

# Programming for Engineers: Removing Barriers and Improving Outcomes

Name: Kin Kwan Leung



Supervisor: Dr. Steven Lind

Submission year: 2021



A dissertation submitted to The University of  
Manchester for the degree of MEng Mechanical  
Engineering in the Faculty of Science and Engineering

Department of Mechanical, Aerospace and Civil  
Engineering

## **Table of Contents**

<b>List of Tables</b>	<b>4</b>
<b>List of Figures</b>	<b>5</b>
<b>Glossary</b>	<b>6</b>
<b>Abstract</b>	<b>7</b>
<b>Declaration</b>	<b>8</b>
<b>Intellectual Property Statement</b>	<b>9</b>
<b>Acknowledgements</b>	<b>10</b>
<b>1. Introduction</b>	<b>11</b>
<b>2. Methodology and Redesign</b>	<b>13</b>
2.1 Finding Secondary Data	13
2.2 Finding Primary Data	15
2.3 Programming the Software	15
<b>3. Literature Review</b>	<b>17</b>
3.1 Barriers to Learning Programming	17
3.2 Teaching Methods Currently In Use	17
3.3 University Level Teaching Method	20
3.3.1 Modifications to Traditional Method	20
3.3.2 E-Learning	27
3.3.3 Projects	33
3.3.4 Competitions	36
3.3.5 Puzzles	36
3.4 Secondary School Level Teaching Method	39
3.4.1 Modifications to Traditional Method	39
3.4.2 Scratch	40
<b>4. Results and Discussion</b>	<b>43</b>
<b>5. Cheatsheets and Software</b>	<b>51</b>
5.1 Cheatsheets	52
5.2 MATLAB Engine API	52
5.3 Software	53

<b>6. Conclusions</b>	<b>59</b>
6.1 Limitations	61
6.2 Future Work	62
<b>7. References</b>	<b>64</b>
<b>Appendices</b>	<b>75</b>
Appendix A - Reflection and Project Management	75
Appendix B - Survey	82
Appendix C - Participants' Paragraph Answers and T-Test Values	87
Appendix D - Cheatsheets	96
Appendix E - Main Menu Code	101
Appendix F - Questions Module Code	103
Appendix G - InputFile.txt Code	134
Appendix H - Test Data and Outcome	135

Final Word Count: 26,276

## List of Tables

1. The grade marks obtained by the students using the traditional approach	19
2. Numerical representation of how well the intended course outcomes were achieved by 'LBTL' and the traditional method	24
3. Average mark and failure rate of mechanical engineering students using the teaching method proposed by Nikolic et al.	27
4. Percentage of students who passed, failed and withdrew using the self-paced course in different semesters	29
5. Average results of each exercise in the robot group project	35
6. Mean programming achievements before and after the experiment	41
7. Numbers and calculations for the T-test	92

## List of Figures

1. The percentage of students who achieved the grade in spring 2012 for the 'Extreme Apprenticeship' approach	24
2. The marks that the students achieved in the first semester programming course (left) and the second semester programming course (right) whilst using 'Coursemarker'	28
3. An example puzzle of the 'PPP' software	37
4. An example of the punched hole exercise	38
5. "Programming skills are important to have in general, regardless of profession"	43
6. "Programming skills are integral to engineers"	44
7. "Would you consider the students to be well-informed about what engineering entails when considering their future career options?"	45
8. Whether teachers believe the given aspects are difficult for students	47
9. Whether teachers agree that the given method should be used to teach programming	49
10. The main menu of ART displayed on the command-line interface	53
11. The command-line interface after choosing option 1 (Fill In the Gap)	54
12a and 12b. The command-line interface after entering the correct line of code and after entering an incorrect line of code for Fill In the Gap exercise	55
13. The command-line interface after entering an incorrect followed by a correct numerical answer for the Correct the Error exercise	56
14. The descriptions for both of the Program Writing exercises	56
15a-15c. Command-line interface if the answers were not provided, incorrect and correct for the Program Writing exercise	58
16. Initial Project Plan	79
17. Revised Project Plan	80
18. Actual Project Timeline	81
19a-19e. Screenshots of the survey	82-86
20a-20e: Screenshots of the Python and MATLAB cheatsheets	96-100

## Glossary

- Application Program Interface (API) = An interface or set of code that allows communication between two separate applications (Mulesoft, 2020)
- Cognitive Apprenticeship = Method of learning where 'learners learn from a more experienced person by way of cognitive and metacognitive skills and processes' (Jonassen, 2008)
- Command-line interface = An 'interface [that] allows the user to interact with the computer by typing in commands' (BBC, 2021a)
- Framework = A pre-constructed foundation with which programmers can use to program applications and software (Christensson, 2013)
- Function = A chunk of code that is executed only when called by the current thread
- Graphical user interface (GUI) = An interface that a user sees and interacts with when utilising an electronic device (OmniSci, 2020)
- Microworld = 'A conceptual model of some aspect of the real world' that allows users to 'explore or manipulate the logic, rules, or relationships' (Hogle, 1995)
- Module = A Python text file containing 'executable statements as well as function definitions' (Python Software Foundation, 2021d)
- Statement = A line of code
- Test data = A series of valid, extreme and invalid input data for a script (BBC, 2021b)

## **Abstract**

With the world becoming more digitally reliant and sophisticated, programming has increasingly become a core tool for much of society, engineers included. As such, the demand for improving the teaching quality of programming is at an all-time high. The lack of studies on teaching secondary school engineering students programming suggests a possible knowledge gap, as many of the encountered studies focus on teaching programming to university level students. Therefore, this project researches into the pedagogical literature and conducts a survey on professionals to find barriers that result in high dropout and failure rates and teaching methods that are considered major improvements to the current methods used, particularly for secondary schools. New software tools based in Python and Python-called MATLAB are written to aid teaching and learning at secondary schools based on the results and findings from the literature and surveys of professionals. The results from the project have identified that motivation is a significant factor in the problem, with the lack of role models, lack of problem-solving skills, difficulty in debugging, and learning syntax and concepts all playing a role in the high dropout rate. 67% of the survey participants believe that students have no knowledge of what engineering entails, meaning that students aspiring to be engineers do not know that programming is a significant part of engineering. Another key finding is that e-learning is currently the more mature method of teaching programming and is suitable due to the low failure rate. The programmed software tool uses 3 unique exercises to combat the barriers that have been identified. Further work on the software tool such as adding competition capabilities and student testing will be needed in order to make the software an impactful teaching tool. Further research into each identified barrier is also needed.

