writing statements which other pairs must then answer with the data shown, or writing questions and the other pairs then writing queries to return the requested data. This would be much more interesting if you actually used your classes unique database.

Maynooth University
National University of Ireland Maynooth

Developed by James Lockwood
& Dr Aidan Mooney

Database Exercise sheets

Exercise 1

What do the following commands return? For answers where it asks for ALL information, just write the names but state that all columns would be returned.

SELECT ALL

SELECT ALL WHERE Age = "15"

SELECT ALL WHERE Fav TV show = "Friends"

SELECT Name WHERE Fav food = "Pizza"

SELECT Name, Age, Fav food, Fav TV show

SELECT Name WHERE Height> "170"AND Dog or cats = "Dogs"

SELECT Name WHERE Fav Hobby = "Video games" OR Fav Hobby = "Watching films"

Exercise 2

Write commands to return the information asked for below.

All information of students who are 17.

All information of students who are over 15.

The names of people who are under 17 and over 160cm tall.

The names, ages and heights of people who like Dogs or Cats.

The names, favourite hobbies and superhero powers of people who have visited more than 5 countries.

All information of people who like Dogs and Rollercoasters or who like neither Dogs or Cats and have never been on a rollercoaster.

PERSONAL FORM

Personal Form

Please answer all questions required. Hit submit when completed.

* Required

1. **Participant ID** *

2. **School** *

3. **Gender** *
   Check all that apply.
   - ☐ Male
   - ☐ Female

4. **Age** *
   Mark only one oval.
   - ◯ 12
   - ◯ 13
   - ◯ 14
   - ◯ 15
   - ◯ 16
   - ◯ 17
   - ◯ 18+

5. **What year of school are you currently in?** *
   Mark only one oval.
   - ◯ 1st Year
   - ◯ 2nd Year
   - ◯ 3rd Year
   - ◯ 4th Year
   - ◯ 5th Year
   - ◯ 6th Year

6. **Are you a native English speaker?** *
Mark only one oval.
- Yes
- No

7. **If no, do you sufficient mastery to study in Ireland? (as measured by a recognised English examination eg. score of 6.0 UG or 6.5 PG IELTS examination)**
Mark only one oval.
- Yes
- No

8. **Would you say you are a morning or night person?** *
Mark only one oval.
- Morning
- Night
- No preference

9. **Do you own a personal smartphone?** *
Mark only one oval.
- Yes
- No

10. **If no, do you have access to one at home?**
Mark only one oval.
- Yes
- No

11. **How many hours per day do you spend on a smartphone?** *
Mark only one oval.
- N/A
- Less than one hour
- 1-3 hours
- More than 3 hours

12. **What do you spend most of your time doing on a smartphone? (select up to 3)**
Check all that apply.

- ☐ Texting (SMS)
- ☐ Calling (using the phones network)
- ☐ Other forms of messaging (Facebook Messenger, Whatsapp, Viber etc.)
- ☐ Facebook
- ☐ Snapchat
- ☐ Instagram
- ☐ Browsing the web (using Safari/Chrome etc.)
- ☐ Playing games
- ☐ Looking up news/information (BBC, RTE, Wikipedia etc.)
- ☐ Taking photos
- ☐ Sending/reading emails
- ☐ Reading books/newspapers etc.
- ☐ Listen to music (ITunes, Spotify)
- ☐ Watch videos (using Netflix, YouTube etc.)
- ☐ Other: ____________________

13. **Do you own a personal laptop/desktop PC?** *
Mark only one oval.

- ◯ Yes
- ◯ No

14. **If no, do you have access to one at home?**
Mark only one oval.

- ◯ Yes
- ◯ No

15. **How many hours per day do you spend on a laptop/PC?** *
Mark only one oval.

- ◯ N/A
- ◯ Less than one hour
- ◯ 1-3 hours
- ◯ More than 3 hours

16. **What do you spend most of your time doing on a laptop/computer? (select up to 3)**
Check all that apply.

- [ ] Instagram
- [ ] Browsing the web (using Safari/Chrome etc.)
- [ ] Facebook & other social media
- [ ] Playing games
- [ ] Looking up news/information (BBC, RTE, Wikipedia etc.)
- [ ] Sending/reading emails
- [ ] School work or similar (using Word, Powerpoint etc.)
- [ ] Listening to music
- [ ] Watching videos (using YouTube, Netflix etc.)
- [ ] Other: ______________________

17. **Do you own a personal tablet?** *
Mark only one oval.

- ( ) Yes
- ( ) No

18. **How many hours per day do you spend on a tablet?** *
Mark only one oval.

- ( ) N/A
- ( ) Less than one hour
- ( ) 1-3 hours
- ( ) More than 3 hours

19. **What do you spend most of your time doing on a tablet? (select up to 3)**
Check all that apply.

- [ ] Instagram
- [ ] Browsing the web (using Safari/Chrome etc.)
- [ ] Facebook
- [ ] Playing games
- [ ] Looking up news/information (BBC, RTE, Wikipedia etc.)
- [ ] Sending/reading emails
- [ ] Snapchat
- [ ] Reading books/newspapers etc.
- [ ] Some form of texting/messaging (Facebook Messenger, Skype, Viber etc.)
- [ ] Listening to music
- [ ] Watching videos (using YouTube, Netflix etc.)
- [ ] Other: ______________________

20. **Have you had previous programming experience?** *
Mark only one oval.

- ( ) Yes
- ( ) No

21. **How much programming experience do you have?** *
Mark only one oval.

- ( ) Never programmed before
- ( ) I have done programming once or twice.
- ( ) I have done programming a number of times
- ( ) I have been programming for over a year

22. **Which programming language/s have you used before?**
Check all that apply.

- [ ] Java
- [ ] Python
- [ ] Scratch
- [ ] C/C++
- [ ] Javascript
- [ ] Other: ______________________

23. **How often do you program?** *
Mark only one oval.

	1	2	3	4	5	
Not at all	( )	( )	( )	( )	( )	Daily

24. **If you have programmed before, how would you rate your programming level?**
Mark only one oval.

	1	2	3	4	5	
Very poor	( )	( )	( )	( )	( )	Very good

25. **Have you ever done any website/app development?** *
Mark only one oval.

- ( ) Yes
- ( ) No

26. **If yes, which of the following tools/technologies have you used?**
Check all that apply.

- ☐ HTML
- ☐ CSS
- ☐ Javascript
- ☐ PHP or other server-side langauge
- ☐ App Inventor
- ☐ Android Studio
- ☐ XCode (Apple App Development tool)

27. **Have you ever heard of any of the following?** *
Check all that apply.

- ☐ Bebras challenge
- ☐ Codecademy
- ☐ Khan Academy
- ☐ Udacity
- ☐ Kangaroo Maths
- ☐ Call to Code
- ☐ Hour of Code (code.org)
- ☐ AILO (All Ireland Linguistics Olympiad)
- ☐ None of the above
- ☐ Other: ______________________

28. **What level of maths did you take/are you taking for Junior Cert?** *
Mark only one oval.

- ○ Foundation
- ○ Ordinary
- ○ Higher

29. **Do either of your parents work in an IT related job?** *
Examples include Software Developer, Systems administrator etc.
Mark only one oval.

- ○ Yes
- ○ No
- ○ Not sure

30. **Any other comments**

H

STUDENT FEEDBACK SURVEY

Algorithms 2 assessment

* Required

1. **Participant code** *

2. **Did you enjoy this class?** *
Mark only one oval.

	1	2	3	4	5	
I did not enjoy this class at all	◯	◯	◯	◯	◯	I really enjoyed this class

3. **What did/didn't you enjoy about this class?**

4. **Do you feel like you learned something in this class?** *
Mark only one oval.

◯ Yes
◯ No
◯ Not sure

5. **If yes, what did you learn?**

6. **If no, why not?**

7. **Any other questions/comments**

Powered by
Google Forms

I

TEACHER FEEDBACK FORM

Intro to CS feedback

Please be honest with your feedback as this will help us to improve the course for future years.
All feedback will be dealt with confidentially.

* Required

1. **School** *

2. **Type of school** *
   Please select all that apply
   Check all that apply.
   - ☐ Mixed
   - ☐ Boys
   - ☐ Girls
   - ☐ Private
   - ☐ Public

3. **Approx. number of students who took part in the class** *

4. **Date of class**

   Example: December 15, 2012

5. **How long did you spend on this class/lesson?** *

6. **Do you feel like your students enjoyed the class?** *
   Mark only one oval.
   - ◯ Yes
   - ◯ No
   - ◯ Not sure

7. **What and why did they enjoy? Or what and why didn't they enjoy?**

8. **Do you feel like your students learned something in the class?** *
Mark only one oval.

- Yes
- No
- Not sure

9. **If yes, what? If no, why not?**

10. **Was the lesson plan and other resources useful/helpful?** *
Mark only one oval.

- Yes
- No
- Some parts were helpful

11. **Would you add/take away anything from the lesson? If either, what?**

12. **Did you feel like you needed any other resources, help or guidance to deliver this lesson?** *
Mark only one oval.

- Yes, more would have been helpful
- No, everything given was sufficient

13. **If yes, what would you have liked?**

14. **Would you teach this lesson again?** *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ Maybe

15. **Any other thoughts or comments**

J

VIEWS OF COMPUTER SCIENCE SURVEY

View of Computer Science

Please answer all questions required. Hit submit when completed.

* Required

1. **Participant ID** *

2. **School** *

3. **Would you/have you considered studying Computer Science in college/university?** *
Mark only one oval.

- ○ Yes
- ○ No
- ○ Maybe

4. **What do you think of when you hear Computer Science?** *

5. **Is Computer Science interesting to you?** *
Mark only one oval.

- ○ Yes
- ○ No
- ○ Maybe

6. **If yes/no, why?**

7. **Is Computer Science challenging?** *
Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ Maybe

8. **If yes/no, why?**

____
____
____
____
____

9. **Using the internet is central in Computer Science** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

10. **Using Word, Excel etc. is central in Computer Science** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

11. **Installing software (eg. Windows, iTunes) is central in Computer Science** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

12. **Programming is central in Computer Science** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

13. **Being able to solve different problems is central in Computer Science** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	⬭	⬭	⬭	⬭	⬭	Strongly agree

14. **Computer Science is an area related to maths** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	⬭	⬭	⬭	⬭	⬭	Strongly agree

15. **A computer scientist should be good at working with others** *
Mark only one oval.

	1	2	3	4	5	
Strongly disgree	⬭	⬭	⬭	⬭	⬭	Strongly agree

16. **Boys/men are more likely to study Computer Science than girls/women** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	⬭	⬭	⬭	⬭	⬭	Strongly agree

17. **Work in Computer Science can be done without a computer** *
Mark only one oval.

	1	2	3	4	5	
Strongly disagree	⬭	⬭	⬭	⬭	⬭	Strongly agree

18. **Have you ever heard the term Computational Thinking?** *
Mark only one oval.

⬭ Yes
⬭ No

19. **If yes, what do you think it is/means?**

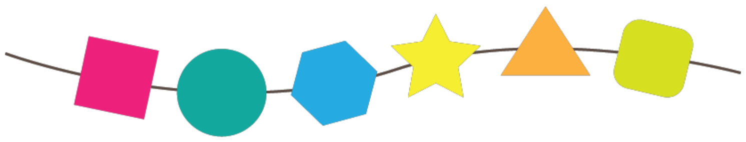
20. **Any further questions or comments**

K

BEBRAS TEST 1

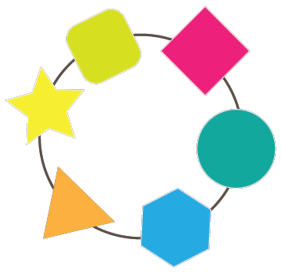
Bracelet

Emily has broken her favourite bracelet. The broken bracelet now looks like this:

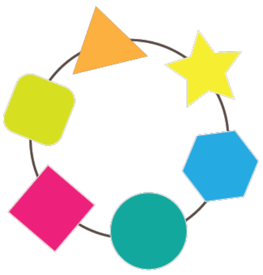


Question:

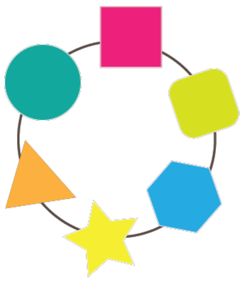
Which of the following four bracelets shows what the bracelet looked like when it was whole?



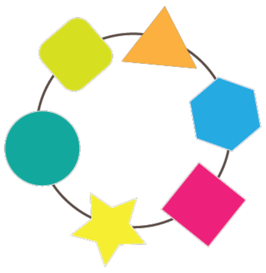
A



B



C



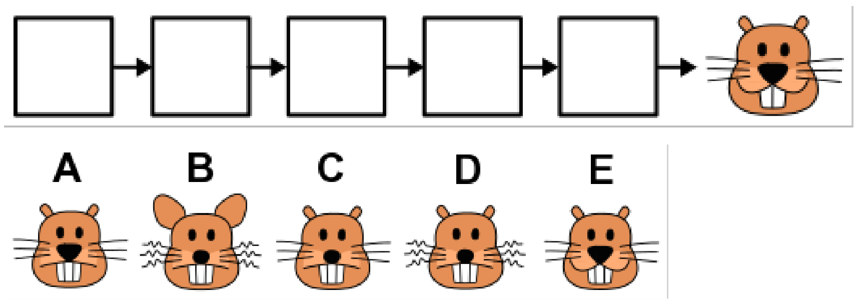
D

Animation

Taro is planning an animation of a face that is made from a sequence of pictures.

To make the animation run smoothly, only one feature of the face should change from one picture to the next.

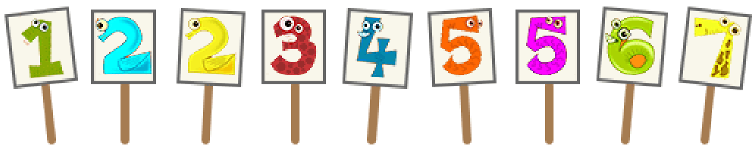
Unfortunately, the pictures got mixed up. Now Taro must find the correct order again. Luckily, he knows which picture is last.



Question:

Put the pictures in the correct order so that Taro makes the correct animation.