## **Declaration**

I hereby declare that this dissertation titled 'Programming for Engineers: Removing Barriers and Improving Outcomes' is of my own original work and that no portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.



## **Intellectual Property Statement**

- i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
- ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.
- iii. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the dissertation, for example graphs and tables (“Reproductions”), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
- iv. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy, in any relevant Dissertation restriction declarations deposited in the University Library, and The University Library’s regulations.

## **Acknowledgements**

I would like to thank my supervisor, Dr. Steven Lind for his continual guidance and recommendations. Without him, this project simply would not have been possible.

I would also like to thank Dr. Martin Simmons from the University of Manchester and Alex Clewett from Technocamps for helping me distribute the surveys during these strange times. Their assistance has ensured that the project was completed with minimal resistance.

## 1. Introduction

Over time, but especially during the past century, the development of technology and innovation caused the world to undergo a transformation where the standard of living across most of the world became higher. As a result, the engineering solutions responsible for supplying that standard became more sophisticated. Examples of the more intricate solutions include modelling and simulation engineering, and CNC machine operating. To combat this, the engineering sector embraced the use of programming to find complex answers, and teaching programming to university-level students gradually became the norm. As the Information Age continued, however, it became evident that that was insufficient and programming should have been taught at an earlier stage, as reflected by the need of the Department of Education (2013a; 2013b) to change the national curriculum to include computing for students in secondary schools as well as the emphasis of programming physical components such as microcontrollers and actuators in design and technology.

Another implication by the change in curriculum is that there are insufficient 'digitally literate' individuals in an ever-developing society, which would place those people at a disadvantage when looking for job prospects against people who do know how to program (Department of Education, 2013a). Hence, the Department of Education (2013a) necessitated the teaching of programming to secondary school students.

Whilst there had been numerous studies relating to the optimum methods for teaching programming to university students such as the use of 'Team-Based Learning' by Elnagar and Ali (2012) or the mixture of 'LEGOs and LabVIEW' by Wang (2001), there has been little focus on the teaching methods of programming to engineering students in secondary schools, either in terms of programming languages sought by engineers, or problems in an engineering context.

The purpose of this project is two-pronged: the first aim is to identify successful teaching methods, obstacles as well as barriers that currently plague programming students in secondary schools (commonly between the ages of 11 to 18). Barriers may be in the form of theory that programming students consider difficult, or it may appear in the form of stereotypes that discourages students from seeking a career in programming in the first place. To accomplish the aim, a set of objectives have been established:

- To create, distribute, collect and analyse a questionnaire from professionals in secondary schools relating to the methods of teaching and identify the barriers that students face
- To study and critically evaluate pedagogical literature for the various successful teaching methods that have been examined

The second aim of the project is to utilise what has been discovered in the survey and the pedagogical literature to create software with a series of programming exercises that work around the obstacles to assist in the study of engineering students in secondary schools. Due to MATLAB being recognised as valuable ‘for simulation-based engineering research and even in some experimental setups’ and the ability of Python to call MATLAB, the objectives of the second aim are outlined as (Azemi and Pauley, 2008):

- To study Python and all of its API capable of assisting in the development of the software as well as the deployment of MATLAB
- To design activities highlighting the common oversights students have with programming
- To design programming exercises in Python and Python-called MATLAB based on GCSE and A-Level engineering problems

- To design cheatsheets for MATLAB and Python to improve understanding of the students
- To test the software with a series of test data in order to identify any bugs

What remains of the project is structured as such: firstly, the tools, methods, and reasoning for choosing to use the methods are discussed in the next section. Afterwards, a literature review based on the first aim is included, followed by an introduction and discussion of the results obtained from the survey. Next, the main features of the written code are presented. Finally, a conclusion based on the research carried out is drawn at the end, before an evaluation of the project and recommendations for future work.

## **2. Methodology and Redesign**

### **2.1 Finding Secondary Data**

In terms of secondary sources of data, a detailed literature review was carried out on academic journals based on Google Scholar, and reports from government websites and examination boards to evaluate teaching techniques that teachers at secondary schools have used to boost the understanding of the students. Many of the academic journals were obtained from the Association of Computing Machinery (ACM), IEEE, and ScienceDirect, with a handful from Semantic Scholar and ResearchGate. An additional focus was made on evaluating teaching methods of programming to university-level students because of the breadth of related academic journals that already exist (Vihavainen et al., 2014). Once collated, an analysis was carried out to find the optimum methods for teaching programming to secondary school students beginning with the identification of barriers that cause researchers to research into the topic of learning programming. Subsequent

subsections then go into the methods of programming teaching that are currently used for university-level and secondary school level, before an analysis into the research into newer successful teaching methods, bearing in mind that the teaching structure of university and secondary schools are unlike hence slight adjustments may be required for the proposed teaching methods.