Animal Competition



The beavers and dogs had a competition. In total nine animals took part.

The nine participants had the following scores: 1, 2, 2, 3, 4, 5, 5, 6,

7. No dog scored more than any beaver.

One dog tied with a beaver.

There were also two other dogs that tied with each other.

Question:

How many dogs took part in the competition?

2, 3, 5, 6 or
7

Cross Country

Three very fast beavers will compete in a cross-country run.

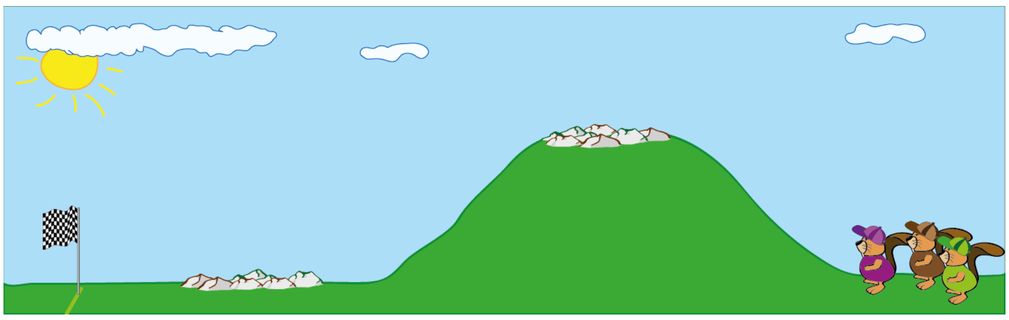
Mr. Brown will overtake one beaver when running uphill.

Mrs. Pink will overtake one beaver when running downhill.

Mrs.Green will overtake one beaver when running across rocks.

The terrain is shown in the picture: uphill, followed by some rocks, downhill and then some more rocks.

Mrs. Pink starts in the first position, followed by Mr. Brown and Mrs. Green.



Question:

In which order will the beavers finish the race?

A. Mrs Pink, Mr Brown, Mrs
Green B. Mr Brown, Mrs Pink,
Mrs Green C. Mr Brown, Mrs
Green, Mrs Pink D. Mrs Green,
Mrs Pink, Mr Brown

Stack Computer

The Stack Computer is loaded with boxes from a conveyer belt. The boxes are marked with a Number or an Operator (+, -, * or /).

The computer is loaded until the top box is a box marked with an operator. This operator is then used on the two boxes below it. The three boxes are then fused into one single box and marked with the outcome of the calculation.

In the Stack Computer, calculations are entered in a different way to a normal calculator.

<u>Examples</u>:

2+3 must be entered as 2 3 +

10-2 must be entered as 10 2 -

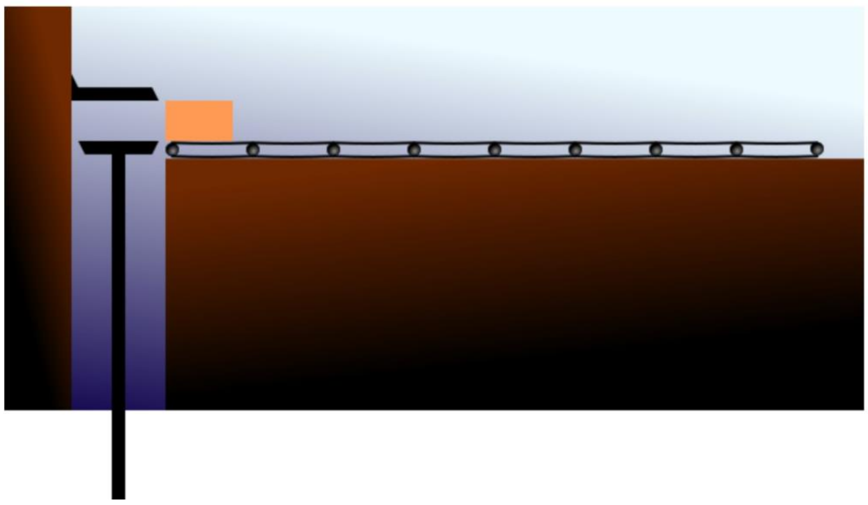
5*2+3 must be entered as 5 2 * 3 +

5+2*3 must be entered as 5 2 3 * +

(8-2)*(3+4) must be entered as 8 2 - 3 4 + *

Question:

How should the following computation be entered: 4*(8+3)-2?

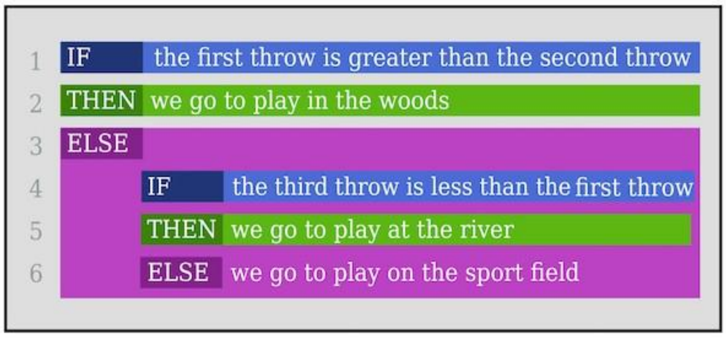


Throw the Dice

After school the young beavers often play together.

To avoid quarrels about where to play, they throw a normal six sided die.

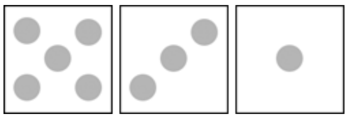
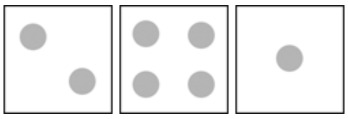
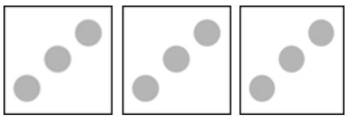
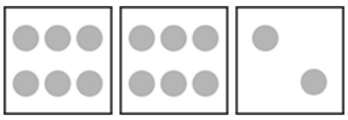
The decision is found according to this rule:



```
IF the first throw is greater than the second throw
THEN we go to play in the woods
ELSE
    IF the third throw is less than the first throw
    THEN we go to play at the river
    ELSE we go to play on the sport field
```

Question:

Which sequence of throws will send the young beavers to the sports field?

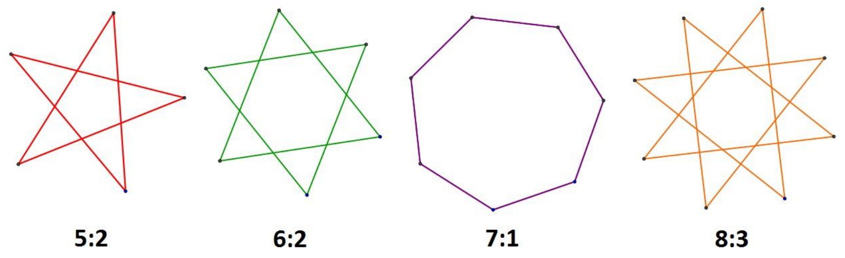
Drawing Stars

Stella the beaver loves to draw stars. She has devised a system for labelling her stars according to their shape. She uses two numbers:

A number of dots for the star.

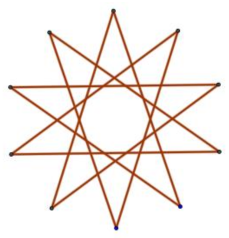
A number indicating if a line from a dot is drawn to the nearest dot (the number is 1), the second closest dot (the number is 2), etc.

Here are four examples of Stella's labelling system:



Question:

How would Stella label the following star?



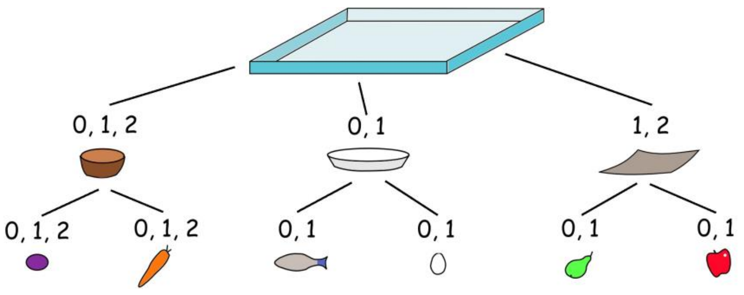
9:3   9:4   10:4 or 10:5

Beaver Lunch

Hm, what to take for lunch today?

The cafeteria gives instructions on how to choose a Beaver lunch.

This is shown as a diagram:



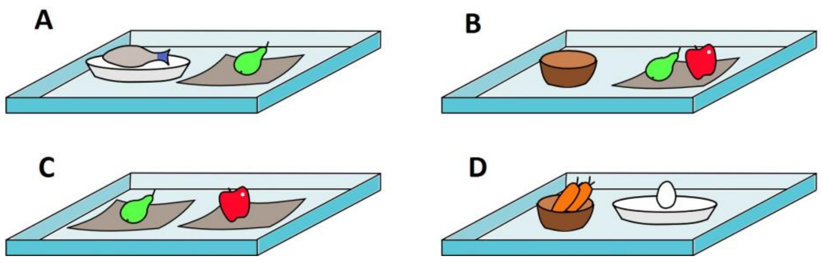
Below the tray you see different types of food containers.

The numbers indicate how many containers of this type can be added to a tray.

Each container can only have food items put in it that are shown below it.

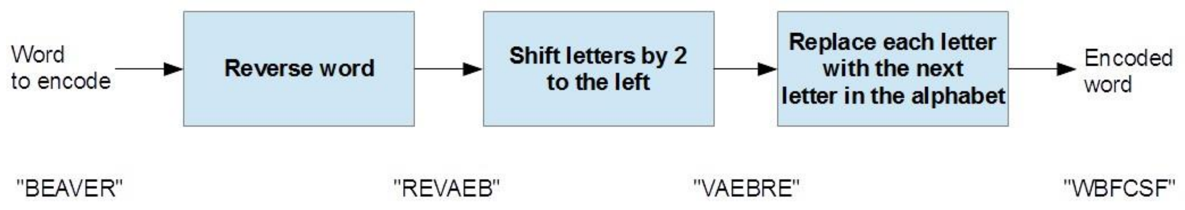
The numbers indicate how many food items of this type can be added to the containers.

Question:
Which of the following lunches is not a proper Beaver lunch?



You Won’t Find It

Beaver Alex and beaver Betty send each other messages using the following sequence of transformations on every word.


Word to encode
Reverse word
Shift letters by 2 to the left
Replace each letter with the next letter in the alphabet
Encoded word
"BEAVER"
"REVAEB"
"VAEBRE"
"WBFCSF"

For example, the word "BEAVER" is transformed to "WBFCSF".

Beaver Betty receives the encoded message "PMGEP" from beaver Alex.

Question:

What did Alex want to say?

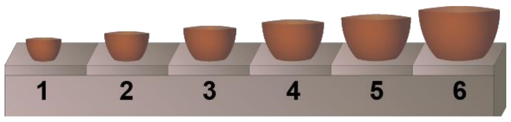
RIVER, KNOCK, FLOOD or LODGE

Bowl Factory

A factory produces sets of 6 bowls of different sizes. A long conveyor belt moves the bowls one by one, from left to right.

Bowl production places the 6 bowls of each set onto the conveyor belt in a random order.

Before packing the bowls, they need to be sorted to look like this:

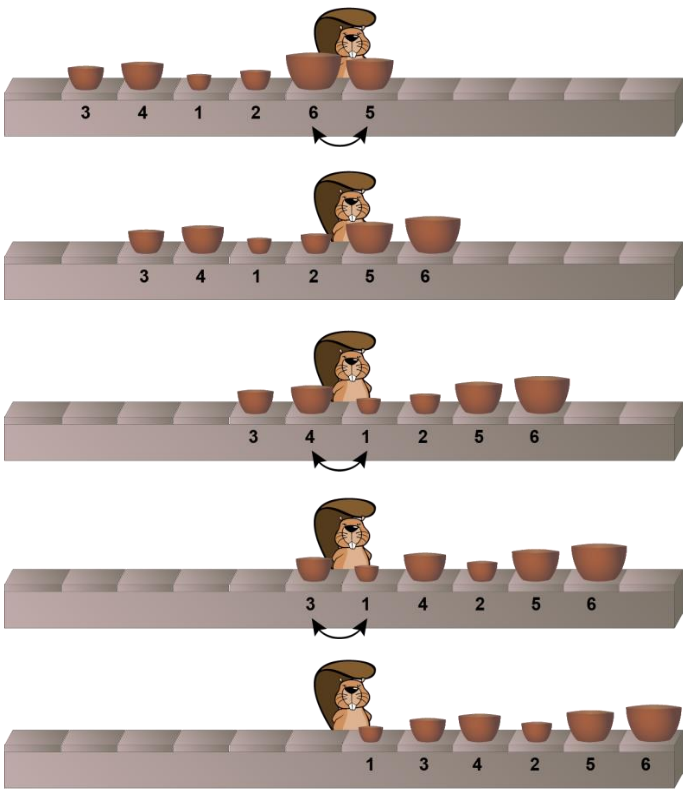


To help with the sorting, the factory places workers along the conveyor belt.

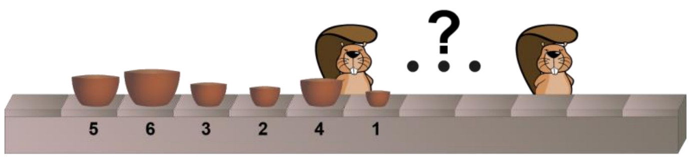
When a set of bowls passes a worker, the beaver will swap any two neighbouring bowls which are in the wrong order.

The worker will keep doing this until the set of 6 bowls has finished passing.

See how the order of a set of bowls changes as it passes one worker:



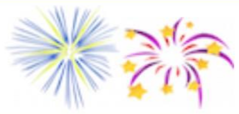
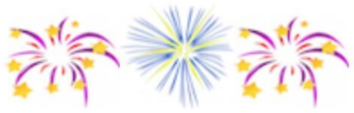
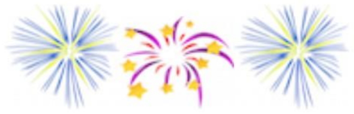
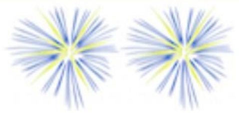
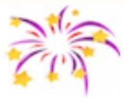
Question:
How many workers should be put along the line to sort the set of bowls on the right?