Filters were applied to the searches to ensure articles were relevant to the research topic, and several keywords, as well as a combination of keywords, were used. The keywords were:

- Diversity
- Engineering programming
- Introductory programming
- Successful programming
- Programming
- Programming skills/course/syntax
- Secondary/Middle school
- Students
- Teaching programming/methods
- University
- Difficulty
- Learning
- Barriers
- Motivation

To prevent old data, which may no longer be accurate, from influencing the project, research papers before 1997 were not considered. Also, extra care was made to ensure that only peer-reviewed journals were included in the literature review. Many

of the research papers considered were not reliant on any particular programming languages since the process of learning programming goes beyond learning any individual programming language (Vihavainen et al., 2011).

The data gathered from the pedagogical literature then formed the basis of the primary data collection.

## **2.2 Finding Primary Data**

Primary data was acquired in the form of surveys completed by professionals in secondary schools; the reason behind this was due to the experience that the professionals have when teaching the secondary school students and may have spotted barriers that current programming students have to deal with. The survey was split into four main sections (see Appendix B): one regarding programming in general; one regarding factors that may affect a student's choice of career; one regarding the challenges that students have about learning programming; one regarding programming teaching methods that the professionals would consider to be successful. Digital copies of the survey were then distributed to secondary schools based on the contacts that the university currently has as well as the contacts that the author has.

The primary and secondary data collection satisfied the first two objectives listed in the introduction and was used to conclude the first aim of the project.

## **2.3 Programming the Software**

Concurrently to carrying out the literature review and the development of the survey, time was allocated to study MATLAB and Python programming languages and its Application Program Interface (API) to allow the communication between MATLAB

and Python codes. This was done by researching online resources and books offered by the university library. The change to not include GUIs due to the difficulty of coding the software in the timeframe given meant that research into the frameworks of Python did not occur.

The first step in programming the software was done by researching the specifications of GCSE and A-Level Mathematics for problems related to engineering, such as mechanics (WJEC, 2019d). Then, using the data gathered from primary and secondary sources, the type of software and engineering-related exercises that should be coded were designed and programmed. During all of this, test data was used to flesh out all the bugs and errors in order to prevent unneeded crashes of the software. Finally, cheatsheets containing the syntax of Python and MATLAB were also made so that students can refer to the sheets when assistance with the exercises is necessary. With the completion of programming the software, all the objectives for the second aim were completed.



### **3. Literature Review**

As set out in the introduction, a series of pedagogical literature was reviewed to find various successful teaching methods. The section is split such that some of the barriers that motivate the research by various researchers are introduced. Then, what is considered current teaching methods are discussed. Following that, research papers are divided between university-level teaching methods and secondary school teaching methods, which is further subdivided into the various categories of teaching methods. In some instances, the teaching methods discussed belong to more than one category.

#### **3.1 Barriers to Learning Programming**

Numerous research has taken place on barriers that could potentially prevent students from learning to program. According to Yacob and Saman (2012), problems such as the lack of problem-solving and abstraction skills have been identified. Although confidence does not necessarily mean success, the lack of confidence and motivation has been shown to be an important factor in students dropping out of courses (Yacob and Saman, 2012). A separate study by Guo (2018) found that non-native English speakers had trouble reading English programming materials such as textbooks and online resources as well as source codes by other people. Nikolic et al. (2018) stated that decomposing problems when learning to program is difficult.

#### **3.2 Teaching Methods Currently In Use**

To begin the investigation into successful teaching methods, the second step was to identify the teaching methods currently in use in schools around the United

Kingdom. Although there is no set method for teaching programming, the examination boards provide the assessments for GCSEs and A-Levels as well as guidance for teaching the subject.

According to the AQA (2020), Pearson (2020), OCR (2020a) and, WJEC (2019a) the method of assessment for GCSE Computer Science is two exams, which may be written exams, or a written exam and an on-screen exam depending on the examination board. Furthermore, AQA (2020), OCR (2020a), and WJEC (2019a) require students to undertake a 20-hour programming project, with the WJEC project accounting for 20% of the GCSE marks. On the other hand, of the examination boards which provide A-Level Computer Science, the course is split into AS Level and A2 Level depending on the level the student would like to achieve. For AS Level, AQA (2019a), OCR (2020b), and WJEC (2020) all have similar assessments with only two exams, whereas for the A2 Level the aforementioned examination boards require a total of four exams as well as a practical project (WJEC, 2019b; OCR, 2020c). Instead of AS and A2 Level computer Science, Pearson (2019a) awards BTEC Level 3 'Software Development Context and Methodologies', which requires two on-screen exams to be completed.

For the WJEC (2021), the newly added blended-learning resources along with the mini activities are supplied for the teachers to plan the lessons around. On the contrary, AQA (2021) provided the teachers with textbooks for teaching the material to students. From the material that has been provided to the teachers by the examination boards as well as the type of assessments available, the teaching methods that are currently in use for secondary schools students can be seen as traditional methods used for assessing other more mature subjects despite having programming projects since the textbooks and blended-learning materials are more

towards content-learning rather than applying the newly-acquired knowledge to practical problems (AQA, 2021; WJEC, 2021). The result that can be observed when using the traditional approach for teaching computer science, and hence programming, is a high percentage of students (34.7% - 41.2%) who achieved grades of D/3 and below according to the various examination boards in 2019 (AQA, 2019b; Pearson, 2019b; OCR, 2019; WJEC, 2019c).