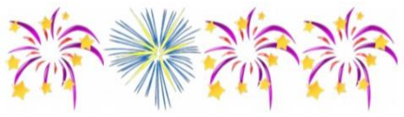
Fireworks

Two beavers live in lodges separated by a large forest.
They decide to send messages to each other by shooting fireworks into the sky above the trees.

Each message is a sequence of words, though the beavers only know five different words.
The beavers can shoot two types of fireworks, one after the other, and know the following codes:

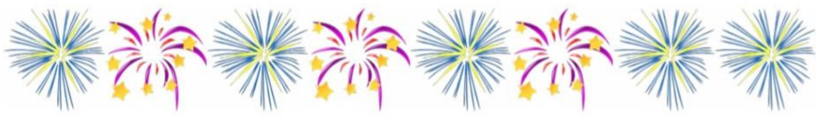
Word	Code
Log	
Tree	
Rock	
River	
Food	

For example, to send the (rather strange) message "food, log, food", a beaver would shoot:



Question?

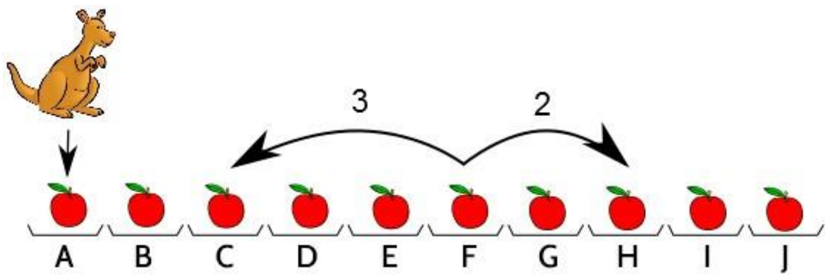
How many **different** meanings can the following sequence of fireworks have?



0, 1, 2, 3, or 4

Kangaroo

There are 10 plates in a row. There is one apple on each plate.



Thomas the kangaroo loves to jump. First, he jumps onto the leftmost plate with the letter A.

On each single jump after this, he either jumps forward two plates, or backwards three plates. (An example of the two possible jumps from one plate is shown with arrows in the picture.)

Thomas only jumps onto plates with an apple.
If he jumps onto a plate, he collects the apple from it.

Question:

If Thomas collects all 10 apples, which apple does he collect last?

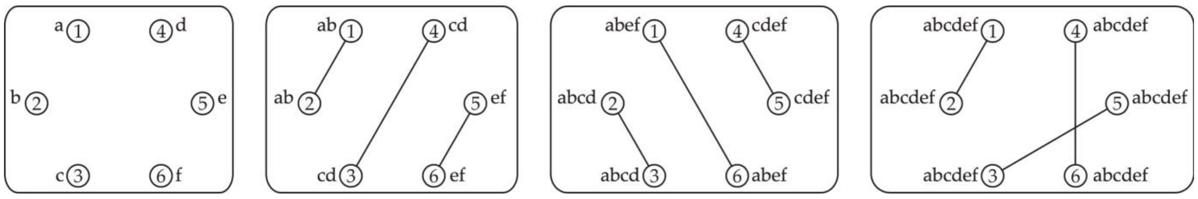
A, B, C, D, E, F, G, H, I or J

Spies

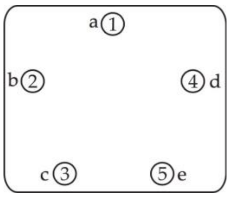
Every Friday, six spies exchange all the information they have gathered during the week. A spy can never be seen with more than one other spy at the same time. So, they have to have several rounds of meetings where they meet up in pairs and share all the information they have at that point.

The group of 6 spies needs only three rounds to distribute all their secrets:

Before the meetings each spy holds a single piece of information. (spy 1 knows 'a', spy 2 knows 'b, etc.). In the first round spies 1 and 2 meet and exchange information so now both know 'ab'. The diagrams show which spies meet in each round with a line. It also shows which pieces of information they all have. After three rounds all information has been distributed.



After an international incident one spy has stopped attending the meetings. What is the minimum number of rounds needed for the five remaining spies to exchange all information?

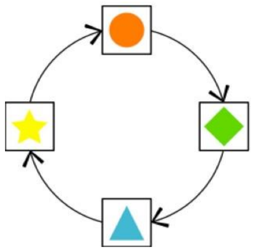


L

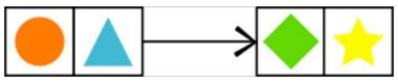
BEBRAS TEST 2

Bebras Painting

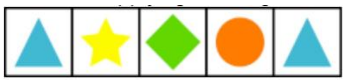
The Beaver children have found a magic roller.
The roller replaces a shape in a painting with the next shape shown by the arrows below.

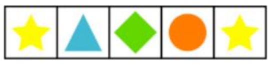


Example:
When Ben uses the magic roller to paint over the painting on the left, he gets the painting on the right.

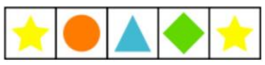


Question:
What will the painting below look like after using the magic roller?

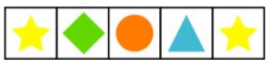




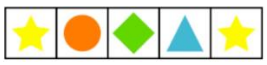
A



B



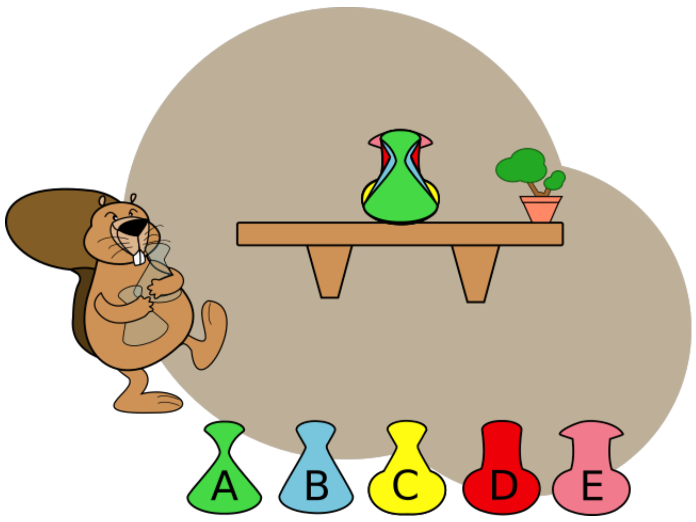
C



D

Bottles

A Beaver puts five bottles on a table.
He places them so that every bottle has a bit showing.
He places the first bottle at the back of the table and puts each new bottle in front of those already placed.



Question:
In what order are the bottles placed when they appear as shown in the picture?

E D C B A
D B C A E
E C D A B
D C E B A

Party Guests

To arrange a dinner party Sara the beaver needs to talk to five friends:

Alicia, Beat, Caroline, David and Emil.

Sara can talk to Emil right away. However, to talk to her other friends, there are a few points to consider:

1- Before she talks to David, she must first talk to Alicia.
2- Before she talks to Beat, she must first talk to Emil.
3- Before she talks to Caroline, she must first talk to Beat and David.
4- Before she talks to Alicia, she must first talk to Beat and Emil.

Question:

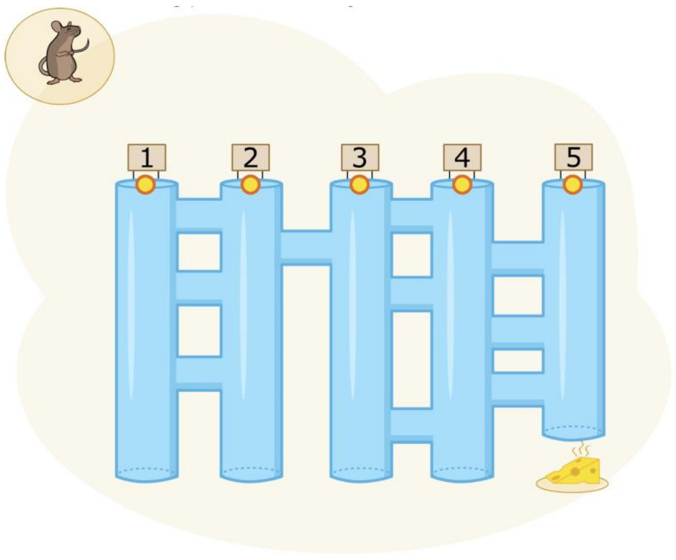
In what order should Sara talk to all of her friends if she wants to talk to all of them? Write them down in the correct order.

Alicia	**Beat**	**Caroline**	**David**	**Emil**

Tube System

A mouse is at the entrance of a tube system. It wants to reach the cheese at the end of tube 5. The mouse always follows these commands:

1. Go downwards until a crossing
2. At the crossing, move through to the next vertical tube
3. Go to command 1



Question:
In which tube should the mouse start so that it reaches the cheese?

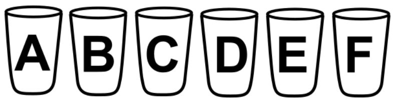
1 2 3 4 or 5

Magic Potion

Betaro Beaver has discovered five new magic potions:

> one makes ears longer
> another makes teeth longer
> another makes whiskers curly
> another turns the nose white
> the last one turns eye's white.

Betaro put each magic potion into a separate beaker. He put pure water into another beaker, so there are six beakers in total. The beakers are labeled A to F. The problem is, he forgot to record which beaker contains which magic potion!

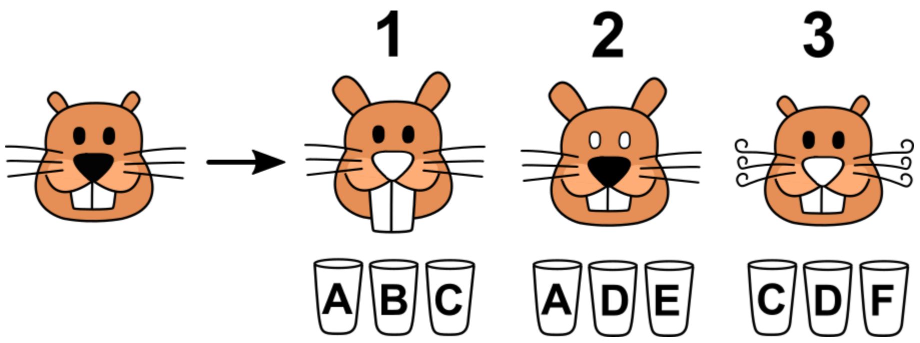


To find out which potion is in each beaker, Betaro set up the following experiments:

1: A beaver drinks from beakers A, B and C together - the effects are shown in Figure 1.

2: A beaver drinks from beakers A, D and E together - the effects are shown in Figure 2.

3: A beaver drinks from beakers C, D and F together - the effects are shown in Figure 3.



Question:
Which beaker contains pure water?

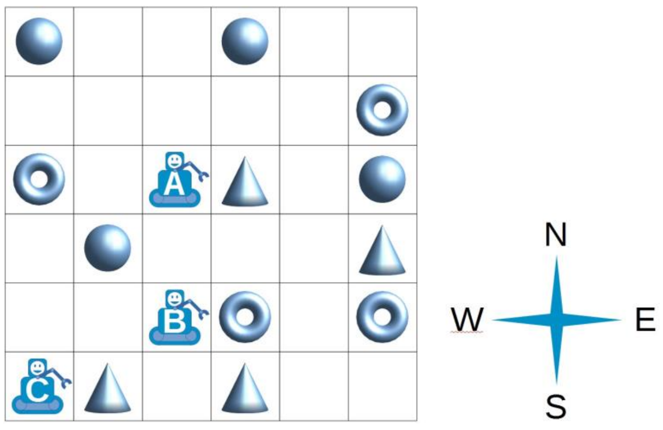
Concurrent Directions

In a warehouse, three robots always work as a team.

When the team gets a direction instruction (N, S, E, W), all robots in the grid will move one square in that direction at the same time.

After following a list of instructions, the robots all pick up the object found in their final square.

For example, if we give the list N, N, S, S, E to the team, then robot A will pick up a cone, robot B will pick up a ring, and robot C will pick up a cone.

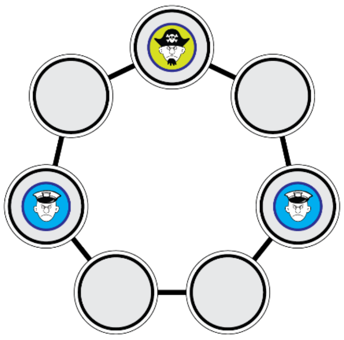


Question:

Which list of instructions can be sent to the robots so that the team picks up exactly a sphere, a cone, and a ring?

- **A.** N, E, E, E
- **B.** N, E, E, S, E
- **C.** N, N, S, E, N
- **D.** N, E, E, S, W

Pirate Hunters



In the game of Pirate Hunters players take turns moving a Pirate or a Policeman.

When it is the police's turn, the player moves a policeman over to a neighbouring circle.

The pirate is faster than a policeman, and skips a circle on his turn, moving two circles.

A policeman cannot move to a circle that is occupied -- either by his colleague policeman, or the pirate.

The game ends when the pirate is forced to move to a circle occupied by one of the policemen. The policeman goes first.

Question:

If the pirate plays the best way possible and makes no mistakes, how many moves will it take the police to capture him?

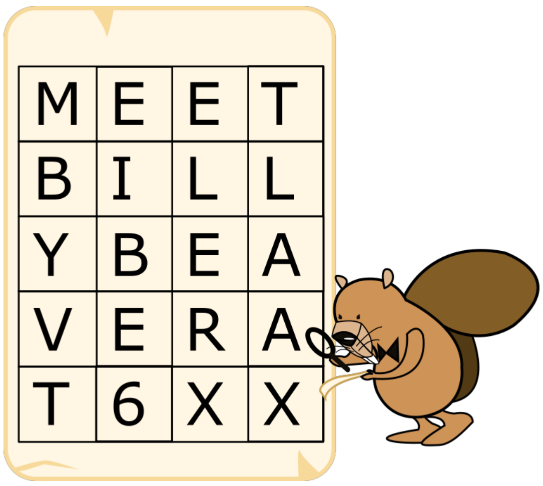
A. The police win in 2 turns
B. The police win in 3 turns
C. The police win in 5 turns
D. The police have no chance of winning

Secret Messages

Agents Boris and Bertha communicate using secret messages. Boris wants to send Bertha the secret message:

MEETBILLYBEAVERAT6

He writes each character in a 4 column grid from left to right and row by row starting from the top. He puts an X in any unused spaces. The result is shown below.