*Table 1: The grade marks obtained by the students using the traditional approach (source: Blumenstein, 2002)*

Semester 1, 2000		Semester 2, 2000		Semester 1, 2001		Semester 2, 2001	
Gr.	%	Gr.	%	Gr.	%	Gr.	%
<b>HD</b>	11.8	<b>HD</b>	8.11	<b>HD</b>	18.1	<b>HD</b>	5.5
<b>D</b>	12.2	<b>D</b>	8.11	<b>D</b>	13.2	<b>D</b>	15.1
<b>C</b>	15.1	<b>C</b>	10.4	<b>C</b>	20.6	<b>C</b>	12.4
<b>P</b>	20.4	<b>P</b>	16.7	<b>P</b>	13.2	<b>P</b>	21.1
<b>F</b>	14.7 (28.2)	<b>F</b>	26.6 (37.8)	<b>F</b>	12.8 (23.5)	<b>F</b>	12.39 (26.1)

Whilst there is no set method of teaching for programming at the university level either, Vihavainen et al. (2011) described what is considered the traditional approach to teaching programming as lectures based around a programming language and several pieces of coursework with predetermined model answers. Blumenstein (2002) conducted a study on such an approach as the method was considered an improvement from before the year 2000. With two hours of lectures and two hours of laboratory sessions per week and centred around the Java programming language, the course described by Blumenstein (2002) utilised two exams, a project, and the laboratory sessions as the assessment methods. Having minor changes over the four semesters, the failure rates were 14.7%, 26.6%, 12.8%, and 12.39% respectively for the students who participated in the entirety of the course (see Table 1). Although Blumenstein (2002) concluded that the failure rates were

acceptable, the high failure rates would later be the premise for further research into alternative teaching methods of programming (El-Zein et al., 2009; Vihavainen et al., 2011; Elnagar and Ali, 2012). Unfortunately, the aforementioned results may be inaccurate as the grade profile percentages do not add up to 100% per semester and no number of participants were given, and the age of the study suggests the results are redundant.

### **3.3 University Level Teaching Method**

#### **3.3.1 Modifications to Traditional Method**

Whilst several studies all agreed that the traditional method of lectures and examinations are unsuitable for teaching programming due to the high dropout rates and lack of engagement from the students, each group sought to modify the traditional method instead of dropping the method altogether (Vihavainen et al., 2011; Elnagar and Ali, 2012; Rubio et al., 2013).

Early research into improving the outcomes for teaching programming included pair programming (McDowell et al., 2002; Nagappan et al., 2003). The main difference between the approach described by Blumenstein (2002) and pair programming is that students were paired up for laboratory sessions and assignments. In the case of McDowell et al. (2002), the resulting improvements included higher quality assignments and a higher retention rate of the students when compared to the traditional approach, though an anomaly was observed on the scores for the final exam. Also using null hypothesis testing, Nagappan et al. (2003) concluded that pair programming resulted in a higher pass rate for exams and coursework, and a reduction in workload for the teaching staff as compared to a control group learning via the traditional approach. Even though the research seems dated, several

researchers would incorporate pair programming and teamwork into newer teaching methods of programming, which are described below.

Lui et al. (2004) implemented the 'Perform' approach to avoid students misinterpreting the concepts given and to avoid ambiguity. Instead of metaphors, analogies, and technical terms, extra examples are given to the students instead (Lui et al., 2004). New concepts taught using the 'Perform' approach would be built up from the examples in order to foster a sense of familiarity and retain confidence, and integrated development environments would be avoided to also retain confidence. The results provided indicated that confidence was built up as a result of the course and the failure rate was dropped by 40% compared to the traditional method (Lui et al., 2004). Because little data was supplied by Lui et al. (2004), the statistic provided by the study cannot be verified, and with insufficient information regarding the method utilised, repeating the method is infeasible for other programming courses. Due to the age of the study, the method proposed by Lui et al. (2004) may be obsolete as new research is conducted and programming evolves. Stressing that other programming courses favour teaching the concepts rather than the skills, Woodley and Kamin (2007) devised a strategy with a higher emphasis on skill-learning; lecture hours were reduced to one every week, and students were assigned projects to work on. On a weekly basis, a two-hour discussion session was assigned where students were asked to hold a presentation detailing the progress of the assignment, which was then subjected to questions and constructive criticism (Woodley and Kamin, 2007). The success of the course was measured by asking the students to replicate the first assignment designated to the students; Woodley and Kamin (2007) reported that less experienced programming students improved more than the experienced programming students by observing

the quality of the code. According to the study, extra effort was taken to analyse the costs of the course, so smaller departments may replicate the same results. However, an improvement over the traditional approach cannot be observed due to the lack of statistics.

Vihavainen et al. (2011) defined the 'Extreme Apprenticeship' method as further development on top of the cognitive apprenticeship model. The goal of the paper was to describe a method of teaching that reduces the dropout rate of students from programming courses by maintaining interest and motivation. The idea behind the model is that on top of the modelling (a demonstration of the concept performed by an expert), scaffolding (exposure of exercises generated by experienced instructors for the students) and fading (mastery of a task) stages of the cognitive apprenticeship model, a set of criteria has to be satisfied; the main points include a higher emphasis on completing a large set of relevant exercises and keeping the hours of lectures to a minimum, as well as continuous feedback to ensure that students were encouraged whenever a small goal has been achieved to retain motivation or guide the student whenever the exercise seemed challenging (Vihavainen et al., 2011). The study concluded that the 'Extreme Apprenticeship' had a significant positive effect on the number of students that passed, with a pass rate of 70.1% and 86.4% as opposed to the 47.7% and 50.0% respectively for the two courses from the same semester the previous year. Although undeniable that the pass rate is higher than any previously observed pass rate by a margin, there has been a general trend of improvement from previous years such that concluding this after only one semester is premature without more data after the new method has been adopted. This may also explain the improved pass rate of this method being lower than the traditional method, as some of the challenges of the method

had not yet been resolved. The focus on the positives meant no possible drawbacks were considered when applying this method. Nonetheless, the goal of increasing motivation can be observed by the anonymous feedbacks remarking the course as 'motivating and rewarding' (Vihavainen et al., 2011).

On the contrary, Elnagar and Ali (2012) proposed the use of what is termed the 'Modified Team-Based Learning' approach for teaching programming. Similar to the approach suggested by Vihavainen et al. (2011), the aim of the paper was to discover a method of teaching that reduces the dropout rate of students. However, the 'Modified Team-Based Learning' also drastically reduced the lecture time whilst also keeping the final examination in (Elnagar and Ali, 2011). According to Elnagar and Ali (2011), the students were given the material beforehand and each session was then split into multiple smaller sections: lecture, discussion, quiz, and feedback; teams were formed at the beginning of the semester for the group discussions, and the feedback of the quiz was given immediately after the quiz (Elnagar and Ali, 2012). In the study, '603 students over two years' participated, meaning the experiment covered four semesters, which is a solid amount of data compared to the study conducted by Vihavainen et al. (2011). From Figure 1, the results show that there is a clear positive trend between the new method Elnagar and Ali (2012) utilised compared to the traditional method used by the control group, as more people obtained a higher grade using the 'Modified Team-Based Learning' method than the traditional approach.

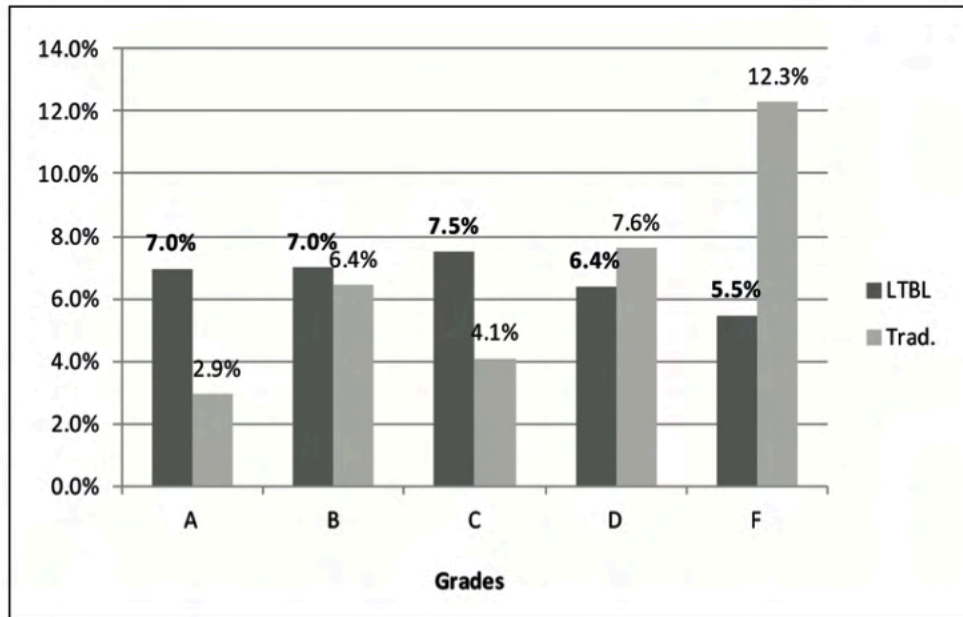


Figure 1: The percentage of students who achieved the grade in spring 2012 for the 'Extreme Apprenticeship' approach (source: Elnagar and Ali, 2012)

Furthermore, Table 2 shows that all of the learning outcomes set by Elnagar and Ali (2012) were better achieved using the 'Modified-Team-Based Learning' approach than the traditional approach.

Table 2: Numerical representation of how well the intended course outcomes were achieved by 'LBTL' and the traditional method (source: Elnagar and Ali, 2012)

Term	Mode	A	B	C	D	E	F
Fall 2010	LTBL	75	73	73	71	75	<b>66</b>
Fall 2010	Traditional	72	70	71	<b>61</b>	<b>66</b>	<b>57</b>
Spring 2011	LTBL	90	83	83	73	74	71
Spring 2011	Traditional	84	80	74	<b>60</b>	<b>65</b>	<b>61</b>
Fall 2011	LTBL	87	85	74	71	79	72
Fall 2011	Traditional	84	83	<b>67</b>	<b>61</b>	74	<b>68</b>
Spring 2012	LTBL	92	88	75	72	77	71
Spring 2012	Traditional	83	83	<b>63</b>	<b>60</b>	<b>65</b>	<b>60</b>

One main advantage of the 'Extreme Apprenticeship' approach by Vihavainen et al. over the 'Modified Team-Based Learning' approach is that the method has been



identified with motivation problems such as socialising rather than working, which the 'Extreme Apprenticeship' method does not have (Elnagar and Ali, 2012).

Although the previous two methods can be adapted for use by engineering students, the approach Rubio et al. (2013) proposed was gauged with engineers in mind, who may not have the computational thinking of students studying computer engineering or science. The method involves utilising Arduino board in an approach termed 'Physical Computing Paradigm', where the 'computational concepts ... [can be taken] into the real world' (Rubio et al., 2013). The approach has very little difference to the traditional teaching method except for the lecturer demonstrating the concepts using the Arduino board and simple circuitry; for instance, using 'loudspeaker to teach arrays' and lights are used to teach conditional structures (Rubio et al., 2013). The demonstrations were then the premise for laboratory sessions held twice to three times during the duration of the course described. According to the study held by Rubio et al. (2013), an increase of 32% of students attained a good level of programming between the 'physical computing paradigm' and the traditional method. Furthermore, an increase of 21% students felt comfortable programming independently from the lecturer and lab assistants compared to the traditional method (Rubio et al., 2013). Similar to the 'Extreme Apprenticeship' method, the study concluded that there was an increase in the motivation by the students as well as an increase in the number of students who enjoyed programming after adopting the new method (Rubio et al., 2013). Nevertheless, the study did not disclose the precise number of students who participated in this study, nor defined what 'a good programming level' is and so is difficult to compare effectiveness with the other teaching methods.