M	E	E	T
B	I	L	L
Y	B	E	A
V	E	R	A
T	6	X	X

Then he creates the secret message by reading the characters from top to bottom and column by column starting from the left:

MBYVTEIBE6ELERXTLAAX

Bertha then uses the same method to reply to Boris. The secret message she sends him is:

OIERKLTEILH!WBEX

Question:
What message does Bertha send back?

OKWHERETOMEET!
OKIWILLBETHERE!
WILLYOUBETHERETOO?
OKIWILLMEETHIM!

Theatre

Three spotlights are used to light the theatre stage in the beavers' forest, a red one, a green one and a blue one.

The colour of the stage depends on which of the three spotlights are turned on. This table shows the possible combinations of colours.

Red light	Green light	Blue light	Stage colour
off	off	off	Black
off	off	on	Blue
off	on	off	Green
off	on	on	Cyan
on	off	off	Red
on	off	on	Magenta
on	on	off	Yellow
on	on	on	White

From the beginning of the show, the lights will be switched on and off in this pattern:

- The red light repeats the sequence: two minutes off, two minutes on.
- The green light repeats the sequence: one minute off, one minute on.
- The blue light repeats the sequence: four minutes on, four minutes off.

Question:

What will the colour of the stage be in the first 4 minutes of the show? Write the colours out in the correct order.

Black Blue Green Cyan Red Magenta Yellow White

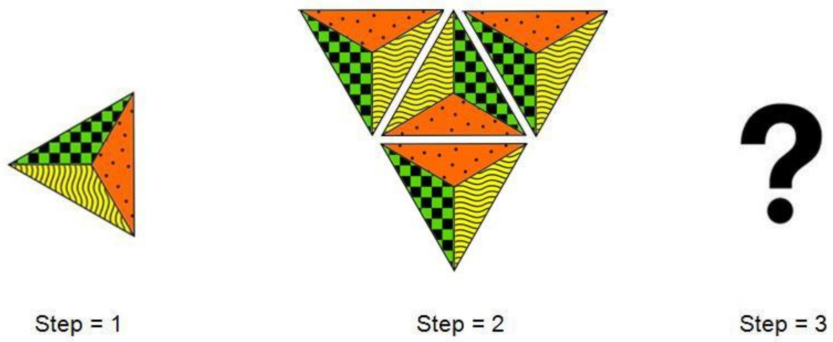
Minute 1	Minute 2	Minute 3	Minute 4

Triangles

A beaver wants to create a mosaic with identical, triangle-shaped tiles.

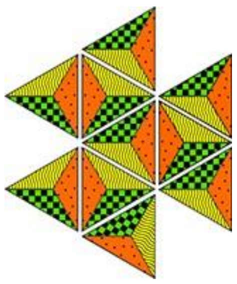
He starts with one tile. He rotates it 90 degrees clockwise and then adds tiles on each side of the triangle-shaped tile, as shown in the picture below.

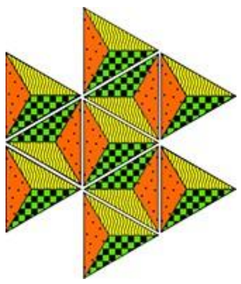
Then he rotates the whole shape 90 degrees clockwise again and adds tiles to the sides as before.

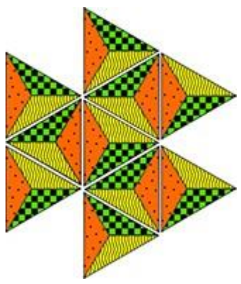


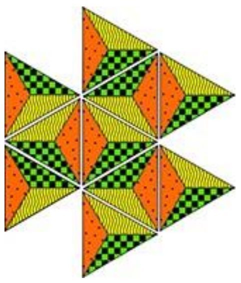
Question:

What will be the final shape of the triangles after step 3?


a


b


c


d

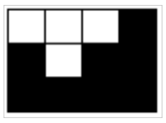
Scanner Code

Two scanners encode an image by translating its pixels into a special code. The code lists the number of all consecutive pixels of the same colour (black or white), followed by the number of all the consecutive pixels of the other colour, and so on. Both scanners start from the top left corner, and go from left to right, and row by row.

The two scanners use different methods at the end of a row:
Scanner A processes the pixels row by row and restarts the encoding on the next row.
Scanner B processes the pixels row by row but does not restart the encoding on the next row.

Example:

The image on the right would be represented by the following codes:

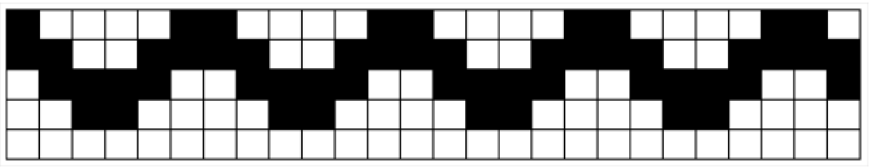


Scanner A: 3,1,1,1,2,4 (3 white, 1 black, 1 black; 1 white, 2 black, 4 black)
Scanner B: 3,2,1,6. (3 white, 2 black, 1 white, 6 black)

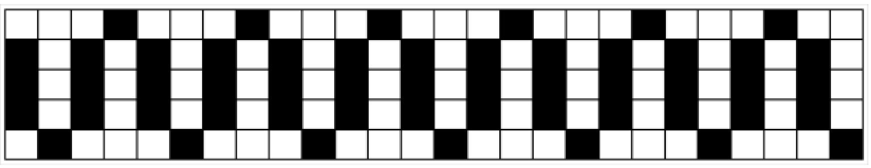
Question:

Which of the following pictures will have the same code no matter which scanner is used?

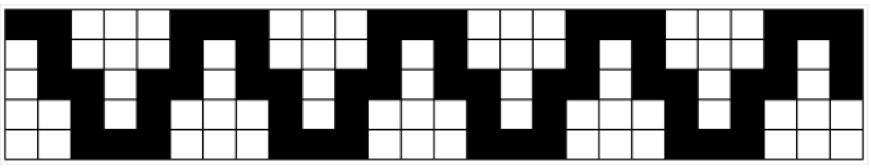
A



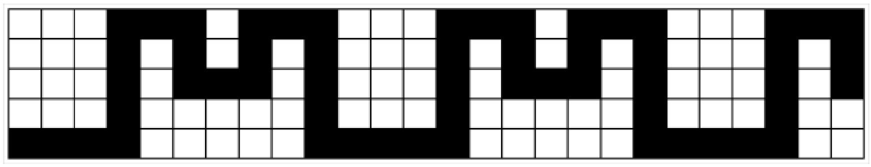
B



C



D



The Game

Beaver Big is playing a game with Beaver Small on the special game board shown.

They start from the leftmost box (Box 5). **Beaver Big goes first.**

She can choose to move Up or Down:
Up will move to Box 4.4; Down will move to Box 5.7.

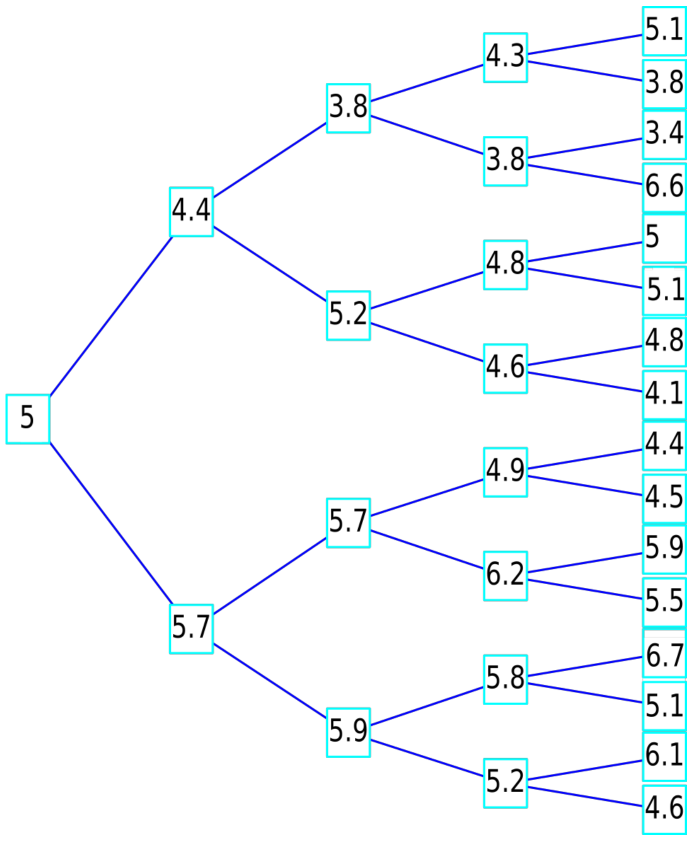
Then, it is Beaver Small's turn to choose Up or Down. From then on they take turns until, finally, Beaver Small chooses a box in the rightmost column.

Because both beavers can see all the numbers on the game board all the time, they are able to plan their moves accordingly.

Question:

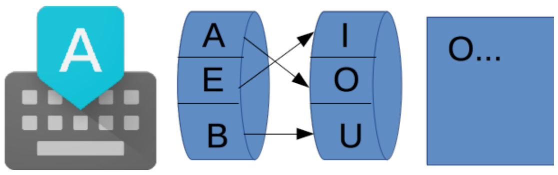
Beaver Big plays so that the final box will have the biggest possible value and Beaver Small plays to get the smallest possible value.

If both always play as well as they possible can, what will the number in the final box be?



B-enigma

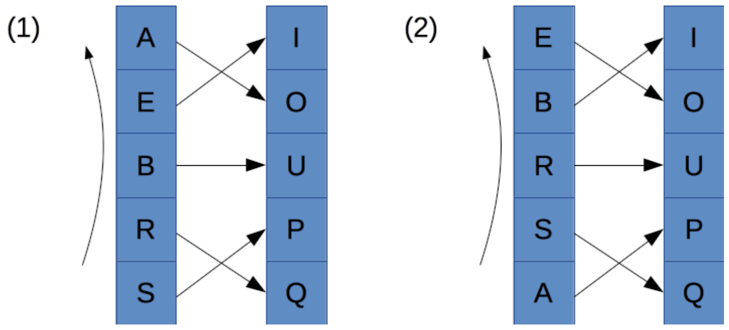
The Beavers need to communicate secretly. They decide to use a mechanism called the B-Enigma machine to hide (encrypt) their messages.



The B-Enigma works as shown above. Each time a letter is typed (e.g. "A"), the left rotor will find a letter on the right rotor according to the arrows (e.g. "O" for "A" in the first step). After typing a letter, the left rotor will move up one position.

This is shown in a different way in the diagram below. After rotating up one position the left rotor will then be in position (2). However, note that the rotor on the right never moves. The links between the two rotors (shown by the straight arrows) also remain the same.

In the diagram below, all the letters available are shown on both rotors.



Question?

The Beavers wish to send the message "BEBRAS".

What will the encrypted message be if we start from position (1)?

A. UOSAEB
B. UOUQOP
C. UOOOIP
D. UOOUPQ

M

RANKING OF BEBRAS PROBLEMS

Highest Qualification Achieved:

Job Title:

Area of work (e.g. if a teacher what subject/s to do you teach):

Do you see any way that these problems could be used within your current work? Please explain.

Problem solving test 1

Please rank the questions from Problem Solving Test 1 from easiest to hardest, with 1 being the easiest question and 13 being the hardest.

Question Name	**Ranking (enter a number between 1-13, use each number only once)**
Animal Competition	
Animation	
Beaver Lunch	
Bowl Factory	
Bracelet	
Cross Country	
Drawing Stars	
Fireworks	
Kangaroo	
Spies	
Stack Computer	
Throw the Dice	
You Won't Find It	

PTO

Please rate the questions from Problem Solving Test 1 on a scale of 1-20 with 1 being easiest and 20 being hardest.

Question Name	**Difficulty (enter a number between 1-20, you may rate questions with the same difficulty)**
Animal Competition	
Animation	
Beaver Lunch	
Bowl Factory	
Bracelet	
Cross Country	
Drawing Stars	
Fireworks	
Kangaroo	
Spies	
Stack Computer	
Throw the Dice	
You Won't Find It	

Problem solving test 2

Please rank the questions from Problem Solving Test 2 from easiest to hardest, with 1 being the easiest question and 13 being the hardest.

Question Name	**Ranking (enter a number between 1-13, use each number only once)**
Bebras Painting	
B-Enigma	
Bottles	
Concurrent Directions	
The Game	
Magic Potion	
Party Guest	
Pirate Hunters	
Scanner Code	
Secret Messages	
Theatre	
Triangles	
Tube System	

Please rate the questions from Problem Solving Test 2 on a scale of 1-20 with 1 being easiest and 20 being hardest.

Question Name	**Difficulty (enter a number between 1-20, you may rate questions with the same difficulty)**
Bebras Painting	
B-Enigma	
Bottles	
Concurrent Directions	
The Game	
Magic Potion	
Party Guest	
Pirate Hunters	
Scanner Code	
Secret Messages	
Theatre	
Triangles	
Tube System	

N

MAYNOOTH UNIVERSITY ETHICAL APPROVAL

For office use only – Application reference number: SRESC-201X-XXXXX

Maynooth University
Social Research Ethics Sub-Committee

Protocol for Tier 2-3 Ethical Review of a Research Project Involving Participation of Humans

Please note the following:

1. The ethics committee will review the protocol and determine eligibility for Expedited Review. If the committee decides that this project is not eligible for expedited review you will be notified and the protocol will automatically be assessed by standard review.
2. Before submitting this application, all researchers named within it should have read and agreed the contents.
3. While attachments may be appended, it is important that you do not simply refer to them, but that you fully address all points in the text of this form. Please keep in mind that your application could be read by someone who is not a specialist in your field, so it is important to make your explanations as clear and thorough as possible.