Attempts at introducing programming to several disciplines at once were conducted separately by Nikolic et al. (2018) and Dawson et al. (2018) in order to improve the programming pass rates of students not studying computer science. Videos and quizzes on topics that needed to be learnt prior to the lecture were released before the lecture. Although both studies reduced the time for lectures, Nikolic et al. (2018) opted for laboratory sessions, where students worked on engineering-based exercises supplied by programming textbooks whereas Dawson et al. (2018) opted for group assignments such that any student who was discouraged from being unable to solve a problem may be supported by peers. Of the 166 mechanical engineers who entered the course proposed by Nikolic et al. (2018) over the course of two years, the average failure rate was 9.7% (see Table 3), lower than the traditional method mentioned in the previous subsection and lower than the proposed 'Extreme Apprenticeship' approach by Vihavainen et al. (2011). On the other hand, the method proposed by Dawson et al. (2018) achieved a failure rate of 4% for degrees unrelated to science, arts, and commerce over a course of two semesters; the pass rate was 83% on the proposed method as opposed to the 75% of the more traditional approach used in the same year (Dawson et al., 2018). In terms of motivation, Nikolic et al. (2018) observed that unless students had a desire to learn about programming passively, the method proposed would be ineffective as a teaching method. In contrast, Dawson et al. (2018) attempted to combat this motivational barrier by centring the course around a project that students could choose.

Of the methods described, Dawson et al. (2018) and Nikolic et al. (2018) were the more successful of the teaching methods based on pass rate. However, when considering the aim of increasing motivation as well, the teaching method by

Vihavainen et al. (2011) seemed to have a higher combined pass rate and motivation retention rate than either of the aforementioned methods.

*Table 3: Average mark and failure rate of mechanical engineering students using the teaching method proposed by Nikolic et al. (source: Nikolic et al., 2018)*

Mechanical N = 166		
AVERAGE	SD	FAILS
70.1%	18.2	10.1%
65.5%	18.0	9.2%
67.8%	18.1	9.7%

### 3.3.2 E-Learning

Contrary to the viewpoint of Vihavainen et al. (2011) that students should not solve problems by themselves due to the possibility of learning bad habits, several studies embraced independent working by introducing automated e-learning approaches of teaching and an emphasis on self-marking systems because having different people mark results would cause inconsistency (Higgins et al., 2005).

Higgins et al. (2005) employed 'Coursemarker', which according to the researchers would increase the reliability, consistency, and quality of the feedback given to students as well as scalability. Specifically-worded questions and skeleton codes for the questions are stored and supplied using 'Coursemarker', which students can access; the students then develop solutions and submit the answers back into the system, which is then checked based on a number of pre-defined factors and feedback is then relayed back to the student automatically (Higgins et al., 2005). The system was incorporated into two courses assessed via reports, multiple-choice questions, and reports on top of two weekly exercises using 'Coursemarker'. From that, six years worth of data was gathered, and Higgins et al. (2005)

concluded that ‘Coursemarker’ had an impact on students becoming good programmers and achieving better grades with a pass rate of 92% and 93% for the two courses and most students obtaining percentages of 70% or higher (see Figure 2). Although the study was done rigorously with contingencies in place to counter plagiarism and bugs, the data is outdated due to the age of the study, and the teaching method may no longer be suitable for implementation into newer courses. Additionally, no data was provided prior to adopting ‘Coursemarker’, and so the provided data can not be used to gauge by how much the students had improved with and without the system in place. In contrast to the ‘Extreme Apprenticeship’ and ‘Physical Computing Paradigm’ approaches from the previous subsection, there was no analysis done on the motivation of the students once ‘Coursemarker’ was adopted (Higgins et al. 2005).

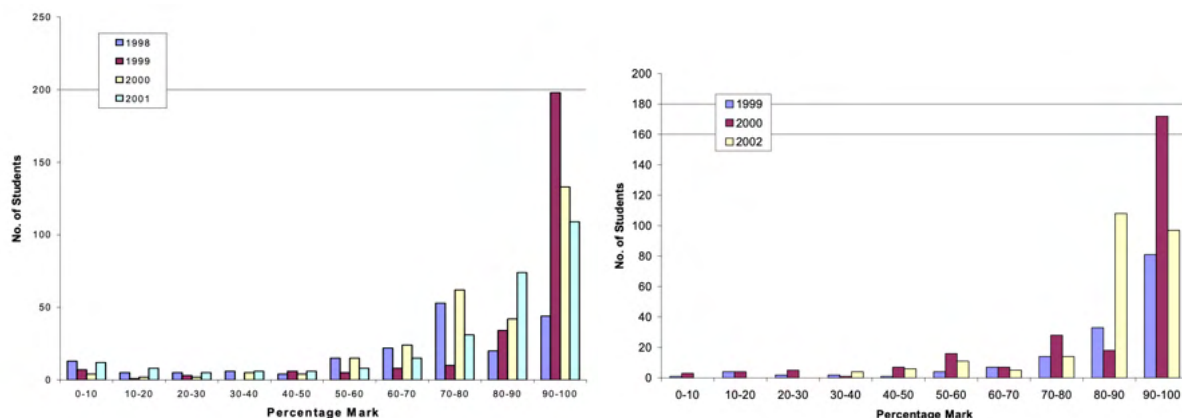


Figure 2: The marks that the students achieved in the first semester programming course (left) and the second semester programming course (right) whilst using ‘Coursemarker’ (Source: Higgins et al., 2005)

With diversity, problems of low motivation, and high dropout rates in mind, Gill and Holton (2006) introduced a self-paced course where lectures were eliminated and the learning material was provided within the course websites. In agreement with Dawson et al. (2018), Gill and Holton (2006) set up a peer support system and allowed teamwork in order to motivate the students to learn; to ensure that students

were not entirely reliant on other members of the team during group work, students were given assignment-related questions to answer during the marking process. Also, to ensure the students were motivated to learn and prevent students from falling too far behind, a series of participation marks were awarded throughout the semester for meetings and filling in progress journals (Gill and Holton, 2006). Utilising chi-square tests, improvements on the pass rates can be observed, though the failure rates seem to have a sizeable difference from semester to semester, with 13% being the latest rate given from 19% (see Table 4) (Gill and Holton, 2006).