INSTRUCTIONS: Place your cursor inside the box that follows each question and begin to type – the box will expand as you type. Please submit this completed form, with all supporting documentation, to the Maynooth University Research Support Office Ethics Committee Secretariat. Please include selected review level in the e-mail subject line: **research.ethics@nuim.ie**

1) Tier 2 Expedited Review [ ]

i. Specific criteria for Tier 2 selection
Please select the specific criterion, from the following https://www.maynoothuniversity.ie/research/research-development-office/research-ethics list that entitles the project to be exempt from standard review.

Tier 2, Select from Criteria Number 1 to 4
3

(Please note that the committee will make the final decision regarding eligibility for tier 2 review)

ii. Please give a short justification for selecting Tier 2 review based on the specific criterion selected above.

iii. Tier 3 Standard Review [x]

2. Information about the researcher(s), collaborator(s), and/or supervisor (if the researcher is a postgraduate student)

Please include a letter from the supervisor (*see template at the end of this form*) outlining how the student is suitably prepared/qualified and will have adequate support to carry out the type of research proposed.

Name:	Qualifications or Student No:	Address/Dept.	Email: ***Provide Maynooth University contact details***	Telephone: ***Provide Maynooth University contact details***	Role in the project:
Aidan Mooney	PhD	Dept. of Computer Science	aidan.mooney@nuim.ie	01-7083354	PI and Lead Researcher
James Lockwood	12305466	Dept. of Computer Science	James.Lockwood@nuim.ie	01-7086099	Researcher and Postgraduate Student

3. Previous ethical approval for this project (if applicable)
(Please attach a copy of your approval letter)

Other Ethical Approval	**Reference**
Maynooth University Ethical Approval [ ] Yes [x] No	*SRESC-201x-xxxxx*
Other Institutions [ ] Yes [x] No [ ] Under review *If you are carrying out research in collaboration with another organisation/group you might require ethical approval from this organisation/group as well as Maynooth University. It is the researcher's responsibility to ensure such permissions are in place. Please indicate whether or not such approval is required. Where applicable provide a copy of the application and letter of approval.*	*Name and Address of organisation*

4. Title. Brief title of the research project:

Creation of a Computational Thinking curriculum for Irish schools

5. Research Objectives. Please summarize briefly the objective(s) of the research, including relevant details such as purpose, research question, hypothesis, etc. (about 150 words).

For this project we will create a Computational Thinking (CT) curriculum for Irish schools. This project will focus on the creation of the curriculum through informed reading of the literature and liaising with school teachers around the country. Once an initial pilot curriculum has been designed it will be delivered in a number of schools to test its effectiveness and suitability within the school system.

Collected data will allow us to modify and enhance, where necessary, the curriculum (and any associated resources) to ensure a better teaching and learning environment for all parties involved. Using the information gathered will allow for the creation of a standalone curriculum and allow us to disseminate the findings to other interested parties. This dissemination will be in the form of one or more conference and/or journal publications and will provide guidelines and recommendations for anybody considering adopting CT in to their classes.

This project will consider the following questions:

- What is the current perception of CT amongst teachers?
- How can Computational Thinking be used to improve students interest in STEM subjects and others?
- How have educational institutions implemented Computational Thinking?
- Why is Computational Thinking important for educational institutions to incorporate into their curriculums?

6. Methodology.

a. Where will the research be carried out?

Location(s)	*Please describe the locations where the research will be carried out. If research will be carried out abroad illustrate how you have given due consideration to the ethical norms for the country/culture etc. Note that when working with institutions abroad you might also require ethical approval from that institution/organisation (see Question 4 above).* This research will be carried out in a number of locations. • The initial research, design of the pilot curriculum and design of the main curriculum will take place in the Department of Computer Science, Maynooth University. • The delivery of the pilot curriculum will take place in a cooperating secondary school in Ireland. • The delivery of the main curriculum will take place in a number of cooperating secondary schools in Ireland.
Proposed start date	*31/10/2016*
Approx Duration	*36 months*

b. Please describe briefly the overall methodological design of the project.

The PACT initiative has run for a number of years as a partnership between researchers in Maynooth University's Department of Computer Science and teachers at selected post primary and primary schools around the country. To date we have worked with over 70 teachers across 9 counties. We have run a number of training days with these teachers where Computational Thinking (CT) ideas have been presented to these teachers and the feedback from these teachers and their principals on their introduction to their class has been extremely positive.

This research project will look at building on the work of PACT to deliver a 20 hour CT curriculum for schools. This curriculum will deliver content and skills that will be transferable across multiple subjects within schools. The goal is to develop a curriculum which a teacher can deliver with little or no training in their classroom.

c. Depending on the methods/techniques to be used, please elaborate upon the research context(s), potential questions / issues to be explored, tasks/tests/measures, frequency/duration of sessions, process of analysis to be used, as appropriate.

Relevant details regarding the procedures for data collection should be reflected in the content of the Information Sheet.

For this research we will look in detail at:

- Teacher Perspective of CT prior to the creation of any curriculum (at the commencement of the research)
    - An online survey will be distributed to teachers gathering details on:
        - Their profile – including, email address, age, school level, teaching experience.
        - Their Computer Science exposure to date.
        - Questions relating to CT – including, their understanding of what CT is, their exposure to CT to date, content they feel would be relevant in a CT curriculum, benefits of incorporating CT in to their teaching.

    This survey can be found at: **http://tinyurl.com/jsq7jae**

- Online and face-to-face feedback on developed materials will be sought from champion teachers* to help mould an appropriate curriculum. This will focus on the generated material and their perceived benefits/weaknesses within a CT curriculum.

- Upon creation of the initial curriculum it will be delivered in a number of schools to ascertain its effectiveness. This will involve James Lockwood delivering a 10 hour pilot curriculum at a Transition Year level in a cooperating school.
    - Feedback will be obtained from the students taking this pilot course at the start and at its completion to gauge the effectiveness of the pilot course and also the appropriateness of fit of the content.
    - Feedback will also be requested from the teacher in the school(s) where the pilot is delivered.

  This feedback from students and teachers will help with the creation of the final 20 hour curriculum.

- Following the enhancement of the pilot curriculum based on feedback from the pilot study the 20 hour curriculum will be delivered in a number of schools.
    - Feedback will be obtained from the students taking this course at the start and at the completion to gauge the effectiveness of the pilot and also the appropriateness of fit of the content.
    - Feedback will also be requested from the teacher in the school(s) where the pilot is delivered.

*As part of the PACT initiative a number of teachers have agreed to work closely with all activities and incentives. These teachers are referred to as champion teachers.

7. Participants.

a. Who will the participants be?

Teachers will be contacted initially, from around the country and abroad, to get feedback relating to Computational Thinking. These teachers will be from the contacts within the schools that PACT currently works with, personal contacts in schools and through the teaching training institutions around the country.

School teachers who will facilitate the delivery of the full curriculum and pilot curriculum in schools. These teachers will liaise with their schools to facilitate the delivery and also be present while the curriculum is delivered.

Students in selected schools who will have the pilot curriculum and full curriculum delivered to them.

b. Approximately how many participants do you expect will be involved?

Maximum of 2000

c. How will participants become involved in your project? If you have formal recruitment procedures, or criteria for inclusion/exclusion, please outline them here.

Where gatekeepers are involved in the process of participant recruitment, please clearly outline procedures relating to their involvement.

Participants will become involved through teachers and networks that we have links with. This will allow us access teachers, who will run the pilot project and full project, which in turns will provide us with access to the students.

d. What will be the nature of their participation? (e.g. one-time/short-term contact, longer term involvement, collaborative involvement, etc.)

For the pilot study James Lockwood will deliver a 10 hour curriculum in one school over an 8 week period to a transition year class. These students will take the entire curriculum.

For the main study James Lockwood will deliver a 20 hour curriculum in a number of schools to students over a period of time facilitated by each school and their timetable constraints.

e. If participants will include those with whom the researcher engages in a relationship of power e.g. student/employee/employer/colleague, explain how the possibility of the power relationship and/or conflict of interest will be minimized.

As the content will be delivered in a classroom environment to students, with the cooperating teacher present, there will be a relationship of power present. However, there will be no formal qualification or assessment in place for the curriculum helping to minimise this relationship of power. There will be short online assessments which the students will take but these will not count towards any final assessment marks for the curriculum.

Where students wish to withdraw from the study or do not wish to participate, James Lockwood, in collaboration with the teacher will arrange alternative arrangements for these students in line with the school policy of that school for non-attendance in particular classes/subjects. The arrangement with the pilot school (Drogheda Grammar School) is that the curriculum will be delivered to 25 students – these will be students who agree to take part in the study. The school have said that they will do the selection of these students. The remaining students will be in another stream taking an alternative subject.

f. Will the participants be remunerated, and if so, in what form?

No.

8. Persons Under 18.

a. Will the research be carried out with persons under age 18? [x] Yes [ ] No
Please see section Child Protection Policy (in particular section 5)

b. If yes, will the sessions be supervised by a guardian or a person responsible for the individual(s)?
[x] Yes [ ] No
***NOTE**: Research cannot begin until Garda clearance has been completed. For Maynooth University Students, this is facilitated through* student.vetting@nuim.ie and Staff humanresources@nuim.ie

Please see *The University Policy on Vetting*

9. Vulnerable Persons.

a. Will the research be carried out with persons who might be considered vulnerable in any way?
[x] Yes [ ] No

b. If yes, please describe the nature of the vulnerability and discuss special provisions/safeguards to be made for working with these persons.

The students who will take the pilot curriculum and the full curriculum will be under 18 years of age and thus deemed vulnerable.

The school teacher, whom James Lockwood will be taking the class time from, will be present for all classes to be delivered in the schools.

Additionally, James Lockwood has Garda clearance from working in Coder Dojo. He will apply for additional clearance to work on this project through the Maynooth University Admissions Office.

***NOTE:** Depending on the nature of the vulnerability, sessions may need to be supervised or the researcher may need to undergo Garda vetting as stated above under point 8. In such cases, the researcher must also be prepared to demonstrate how s/he is suitably qualified or trained to work with such persons.*

10. Risks.

a. Please describe any possible risks or conflicts arising from the research techniques or procedures such as: power relationships or other conflict of interests i.e. supervisor-student relationship, physical stress/reactions or psychological emotional distress or reactions.

Please consider any potential risks that may arise from the publication of results of this research. It is important to note that power relationships may exist in situations other than supervisor-student relationships.

It is envisaged that there will be no additional risks for the student as they will be taking classes in their normal school setting.

b. If you anticipate the possibility of risks, how will these potential risks be addressed and what measures have you put in place to minimize them?

Please consider that issues may arise for participants following research participation. Where appropriate, contact details for relevant sources of information and/or support should be included in the participant's Information Sheet.

11. Informed Consent.

Please answer the following questions about how you inform participants about your research and then obtain their consent:
***NOTE:** Please see the template at the end of this form showing standard information that must be included on all consent forms.*

a. Do research participants sign a written consent form and receive a copy for their records? If not, do they receive an information sheet that provides what they need to know before deciding to participate?

Yes

b. When, where, and by whom is consent obtained?

Consent will be obtained prior to the delivery of any of the curriculums within the schools from the cooperating teachers.

Consent will be obtained from each student's parents/guardians to study the curriculum. These forms will be paper based and will be distributed to each cooperating teacher in advance of the commencement of the curriculum. These forms will be all signed and returned to the teacher prior to the commencement of the curriculum. A copy of these forms will be returned to the parents/guardians where they wish to have one.

If students wish to withdraw or not participate in the study arrangements will be put in place, in line with school policy, around students not attending particular classes/subjects. The arrangement with the pilot school (Drogheda Grammar School) is that the curriculum will be delivered to 25 students – these will be students who agree to take part in the study. The school have said that they will do the selection of these students. The remaining students will be in another stream taking an alternative subject.

c. If children or vulnerable persons are involved, please explain your procedure for obtaining their assent.

Assent forms for each student will be gathered and a copy returned to each student if they wish. These forms will be distributed to the cooperating teachers prior to the commencement of the curriculums and will all be returned in advance of starting. An accompanying information sheet will also be distributed to each student.

d. For projects in which participants will be involved over the long term, how will you ensure that participants have an ongoing opportunity to negotiate the terms of their consent?

Please bear in mind that in order to negotiate further consent identifiers will have to be collected with the *data. If data is completely anonymous please ensure that consent is sought from participants at the outset to maintain and re-use their data.*

e. What will the participants be told about the study?

The participants will be told that the study is looking at the creation of a curriculum for Computational Thinking. They will be informed that through their feedback and comments the curriculum will be tailored to create a goodness of fit curriculum for Irish schools.

f. What information, if any, will be withheld about the research procedure or the purposes of the investigation? Please explain your justification for withholding this information. If any deception will be involved, please be sure that the technique is explained above under methodology, and explain here why the deception is justified.

No information will be withheld from the participants.

12. Follow-up. As appropriate, please explain what strategies you have in place to debrief or follow up with participants.

Upon completion of the pilot curriculum James Lockwood will revisit the cooperating school and relate to the students the findings of the study. School management and all teachers will also be invited.

Upon completion of the main curriculum study James Lockwood will revisit the cooperating schools and relate to the students the findings of the study and future direction of the research. School management and all teachers will also be invited.

13. Confidentiality/Anonymity of Data.
Please consult Maynooth University data protection procedures: http://dataprotection.nuim.ie/protection_procedures.shtml

a. Recording of personally identifiable information about research participants

Identifier *(Typically, by their very nature projects involving repeated contact with research participants require the collection and retention of identifiers)*	**Y/ N** (Select all those applicable)
Name and Contact Details	Y
Details regarding Geographical location, culture, ethnicity etc.	Y
Video recording	N
Audio recording	Y
Other please specify	
Not applicable	

b. If yes, to any of the above please explain how confidentiality and/or anonymity are assured?

Please ensure that participants are informed of the limits to confidentiality as outlined in section 3.3 of the ethics policy (http://research.nuim.ie/system/files/images/Ethics%20Policy%20Approved%20by%20AC%2012%2002%2012.pdf)

The following or similar text may be used in consent/information sheet.

'It must be recognized that, in some circumstances, confidentiality of research data and records may be overridden by courts in the event of litigation or in the course of investigation by lawful authority. In such circumstances the University will take all reasonable steps within law to ensure that confidentiality is maintained to the greatest possible extent.'

When the participants submit any feedback the data they enter is stored securely within the Computer Science department. This will include their personal details along with the school they teach at or study at. In the surveys names will not be recorded, just email address. A unique referencing system will be used to map the teachers email to a unique identifier. This unique identifier will be stored with the data collected.

When obtaining feedback from the champion teachers face-to-face interviews will take place. These interviews will be recorded and any interesting/valuable feedback raised will be used in future publications.

Feedback surveys from students will not ask for nor store any personal details. The details stored will be their school and questions relating to the curriculum that had been delivered to them.

c. If yes, to any of the above please explain the following: where will data be stored; how you will safeguard this information; if identifiers will be removed from the data, at what point will they be removed; if identifiers will not be removed, why they must be retained and who will retain the key to re-identify the data.

Please note: primary data should be anonymised and retained for a period of ten years from publication as outlined in the University's Research Integrity Policy.

Data storage
If identifiers are collected please show how they will be stored separately to the data and deleted as soon as is feasible. Please state who will have access to the identifiers and/or the data.
Please indicate that data will be encrypted, devices password protected and stored on campus either on a desktop computer or secure server
Please note that data must be removed from mobile devices as soon as possible following collection.

Data retention
If identifiers are being used please outline how long you intend to retain the data in an identifiable manner and for what purpose. Please note that identifiable data cannot be retained indefinitely.

Secondary use and processing
If you intend to retain data for future secondary use and analyses (i.e. to the use information originally collected for a purpose other than the current research purpose) consent should be sought for such retention and processing.

For identifiable data consent must be sought for each specific subsequent use.

All submitted data will be stored securely within the Computer Science department.

Identifiers will be retained until completion of the delivery of the curriculum to the schools.

Data, collected during this study, will be retained for a period of ten years from publication as outlined in the University's Research Integrity Policy. This will allow for future reassessment or verification of the data from primary sources, if necessary.

NOTE*: Include this information in the consent form, information sheet, or consent script.*

d. Please explain how, when, and by whom data will be destroyed once the period of data retention has expired

Electronic data should be overwritten
Paper data should be destroyed by confidential shredding

Electronic data will be destroyed by removing the current cohort of participant details from the database storing the information on the secure server located in the Computer science department. This data will be removed by the researchers in conjunction with a technician in the department of Computer Science.

Assertion forms and consent forms will be confidentially shredded within the department of Computer Science.

14. Ethics in subsequent outputs. What are your plans for protecting the safety and integrity of research participants in publications, public presentations, or other outputs resulting from this research? How will subjects' permission for further use of their data be obtained?

Permission to use feedback from electronic surveys will be requested within each survey in the form of a tick box. In the assert forms and consent forms the person signing will be asked if they agree to their feedback being used in publication.
No personal details of any participant will be portrayed in publication. High level will be used in relation to student data information (for example, age, gender, etc.) and equally for teacher data (for example, teaching experience, subjects being taught etc.)

***NOTE**: If the data is not anonymised, additional consent would have to be obtained before the data could be deposited in an archive such as the Irish Qualitative Data Archive (http://www.iqda.ie/) or the Irish Social Science Data Archive (http://issda.ucd.ie/).*

15. Professional Codes of Ethics. Please append a professional code of ethics governing research in your area to this protocol, and/or provide a link to the website where the code may be found.

https://www.maynoothuniversity.ie/research/research-development-office/policies

16. Declaration

This declaration must be signed by the applicant(s) ***(electronic signature is sufficient)***.

I(we) the undersigned researcher(s) acknowledge(s) and agree that:

a) It is my (our) sole responsibility and obligation to comply with all Irish and EU legislation relevant to this project.
b) That all personnel working on this project comply with Irish and EU legislation relevant to this project.
c) That the research will be conducted in accordance with the Maynooth University Research Ethics Policy.
d) That the research will be conducted in accordance with the Maynooth University Research Integrity Policy.
e) That the research will not commence until ethical approval has been granted.

Signature of Applicant(s): [illegible] [illegible]

Date: 02/11/2016

<u>Check List</u>

1)	Completed application form	[x]
2)	Letter from supervisor if applicant is a student	[ ]

<u>if applicable – copies of:</u>

3)	prior ethical approval	[ ]
4)	ethical approval from other institutions	[ ]
5)	proposed information sheet	[x]
6)	proposed consent form	[x]
7)	Documentary evidence for the use of existing data records, sourced from third party organisations, that consent was originally sought for data to be used for research purposes.	[ ]

TEMPLATE FOR SUPERVISOR'S LETTER

The supervisor's letter should outline how the student is suitably prepared to carry out the type of research proposed. The following points should be addressed:

- Please elaborate on the student's preparedness to undertake the proposed research.
- Please provide details of the student's methodological competence to undertake the research project.
- Please address your confidence in the student's ability to manage risks that may arise as part of the research project.
- Please describe the support that you and the department will give to the student in the management and the execution of the research project.

TEMPLATE FOR INFORMATION SHEET

The form and content of information sheets and consent forms varies according to the nature of each project; however, the following standard information must be included on all forms used in projects affiliated with Maynooth University:

- The form should be written in simple language and should vary depending on whether the participants are children and/or adults.
- If participants are not native English speakers then the information sheet should be translated appropriately.
- The information sheet should contain details of the project, what is required of participants and what will happen to the data generated from it.
- It should also include details about how the data will be safeguarded, for what purposes it may be used, for how long it will be kept and how and when it will be destroyed.
- The form should outline the limits to confidentiality. The following or similar text may be used:
  'It must be recognized that, in some circumstances, confidentiality of research data and records may be overridden by courts in the event of litigation or in the course of investigation by lawful authority. In such circumstances the University will take all reasonable steps within law to ensure that confidentiality is maintained to the greatest possible extent.'
- Participants should be given the name and contact details of a qualified person (e.g. doctor/nurse/counsellor) should they experience any stress or reactions following participation.

TEMPLATE FOR CONSENT FORM

- Researcher(s) name, address and contact number (provide Maynooth University details only, no personal details or phone numbers should be supplied).
- Supervisor(s) name, address and contact number (if applicable).
- If the research is not anonymous participants should be told that they can withdraw their consent at any time up until the work is published as well the right to access their data at their discretion.
- The following statement should be included on all consent forms (verbatim):

  If during your participation in this study you feel the information and guidelines that you were given have been neglected or disregarded in any way, or if you are unhappy about the process, please contact the Secretary of the Maynooth University Ethics Committee at research.ethics@nuim.ie or +353 (0)1 708 6019. Please be assured that your concerns will be dealt with in a sensitive manner.

O

CURRICULUM LINKS TO THE JUNIOR CERTIFICATE CODING SHORT COURSE AND THE LEAVING CERTIFICATE COURSE

Curriculum Links to the Leaving Certificate Computer Science Specification

Strand 1: Computational thinking

1.1 describe a systematic process for solving problems and making decisions

- Algorithms
- Binary & Data
- Cryptography
- Introduction to Computer Science
- Introduction to Computational Thinking
- Python
- Scratch
- Searching & Sorting

1.2 explain how the power of computing enables different solutions to difficult problems

- Cryptography
- Databases
- Finite State Machines
- Introduction to Computer Science
- Python
- Scratch

1.3 solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion

- Algorithms
- Cryptography
- Introduction to Computer Science
- Introduction to Computational Thinking
- Programming Concepts
- Python
- Scratch
- Searching and Sorting
- Web Development

1.4 solve problems using skills of logic

- Algorithms
- Cryptography
- Introduction to Computer Science
- Introduction to Computational Thinking
- Programming Concepts
- Python
- Scratch
- Searching and Sorting

1.5 evaluate alternative solutions to computational problems

- Algorithms
- Cryptography
- Introduction to Computer Science
- Programming Concepts
- Python
- Scratch
- Searching and Sorting

1.6 explain the operation of a variety of algorithms

- Algorithms
- Cryptography
- Programming Concepts
- Python
- Scratch
- Searching and Sorting

1.7 develop algorithms to implement chosen solutions

- Algorithms
- Cryptography
- Introduction to Computer Science
- Python
- Scratch
- Searching and Sorting

1.8 evaluate the costs and benefits of the use of computing technology in automating processes

- Cryptography
- Introduction to Computer Science

1.9 use modelling and simulation in relevant situations

- Algorithms
- Programming Concepts
- Python
- Scratch

1.10 discuss when heuristics should and could be used and explain the limitations of using heuristics

Strand 1: Computers and society

1.11 discuss the complex relationship between computing technologies and society including issues of ethics

1.12 compare the positive and negative impacts of computing on culture and society

- Introduction to Computer Science

1.13 identify important computing developments that have taken place in the last 100 years and consider emerging trends that could shape future computing technologies

1.14 explain when and what machine learning and AI algorithms might be used in certain contexts

1.15 consider the quality of the user experience when interacting with computers and list the principles of universal design, including the role of a user interface and the factors that contribute to its usability

1.16 compare two different user interfaces and identify different design decisions that shape the user experience

1.17 describe the role that adaptive technology can play in the lives of people with special needs

1.18 recognise the diverse roles and careers that use computing technologies

- Introduction to Computer Science

Strand 1: Designing and developing

1.19 identify features of both staged and iterative design and development processes

- Python
- Scratch

1.20 collaborate and assign roles and responsibilities within a team to tackle a computing task

- Algorithms
- Cryptography
- Databases
- Introduction to Computer Science
- Programming Concepts
- Searching and Sorting

1.21 identify alternative perspectives, considering different disciplines, stakeholders and end users

- Algorithms
- Cryptography
- Python
- Scratch

1.22 read, write, test, and modify computer programs

- Python
- Scratch

1.23 reflect and communicate on the design and development process

- Python
- Scratch

Strand 2: Abstraction

2.1 use abstraction to describe systems and to explain the relationship between wholes and parts

2.2 use a range of methods for identifying patterns and abstract common features

- Algorithms
- Cryptography
- Finite State Machines
- Searching and Sorting

2.3 implement modular design to develop hardware or software modules that perform a specific function

2.4 illustrate examples of abstract models

Strand 2: Algorithms

2.5 use pseudo code to outline the functionality of an algorithm

- Algorithms
- Programming concepts
- Python
- Scratch

2.6 construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement

- Algorithms
- Programming Concepts
- Python
- Scratch

2.7 implement algorithms using a programming language to solve a range of problems

- Python
- Scratch

2.8 apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm

- Scratch
- Searching and Sorting

2.9 assemble existing algorithms or create new ones that use functions (including recursive), procedures, and modules

- Scratch
- Searching and Sorting

2.10 explain the common measures of algorithmic efficiency using any algorithms studied

- Searching and Sorting

Strand 2: Computer systems

2.11 describe the different components within a computer and the function of those components

2.12 describe the different types of logic gates and explain how they can be arranged into larger units to perform more complex tasks

2.13 describe the rationale for using the binary number system in digital computing and how to convert between binary, hexadecimal and decimal

- Binary and Data

2.14 describe the difference between digital and analogue input

2.15 explain what is meant by the World Wide Web (WWW) and the Internet, including the client server model, hardware components and communication protocols

- Web Development

Strand 2: Data

2.16 use data types that are common to procedural high-level languages

- Python
- Scratch

2.17 use ASCII and Unicode character sets to encode/decode a message and consider the importance of having such standards

2.18 collect, store and sort both continuous and discrete data

Stand 2: Evaluation and testing

2.19 test solutions and decisions to determine their short-term and long-term outcomes

- Algorithms
- Python
- Scratch

2.20 identify and fix/debug warnings and errors in computer code and modify as required

- Scratch
- Python

2.21 critically reflect on and identify limitations in completed code and suggest possible improvements

- Python
- Scratch

2.22 explain the different stages in software testing

Applied Learning Task 1: Interactive Information Systems

3.1 understand and list user needs/requirements before defining a solution

- Python
- Scratch
- Algorithms

3.2 create a basic relational database to store and retrieve a variety of forms of data types

- Databases

3.3 use appropriate programming languages to develop an interactive website that can display information from a database that meets a set of users' needs

Applied Learning Task 2: Analytics

3.4 develop algorithms that can find the frequency, mean, median and mode of a data set

- Python

3.5 structure and transform raw data to prepare it for analysis

3.6 represent data to effectively communicate in a graphical form

3.7 use algorithms to analyse and interpret data in a way that informs decision-making

- Binary and Data
- Databases
- Searching and Sorting

Applied Learning Task 3: Modelling and Simulation

3.8 develop a model that will allow different scenarios to be tested

- Python
- Scratch

3.9 analyse and interpret the outcome of simulations both before and after modifications have been made

3.10 explain the benefits of using agent-based modelling and how it can be used to demonstrate emergent behaviours

Applied Learning Task 4: Embedded Systems

3.11 use and control digital inputs and outputs within an embedded system

3.12 measure and store data returned from an analogue input

3.13 develop a program that utilises digital and analogue inputs

3.14 design automated applications using embedded systems

Curriculum links to Junior Certificate Coding Short Course

Stand 1: Computer Science Introduction

1.1 present and share examples of what computers are used for and discuss their importance in modern society and in their lives
- Introduction to Computer Science

1.2 describe the main components of a computer system (CPU, memory, main storage, I/O devices, buses)
- Programming concepts

1.3 explain how computers are devices for executing programs via the use of programming languages
- Introduction to Computer Science
- Programming concepts
- Python
- Scratch

1.4 develop appropriate algorithms using pseudo-code and/or flow charts
- Algorithms
- Cryptography
- Introduction to Computer Science
- Python
- Scratch
- Searching and Sorting

1.5 write code to implement algorithms
- Python
- Scratch

1.6 discuss and implement core features of structured programming languages, such as variables, operators, loops, decisions, assignment and modules
- Programming concepts
- Python
- Scratch

1.7 test the code
- Python
- Scratch

1.8 evaluate the results in groups of two or three
- Python
- Scratch

Strand 2: Let's get connected

2.1 discuss the basic concepts underlying the internet

2.2 describe how data is transported on the internet and how computers communicate and cooperate through protocols

2.3 explain how search engines deliver results

2.4 build a website using HTML and CSS to showcase their learning

- Web Development

2.5 explain how computers represent data using 1's and 0's

- Binary and Data

2.6 investigate how drawings and photos are represented in computing devices

2.7 identify a topic or a challenge in computer science that inspires them

- Introduction to Computer Science

2.8 conduct research on the topic/challenge

2.9 present a proposal for discussion and reflect on feedback

2.10 convince their peers that an idea is worthwhile

Strand 3: Coding at the next level

3.1 creatively design and write code for short programming tasks to demonstrate the use of operators for assignment, arithmetic, comparison, and Boolean combinations

- Python
- Scratch

3.2 complete short programming tasks using basic linear data structures (e.g. array or list)

3.3 demonstrate how functions and/or procedures (definition and call) capture abstractions

- Python

3.4 describe program flow control, e.g. parallel or sequential flow of control – language dependent

- Programming concepts
- Python
- Scratch

3.5 document programs to explain how they work

3.6 present the documented code to each other in small groups

3.7 analyse code to determine its function and identify errors or potential errors

- Python
- Scratch

P

INFORMATION SHEET

Information Sheet

Research Project:	Developing a Computational Thinking curriculum for secondary schools.
Aim:	Develop, teach and analyse a Computational Thinking curriculum for Transition Year students
Researcher:	Mr James Lockwood, Department of Computer Science, Maynooth University, Maynooth, Co. Kildare Dr Aidan Mooney, Department of Computer Science, Maynooth University, Maynooth, Co. Kildare
Contact details:	James Lockwood, james.lockwood@nuim.ie> Aidan Mooney, amooney@cs.nuim.ie

The Purpose:

The purpose of the research study is to investigate the impact and effectiveness of a Computational Thinking course which will be designed for and taught to Transition Year students.