*Table 4: Percentage of students who passed, failed and withdrew using the self-paced course in different semesters (source: Gill and Holton, 2006)*

Value	Spring '05	Fall '04	Summer '04	Spring '04	Fall '03
Enrollment	71 [2]	79	34	93	116
Passing	65%	63%	68%	54%	50%
D & F	13%	20%	11%	24%	19%
WD	19%	18%	22%	22%	31%
DWF	35%	37%	32%	46%	50%

Disagreed that lectures should be completely eliminated, an engineering-programming-based study decided to use what is called the 'Self-Practice Online Tool' (SPOT) to aid students in understanding the concepts taught in lectures with the aim of reducing the number of students who fail the course (El-Zein et al., 2009). According to the study, 'SPOT' has three banks of questions: one for testing the understanding of the student on the syntax, one for asking the students to fill in gaps in a half-filled code, and another for telling the students to design and solve problems from nothing. In the feedback survey given by the author of the study, 383 responses were given during the two years of the experiment, of which '75% found ... [SPOT] to be useful or very useful'. The exams and quizzes provided by the course were noted to be more difficult whilst the experiment was occurring as opposed to the years before the experiment. Despite this, one noticeable difference

between the years 2006 to 2007 is that the failure rate dropped from 20% to 15% for the course with clear improvements for certain questions of the exam (El-Zein et al., 2009). According to the evaluation of the students on the course from the El-Zein et al. study (2009), the effectiveness in learning due to the teaching had increased throughout the three years by 17% and 15% respectively, which implied that the students were more willing to learn the content. The study did not achieve the aim of reducing the number of students who failed the course, since the failure rate was 14% in 2005 and 15% in 2007, possibly indicating that the method by Gill and Holton (2006) was superior. On the contrary, the trend implied that had the study by El-Zein et al. (2009) carried on for another year, the aim of the study would have been achieved.

In 2013, Rehberger et al. also conducted an e-learning study with the reasoning that large programming classes that are common at universities cause a lot of organisation problems, which is akin to the reasoning for developing the aforementioned 'Coursemarker'. With the name 'PIT', the types of questions that can be asked are also similar to 'Coursemarker'; questions about developing and submitting code, logic circuits, flowcharts, single and multiple-choice questions as well as text-based questions (Higgins et al., 2005; Rehberger et al., 2013). The main dissimilarity of the 'PIT' tool with the 'Coursemarker' is that the 'PIT' tool has the capability to conduct live polls during lectures, and students can use the tool to program alongside the lecturer during tutorials when the solutions are being developed live (Rehberger et al., 2013). To complete the study, the 'PIT' tool was incorporated into a course, where three tests during the semester are given along with a final exam. The only relevant conclusion that can be drawn from the data is the average score of the course rated by the students improved from 3.8 to 2.3 out

of 6 (Rehberger et al., 2013). Although that is a sign of improvement, there could be a number of factors besides the 'PIT' tool which could have positively affected the course rating. Furthermore, since only one year worth of data was gathered, the study given by Rehberger et al. (2013) suffers the same problem as the 'Extreme Apprenticeship' study, where the conclusion may be premature due to an insufficient amount of data.

Another e-learning method is by the name of 'Pex4Fun' by Tillmann et al. (2013), which is a massive open online course (MOOC). Similar to 'Coursemarker', the motivation behind the 'Pex4Fun' platform was due to feedback given to the students, although the goal was more to do with providing dynamic feedback as changes are made to the solutions by the students. By iteratively providing solutions, students work towards an ideal solution provided by the creator of the question, starting with either a faulty skeleton code or from scratch (Tillmann et al., 2013). As with the other e-learning approaches, 'Pex4Fun' can be scaled up to many students at the same time. Due to the fact that anyone can create the questions or 'coding duels', the type of questions are quite broad, and more can be covered (Tillmann et al., 2013). 'Pex4Fun' can be incorporated into teaching courses due to the flexibility of the platform, and the grading criteria can also be adjusted (Tillmann et al., 2013). For example, 'Pex4Fun' has been used for classroom teaching as well as competitions. Unfortunately, the lack of data from the paper has meant that there is no way to compare the effectiveness of 'Pex4Fun' in comparison to the other aforementioned teaching methods.

Just like pair programming, one of the consequences of shifting into e-learning is that less pressure would be applied on the lecturers, which allows the lecturers to plan other activities for the students; something that 'Coursemarker' aimed for, and

'SPOT' and 'Pex4Fun' indirectly accomplished despite having had other aims (Higgins et al., 2005; El-Zein et al., 2009; Tillmann et al., 2013). What is also agreed on between the three groups of researchers are the attempts at providing quality feedback. However, Buyrukoglu et al. (2016) disagreed that fully automated systems provide high-quality feedback and instead opted for a semi-automatic approach to allow an increase in efficiency while also providing high-quality comments and feedback. Although not a full teaching method, Buyrukoglu et al. (2016) devised a system that allows improvements on assessing code of e-learning approaches by separating student code submissions into segments and codifying each segment. Each individual with the same codified segment would achieve the same feedback, but any segments of code that could not be codified by the system would be manually marked by the lecturer. From subsequent research, Buyrukoglu et al. (2019) concluded with the aid of null hypotheses that the semi-automatic approach is more efficient than the traditional approach of marking.

Overall, the pass rates of e-learning approaches were similar or higher than for the method described by Vihavainen et al. (2011) in the previous subsection, such as 'SPOT' which had a failure rate of 15% compared to an average of 21.75% for the two courses. What is also observed is that many of the e-learning approaches described in the current section also have elements from other categories such as exams or projects. Furthermore, the years of publication of the numerous studies analysed in the current section imply that the e-learning approach of teaching is mature.