The Participant:

The participant mentioned throughout this information sheet, refers to a Transition Year student currently enrolled in a participating school.

The Tasks:

My child's participation will consist in taking part in the course which will involve practical exercises, various assignments as well as filling out feedback forms and information sheets for the various metrics we are investigating.

The Procedure

The course will be taught over a period agreed with by the participating school and in conjunction with the relevant departments in the school.

The participant will then be given an informed consent form and information sheet. The participant will be then reassured that all gathered data will be immediately encoded so that participants will only be identified by a reference number and no identifying information will be stored. A participant ideally should be native English speakers or have a sufficient mastery of English to study in Ireland (as measured by a recognised English examination e.g. score of 6.0 UG or 6.5 PG IELTS examination).

Once consent has been given, the participant will be asked to complete a series of background surveys which collect data on a number of metrics which will be used to analyse the impact of the course. These metrics may include but not be limited to: personal information (ie. gender, age etc.), mathematical ability, interest in computers, self-efficacy/self-esteem, anxiety, interaction with technology, college/further education plans, parental occupation etc.

Non-participation and exit from the study:

There is no requirement to participate in this study. A participant may choose to exit the study at any time up until the work is published. The participant must send their request in writing to the researcher and all data collected to date for that participant will be destroyed.

Anonymity and security of data:

As soon as the data is collected it will be encoded with a unique identity key. No records of the participant's identity will be stored. It must be recognized that, in some circumstances, confidentiality of research data and records may be overridden by courts in the event of litigation or during investigation by lawful authority. In such circumstances the University will take all reasonable steps within law to ensure that confidentiality is maintained to the greatest possible extent.

The data will be stored on a secure server located in Ireland. Only the above-named researcher will have access to it. The data will be kept for the duration of the research and will be destroyed thereafter.

Access to your data:

At any time, a participant can request a copy of their data that is stored by contacting the researcher. The participant will need to provide their unique key so that the researcher can identify their data.

Questions:

If you have any further questions, please contact the researchers using the above contact details.

Note on Computational Thinking

Computational Thinking is about combining the creativity of human thinking with the power of computing machines to solve problems across a range of disciplines. It is a core skill which is crucial to many 21st century careers, not just in information technology, but also physics, finance, engineering and bioinformatics. The focus of the module is not on learning facts about computers but on developing creative ideas and new ways of thinking.

Q CONSENT FORM

Consent Form

Research Project:	Developing a Computational Thinking curriculum for secondary schools.
Aim:	Develop, teach and analyse a Computational Thinking curriculum for Transition Year students
Researcher:	Mr James Lockwood, Department of Computer Science, Maynooth University, Maynooth, Co. Kildare Dr Aidan Mooney, Department of Computer Science, Maynooth University, Maynooth, Co. Kildare
Contact details:	James Lockwood, james.lockwood@nuim.ie Aidan Mooney, amooney@cs.nuim.ie

I, ______________________________ give permission for my child _________________________________ to participate in the research project being carried out by Mr James Lockwood and Dr Aidan Mooney to investigate the impact and effectiveness of a Computational Thinking course for Transition Year students.

My child's participation will consist in taking part in the course which will involve practical exercises, various assignments as well as filling out feedback forms and information sheets for the various metrics we are investigating. These metrics may include but not be limited to: gender, age, mathematical ability, interest in computers, self-efficacy/self-esteem, anxiety, interaction with technology, college/further education plans, parental occupation etc.

The data gathered will only be used by the researchers above and the findings may be published in suitable conferences and journals. I can access their data at my discretion. I understand that the research will not form any kind of medical diagnosis or treatment.

I have received assurance from the researchers that the information that my child and I will share will remain strictly confidential and that no information that discloses my identity will be released or published. However, I recognize that, in some circumstances, confidentiality of research data and records may be overridden by courts in the event of litigation or during investigation by lawful authority. In such circumstances the University will take all reasonable steps within law to ensure that confidentiality is maintained to the greatest possible extent

I am free to withdraw my child from the study at any time until the work is published. My child can refuse to answer any of the questions asked or to participate in any of the tasks. All data gathered will be stored in a secure manner and only the above-named researchers will have access to it. There are two copies of the consent form, one of which I may keep. If I have any questions about the research project, I may contact the named researchers above at the contact details provided.

If during your participation in this study you feel the information and guidelines that you were given have been neglected or disregarded in any way, or if you are unhappy about the process, please contact the Secretary of the Maynooth University Ethics Committee at research.ethics@nuim.ie or +353 (0)1 708 6019. Please be assured that your concerns will be dealt with in a sensitive manner.

Name:	______________________________	
Signed:	______________________________	Date:______________________________
Researchers Signature:	_____________________________	Date:_____________________________

BIBLIOGRAPHY

[1] S. I. Ahamed, D. Brylow, R. Ge, P. Madiraju, S. J. Merrill, C. A. Struble, and J. P. Early. "Computational thinking for the sciences: a three day workshop for high school science teachers". In: *Proceedings of the 41st ACM technical symposium on Computer science education*. 2010, pp. 42–46.

[2] Neil Anderson, Colin Lankshear, Carolyn Timms, and Lyn Courtney. "'Because it's boring, irrelevant and I don't like computers': Why high school girls avoid professionally-oriented ICT subjects". In: *Computers & Education* 50.4 (2008), pp. 1304–1318.

[3] S. Atmatzidou and S. Demetriadis. "Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences". In: *Robotics and Autonomous Systems* 75 (2016), pp. 661–670.

[4] I. Z. Bargury, O. Muller, B. Haberman, D. Zohar, A. Cohen, D. Levy, and R. Hotoveli. "Implementing a new computer science curriculum for middle school in Israel". In: *Proceedings Frontiers in Education Conference (FIE)* (2012), pp. 1–6.

[5] A. R. Basawapatna, A. Repenning, K. H. Koh, and M. Savignano. "The Consume-create spectrum: Balancing convenience and computational thinking in stem learning". In: *Proceedings of the 45th ACM technical symposium on Computer science education*. 2014, pp. 659–664.

[6] S. Basu, J. S. Kinnebrew, and G. Biswas. "Assessing student performance in a computational-thinking based science learning environment". In: *International Conference on Intelligent Tutoring Systems*. Springer International Publishing, 2014, pp. 476–481.

[7] T. Bell, C. Duncan, and J. Atlas. "Teacher Feedback on Delivering Computational Thinking in Primary School". In: *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. 2016.

[8] Marina Umaschi Bers. *Coding as a playground: Programming and computational thinking in the early childhood classroom*. Routledge, 2017.

[9] Sylvia Beyer, Kristina Rynes, Julie Perrault, Kelly Hay, and Susan Haller. "Gender differences in computer science students". In: *ACM SIGCSE Bulletin* 35.1 (2003), pp. 49–53.

[10] L. Blum and T. J. Cortina. "CS4HS: an outreach program for high school CS teachers." In: *ACM SIGCSE Bulletin* 39.1 (2007), pp. 19–23.

[11] A. Brancaccio, M. Marchisio, C. Palumbo, C. Pardini, A. Patrucco, and R. Zich. "Problem Posing and Solving: Strategic Italian Key Action to Enhance Teaching and Learning Mathematics and Informatics in the High School". In: *Proccedings Computer Software and Applications Conference (COMPSAC)*. 2015, pp. 845–850.

[12] Jim Broadbent and WL Poon. "Self-regulated learning strategies & academic achievement in online higher education learning environments: A systematic review". In: *The Internet and Higher Education* 27 (2015), pp. 1–13.

[13] Francisco Buitrago Flórez, Rubby Casallas, Marcela Hernández, Alejandro Reyes, Silvia Restrepo, and Giovanna Danies. "Changing a generation's way of thinking: Teaching computational thinking through programming". In: *Review of Educational Research* 87.4 (2017), pp. 834–860.

[14] A. Bundy. "Computational thinking is pervasive." In: *Journal of Scientific and Practical Computing* 1(2) (2007), pp. 67–69.

[15] T. Carvalho, D. Andrade, J. Silveira, V. Auler, S. Cavalheiro, M. Aguiar, L. Foss, A. Pernas, and R. Reiser. "Discussing the challenges related to deployment of computational thinking in brazilian basic education". In: *Proceedings of 2nd Workshop-School on Theoretical Computer Science (WEIT)*. 2013, pp. 111–115.

[16] M. E. Caspersen and P. Nowack. "Computational thinking and practice: A generic approach to computing in Danish high schools". In: *Proceedings of the 15hth Australasian Computing Education Conference*. 2013, pp. 137–143.

[17] Guanhua Chen, Ji Shen, Lauren Barth-Cohen, Shiyan Jiang, Xiaoting Huang, and Moataz Eltoukhy. “Assessing elementary students’ computational thinking in everyday reasoning and robotics programming”. In: *Computers & Education* 109 (2017), pp. 162–175.

[18] V. Chiprianov and L. Gallon. “Introducing Computational Thinking to K-5 in a French Context”. In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 2016, pp. 112–117.

[19] S. S. Cho, V. P. Pauca, D. Johnson, and Y. V. James. “Computational thinking for the rest of us: A liberal arts approach to engaging middle and high school teachers with computer science students”. In: *Proceedings of the Society for Information Technology & Teacher Education International Conference*. 2014, pp. 79–86.

[20] T. J. Cortina, W. P. Dann, C. Frieze, C. Ciminillo, C. Tananis, and K. Trahan. “Work in progress: ACTIVATE: Advancing computing and technology interest and innovation through teacher education.” In: *Proceedings Frontiers in Education Conference (FIE)*. 2012, pp. 1–2.

[21] National Council for Curriculum and Assessment. “Coding”. In: (2017). [accessed August 8, 2018].

[22] P. Curzon. “cs4fn and computational thinking unplugged”. In: *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*. 2013, pp. 47–50.

[23] P. Curzon, P. W. McOwan, N. Plant, and L. R. Meagher. “Introducing teachers to computational thinking using unplugged storytelling”. In: *Proceedings of the 9th Workshop in Primary and Secondary Computing Education*. 2014, pp. 89–92.

[24] V. Dagiene and G. Stupuriene. “Informatics in Education”. In: *Bebras-a sustainable community building model for the concept based learning of informatics and computational thinking.* 2016.

[25] S. Davies. "The effects of emphasizing computational thinking in an introductory programming course". In: *Proceedings Frontiers in Education Conference (FIE)*. 2008.

[26] Peter J Denning. "The profession of IT Beyond computational thinking". In: *Communications of the ACM* 52.6 (2009), pp. 28–30.

[27] Peter J Denning. "Remaining trouble spots with computational thinking". In: *Communications of the ACM* 60.6 (2017), pp. 33–39.

[28] K. Falkner, R. Vivian, and N. Falkner. "Teaching Computational Thinking in K-6: The CSER Digital Technologies MOOC." In: *Proceedings of the 17th Australasian Computing Education Conference (ACE)*. (Vol. 27). 2015, p. 30.

[29] R. Folk, G. Lee, A. Michalenko, A. Peel, and E. Pontelli. "GK-12 DISSECT: Incorporating computational thinking with K-12 science without computer access". In: *Proccedings Frontiers in Education Conference (FIE)*. 2015.

[30] I. Fronza, N. El Ioini, and L. Corral. "Students want to create apps: leveraging computational thinking to teach mobile software development." In: *Proceedings of the 16th Annual Conference on Information Technology Education*. 2015, pp. 21–26.

[31] Francisco José García-Peñalvo. "Computational Thinking". In: (2018).

[32] M. R. González. "EDULEARN15". In: *Computational thinking test: Design guidelines and content validation.* 2015.

[33] L. Gouws, K. Bradshaw, and P. Wentworth. "October". In: *First year student performance in a test for computational thinking*. In Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference . ACM, 2013, pp. 271–277.

[34] S. Grover, S. Cooper, and R. Pea. "Assessing computational learning in K-12". In: *Proceedings of Conference on Innovation & technology in computer science education*. 2014, pp. 57–62.

[35] S. Grover and R. Pea. "Using a discourse-intensive pedagogy and android's app inventor for introducing computational concepts to middle school students". In: *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, pp. 723–728.

[36] S. Grover, R. Pea, and S. Cooper. "Designing for deeper learning in a blended computer science course for middle school students". In: *Computer Science Education* 25.2 (2015), pp. 199–237.

[37] S. Grover, R. Pea, and S. Cooper. "Factors influencing computer science learning in middle school". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 2016, pp. 552–557.

[38] Yasemin Gülbahar and Filiz Kalelioğlu. "Competencies of High School Teachers and Training Needs for Computer Science Education". In: *Proceedings of the 6th Computer Science Education Research Conference*. ACM. 2017, pp. 26–31.

[39] R. J. Haddad and Y. Kalaani. "Can computational thinking predict academic performance?" In: *Proceedings Integrated STEM Education Conference (ISEC)*. 2015, pp. 225–229.

[40] Ting-Chia Hsu, Shao-Chen Chang, and Yu-Ting Hung. "How to learn and how to teach computational thinking: Suggestions based on a review of the literature". In: *Computers & Education* 126 (2018), pp. 296–310.

[41] Aleata Hubbard. "Pedagogical content knowledge in computing education: a review of the research literature". In: *Computer Science Education* (2018), pp. 1–19.

[42] P. Hubwieser and A. Mühling. "9th Workshop in Primary and Secondary Computing Education (WiPSCE)". In: *Playing PISA with bebras.* 2014.

[43] P. Hubwieser and A. Mühling. "Learning and Teaching in Computing and Engineering (LaTiCE)". In: *Investigating the psychometric structure of Bebras contest: towards mesuring computational thinking skills.* 2015.

[44] S. P. Imberman, D. Sturm, and M. Q. Azhar. “Computational thinking: expanding the toolkit”. In: *Journal of Computing Sciences in Colleges* 29.6 (2014), pp. 39–46.

[45] Jambav Inc. *ToonDoo.com*. `http://www.toondoo.com`. [accessed June 23, 2017]. 2012.

[46] J. T. Jenkins, J. A. Jerkins, and C. L. Stenger. “A plan for immediate immersion of computational thinking into the high school math classroom through a partnership with the alabama math, science and technology initiative”. In: *Proceedings of the 50th Annual Southeast Regional Conference*. 2012, pp. 148–152.

[47] F. Kalelioglu, Y. Gülbahar, and V. Kukul. “A Framework for Computational Thinking Based on a Systematic Research Review”. In: *Baltic Journal of Modern Computing* 4.3 (2016), p.583.

[48] B. Kitchenham and S. Charters. “Guidelines for performing Systematic Literature reviews in Software Engineering Version 2.3”. In: *Engineering* 45(4ve) (2007).

[49] K. H. Koh, A. Repenning, H. Nickerson, Y. Endo, and P. Motter. “Will it stick?: exploring the sustainability of computational thinking education through game design”. In: *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, pp. 597–602.

[50] K. H. Koh, A. Basawapatna, H. Nickerson, and A. Repenning. “Real time assessment of computational thinking”. In: *Proceedings Visual Languages and Human-Centric Computing (VL/HCC)*. 2014.

[51] Özgen Korkmaz, Recep Çakir, and M Yaşar Özden. “A validity and reliability study of the Computational Thinking Scales (CTS)”. In: *Computers in Human Behavior* 72 (2017), pp. 558–569.

[52] J. L’Heureux, D. Boisvert, R. Cohen, and K. Sanghera. “IT problem solving: an implementation of computational thinking in information technology”. In: *Proceedings of the 13th annual conference on Information technology education*. 2012, pp. 183–188.

[53] I. Lee, F. Martin, J. Denner, B. Coulter, W. Allan, J. Erickson, J. Malyn-Smith, and L. Werner. "Computational thinking for youth in practice". In: *ACM Inroads* 2.1 (2011), pp. 32–37.

[54] W. L. Li, C. F. Hu, and C. C. Wu. "Teaching High School Students Computational Thinking with Hands-on Activities". In: *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. 2016, pp. 371–371.

[55] A. Lishinski, A. Yadav, R. Enbody, and J. Good. "The Influence of Problem Solving Abilities on Students' Performance on Different Assessment Tasks in CS1". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. 2016, pp. 329–334.

[56] J. Lockwood and A. Mooney. "Computer Science 2 Go: Teaching Computer Science Through Unplugged Puzzles And Games". In: *8th Irish Conference on Game-Based Learning IGBL 2018 Cork City Ireland 28th and 29th June 2018*.

[57] J. Lockwood and A. Mooney. "Teaching computer science through unplugged puzzles and games". In: *Computers in Education Society of Ireland Conference 2018*.

[58] J. Lockwood and A. Mooney. "A Pilot Study Investigating The Introduction Of A Computer-Science Course at Second Level Focusing on Computational Thinking". In: *Irish Journal of Education* (2017), pp. 108–127.

[59] J. Lockwood and A. Mooney. "Computational Thinking in Secondary Education: Where does it fit? A systematic literary review." In: *International Journal of Computer Science Education in Schools* 2018 (2018), pp. 41–60.

[60] J. Lockwood and A. Mooney. "Developing a Computational Thinking Test using Bebras problems". In: *TACKLE: the 1st Systems of Assessments for Computational Thinking Learning workshop at EC-TEL 2018*. Vol. 2018. 2018.

[61] J. Lockwood, A. Mooney, and G. O'Mahony. "An Introduction to delivering Computational Thinking". In: *Computers in Education Society of Ireland Conference 2017*.

[62] J. J. Lu and G. H. Fletcher. "Thinking about computational thinking". In: *ACM SIGCSE Bulletin* 41.1 (2009), pp. 260–264.

[63] S. Y. Lye and J. H. L. Koh. "Review on teaching and learning of computational thinking through programming: What is next for K-12?" In: *Computers in Human Behavior* 41 (2014), pp. 51–61.

[64] L. Mannila, V. Dagiene, B. Demo, N. Grgurina, C. Mirolo, L. Rolandsson, and A. Settle. "Computational thinking in K-9 education". In: *Proceedings of the Working group reports of the Innovation & Technology in Somputer Science Education Conference*. 2014, pp. 1–29.

[65] K. Mensing, J. Mak, M. Bird, and J. Billings. "Computational model thinking and computer coding for US Common Core Standards with 6 to 12 year old students". In: *Proceedings Emerging eLearning Technologies and Applications (ICETA)*. 2013, pp. 17–22.

[66] A. Mooney, J. Duffin, T. Naughton, R. Monahan, J. Power, and P. Maguire. "PACT: An initiative to introduce computational thinking to second-level education in Ireland". In: *Proceedings of International Conference on Engaging Pedagogy (ICEP)*. 2014.

[67] Robles G. Moreno-León J. and M. Román-González. "Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking." In: *Revista de Educación a Distancia* (2015).

[68] P. Morreale and D. Joiner. "Changing perceptions of computer science and computational thinking among high school teachers". In: *Journal of Computing Sciences in Colleges* 26.6 (2011), pp. 71–77.

[69] P. Morreale, D. Joiner, and G. Chang. "Connecting undergraduate programs to high school students: teacher workshops on computational thinking and computer science". In: *Journal of Computing Sciences in Colleges* 25.6 (2010), pp. 191–197.

[70] P. Morreale, C. Goski, L. Jimenez, and C. Stewart-Gardiner. "Measuring the impact of computational thinking workshops on high school teachers". In: *Journal of Computing Sciences in Colleges* 27.6 (2012), pp. 151–157.

[71] N. Nesiba, E. Pontelli, and T. Staley. "DISSECT: Exploring the relationship between computational thinking and English literature in K-12 curricula". In: *Proceedings Frontiers in Education Conference (FIE)*. 2015.

[72] Tom Neutens and Francis Wyffels. "Bringing Computer Science Education to Secondary School: A Teacher First Approach". In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM. 2018, pp. 840–845.

[73] Carl O'Brien. "Computer science to be fast-tracked onto Leaving Cert". In: (2017). [accessed August 8, 2018].

[74] S. Papert. *Mindstorms: Children, computers and powerful ideas*. New York, NY, USA: Basic Books, Inc., 1980.

[75] Bjarke Kristian Maigaard Kjær Pedersen, Kamilla Egedal Andersen, Simon Köslich, Fardin Sherzai, Jacob Nielsen, et al. "Towards playful learning and computational thinking—Developing the educational robot BRICKO". In: *Integrated STEM Education Conference (ISEC), 2018 IEEE*. IEEE. 2018, pp. 37–44.

[76] K. L. Pokorny and N. White. "Computational thinking outreach: reaching across the K-12 curriculum". In: *Journal of Computing Sciences in Colleges* 27.5 (2012), pp. 234–242.

[77] K. Quille, S. Bergin, and A. Mooney. "PreSS#, A Web-Based Educational System to Predict Programming Performance". In: *International Journal of Computer Science and Software Engineering* (2015), pp. 178–189.

[78] Keith Quille, Natalie Culligan, and Susan Bergin. "Insights on Gender Differences in CS1: A Multi-institutional, Multi-variate Study." In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM. 2017, pp. 263–268.

[79] Mohd Rizal Bin Razak and Nor Zalina Binti Ismail. "Influence of mathematics in programming subject". In: *AIP Conference Proceedings*. Vol. 1974. 1. AIP Publishing. 2018, p. 050011.

[80] A. Repenning, D. Webb, and A. Ioannidou. "Scalable game design and the development of a checklist for getting computational thinking into public schools". In: *Proceedings of the 41st ACM technical symposium on Computer science education*. 2010, pp. 265–269.

[81] L. Ribeiro, D. J. Nunes, M.K Da Cruz, and E.D.S Matos. "Computational Thinking: Possibilities and Challenges". In: *Proceedings of 2nd Workshop-School on Theoretical Computer Science (WEIT)*. 2013, pp. 22–25.

[82] Moreno-León J. Román-González M. and G. Robles. "International Conference on Computational Thinking Education (CTE 2017)". In: 2017.

[83] J. F. Roscoe, S. Fearn, and E. Posey. "Teaching Computational Thinking by Playing Games and Building Robots". In: *Proceedings International Interactive Technologies and Games Conference (iTAG)*. 2014.

[84] Olgun Sadik, Anne-Ottenbreit Leftwich, and Hamid Nadiruzzaman. "Computational Thinking Conceptions and Misconceptions: Progression of Preservice Teacher Thinking During Computer Science Lesson Planning". In: *Emerging Research, Practice, and Policy on Computational Thinking*. Springer, 2017, pp. 221–238.

[85] J. Shailaja and R. Sridaran. "Computational Thinking the Intellectual Thinking for the 21st century". In: *International Journal of Advanced Networking & Applications Special Issue* 2015 (2015), pp. 39–46.

[86] M. Sherman and F. Martin. "The assessment of mobile computational thinking". In: *Journal of Computing Sciences in Colleges* 30.6 (2015), pp. 53–59.

[87] M. M. Sysło and A. B. Kwiatkowska. "Learning Mathematics supported by computational thinking". In: *Constructionism and Creativity* (2014), pp. 258–268.

[88] *TIOBE Index*. URL: `https://www.tiobe.com/tiobe-index/`.

[89] R. Taub, M. Armoni, and M. Ben-Ari. “CS unplugged and middle-school students’ views, attitudes, and intentions regarding CS.” In: *ACM Transactions on Computing Education (TOCE)* (2012).

[90] M. Van Dyne and J. Braun. “Effectiveness of a computational thinking (cs0) course on student analytical skills”. In: *Proceedings of the 45th ACM technical symposium on Computer science education*. 2014, pp. 133–138.

[91] J. Vanicek. “International Conference on Informatics in Schools: Situation, Evolution, and Perspectives”. In: *Bebras informatics contest: criteria for good tasks revised*. 2014.

[92] C. Vieira and A. J. Magana. “Using backwards design process for the design and implementation of computer science (CS) principles: A case study of a colombian elementary and secondary teacher development program”. In: *Proceedings Frontiers in Education Conference (FIE)*. 2013, pp. 879–885.

[93] J. Voogt, P. Fisser, J. Good, P. Mishra, and A. Yadav. “Computational thinking in compulsory education: Towards an agenda for research and practice”. In: *Education and Information Technologies* 20.4 (2015), pp. 715–728.

[94] H. Webb and M. B. Rosson. “Using scaffolded examples to teach computational thinking concepts”. In: *Proceeding of the 44th ACM technical symposium on Computer science education*. 2013, pp. 95–100.

[95] Mary Webb, Niki Davis, Tim Bell, Yaacov J Katz, Nicholas Reynolds, Dianne P Chambers, and Maciej M Sysło. “Computer science in K-12 school curricula of the 2lst century: Why, what and when?” In: *Education and Information Technologies* 22.2 (2017), pp. 445–468.

[96] L. Werner, J. Denner, S. Campe, and D. C. Kawamoto. “The fairy performance assessment: measuring computational thinking in middle school”. In: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. 2012, pp. 215–220.

[97] J. M. Wing. “Computational thinking”. In: *Communications of the ACM* 49.3 (2006), pp. 33–35.

[98] J. M. Wing. "Computational thinking and thinking about computing". In: *Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences* 366.1881 (2008), pp. 3717–3725.

[99] U. Wolz, M. Stone, S. M. Pulimood, and K. Pearson. "Computational thinking via interactive journalism in middle school". In: *Proceedings of the 41st ACM technical symposium on Computer science education*. 2010, pp. 239–243.

[100] U. Wolz, M. Stone, K. Pearson, S. M. Pulimood, and M. Switzer. "Computational thinking and expository writing in the middle school". In: *ACM Transactions on Computing Education (TOCE)* 11 (2011), p. 2.

[101] B. Worrell, C. Brand, and A. Repenning. "Collaboration and Computational Thinking: A classroom structure". In: *Proceedings Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2015, pp. 183–187.

[102] A. Yadav, N. Zhou, C. Mayfield, S. Hambrusch, and J. T. Korb. "Introducing computational thinking in education courses". In: *Proceedings of the 42nd ACM technical symposium on Computer science education*. 2011, pp. 465–470.

[103] A. Yadav, C. Mayfield, N. Zhou, S. Hambrusch, and J. T. Korb. "Computational thinking in elementary and secondary teacher education". In: *ACM Transactions on Computing Education (TOCE)* 14 (2014).

[104] Aman Yadav, Sarah Gretter, Jon Good, and Tamika McLean. "Computational thinking in teacher education". In: *Emerging Research, Practice, and Policy on Computational Thinking*. Springer, 2017, pp. 205–220.

[105] K. Yevseyeva and M. Towhidnejad. "Work in progress: Teaching computational thinking in middle and high school". In: *Proceedings Frontiers in Education Conference (FIE)*. 2012.

[106] I. Zur Bargury. "A new curriculum for junior-high in computer science". In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. 2012, pp. 204–208.