### 3.3.3 Projects

Another proposed teaching method is to let the students form groups and work together to accomplish a project (Ortiz et al., 2017).

Cyr et al. (1997) suggested combining both LEGO® bricks and LabVIEW™ software to generate engineering-related experiments. The main reason for the research was to find a low-cost method of teaching engineering concepts such that many schools and universities would be able to replicate such a method. Despite having similar proportions of lectures and laboratory session times as the traditional method, the method by Cyr et al. (1997) is highly focused on the physical electronics components of LEGO bricks and data acquisition, and a final group project is also required based on a given theme. In a subsequent paper, the authors also proposed the use of RoboLab which does not require LabVIEW software in order to lower the cost of the teaching method (Erwin et al., 2000). Although a number of success stories were supplied by the study for primary, secondary, and university level students, there were no statistics to suggest that the proposed method is an improvement on the traditional method of teaching.

In 2011, Esteves et al. investigated the use of Second Life® virtual world to teach students programming. Projects were worked on by pairs and the teaching staff would attend a weekly two-hour meeting in the software with the groups and a face-to-face meeting every month to guide the students. Whilst the study did not provide sufficient evidence that the method is an improvement on the traditional method, conclusions drawn by Esteves et al. (2011) regarding communication issues may be used to improve e-learning approaches of teaching.

Also in 2011, Sun and Sun experimented with a modular programming method, involving splitting a complex project into smaller segments that students work on

every week; the idea behind the approach is that engineering students do not have much experience in programming prior to the course and so students needed to be eased into the subject. Similar to the previous two studies, although the study reported that the score increased by 30%, the number of participants was not disclosed and there is insufficient evidence to believe that the method is an improvement from the traditional approach.

With the goals of maintaining motivation, decreasing the fail rate of the students, and increasing the efficiency of the teaching process, Ortiz et al. (2017) devised a group project centred around robots, where the students were placed into groups of four and had to build and program a robot based on a set of criteria including speed and movement control, obstacle detection and route tracking utilising a list of predefined components. For comparison, the study separated a group of students into two; one of which became the control group which was exposed to the traditional method of teaching and the other became the experimental group with the robots (Ortiz et al., 2017). The study concluded that whilst all the students had similar programming skills prior to the course, a null hypothesis was conducted which proved that the students from the experimental group had a much higher mark; the average marks for each of the given six exercises were at least 7 marks greater for the experimental group than the control group, with the gap increasing as further exercises were supplied (see Table 5). Furthermore, the questionnaire completed by students revealed that the motivation of students was maintained, completing the majority of the aims that the study initially set (Ortiz et al., 2017). Due to the employment of t-test for null hypothesis and control groups, the argument that the proposed teaching method is an improvement from the traditional teaching

method is strong, but the group of participants, which were 60 students, are smaller than some of the aforementioned studies (Ortiz et al., 2017).

*Table 5: Average results of each exercise in the robot group project (source: Ortiz et al., 2017)*

Assessment	Control group		Experim. group	
	Mean	Std. Dv.	Mean	Std. Dv.
EX. 1	80.1	13.85	87.6	10.22
EX. 2	75.4	11.82	86.5	9.67
EX. 3	72.5	10.84	88.4	8.92
EX. 4	70.2	9.13	87.3	8.29
EX. 5	64.1	8.85	84.3	7.71
EX. 6	60.2	8.68	82.8	7.15

In a bid to improve the teaching of programming concepts to students, Tsai (2019) experimented with visual programming languages (VPL) for university students since VPL removed certain barriers such as syntax errors, which made VPL attractive to secondary school programming teachers. 180 students participated in the study with 84 students being the control group and learning programming through the traditional approach described by Blumenstein (2002). Students of the experimental group began by learning basic programming concepts on App Inventor 2 (AI2) for the first two-thirds of the course; the students were then allowed to choose a project to complete based on the preference of the students (Tsai, 2019). Based on the ANCOVA tests conducted by the researcher, there were significant differences in the mean marks achieved by the experimental group in comparison to the control group, meaning that utilising visual programming languages had a positive impact on learning programming concepts. Since Guo (2018) suggested visual learning to combat the language barrier that non-native English speakers have, the method by Tsai (2019) would be a suitable approach of combating such a barrier.

### **3.3.4 Competitions**

Alike Cyr et al. (1997), Wang (2001) evaluated the use of LEGO bricks and LabVIEW software, where the course was more focused on a “hands-on” approach of teaching than the traditional method and so there were no final examinations (Wang, 2001). With elements of pair programming, the students needed to complete ‘approximately 10 LEGO based assignments of increasing difficulty’ throughout the semester, with an added twist and emphasis on creative solutions to stimulate competition between teams (Wang, 2001). From the journal, the year the method for programming was introduced caused a dip in the success rate of the students in the course. However, the success rate increased back to levels before the reform after one year, and the dropout rate of students also steadily decreased (Wang, 2001).

### **3.3.5 Puzzles**

Another branch of research into teaching programming is puzzles; the focus of these types of research are usually for increasing the motivation of the students (Parsons and Haden, 2006; Merrick, 2010).

Termed 'Parson's Programming Puzzles' (PPP), the tool created by Parsons and Haden (2006) used only drag-and-drop puzzles to teach students programming. According to the study, lines of code were mixed in with incorrect statements and the job of the students was to rearrange the lines of code to form a program based on the given specification (see Figure 3). Furthermore, the study stated that the tool is an automated tool, so the property of allowing instant feedback by e-learning tools is also shared by the 'PPP'. 82% of the students agreed that the tool was useful as a learning and revision tool; however, the high percentage of satisfaction

with the proposed tool is due to the low number of participants of the study, which consisted of 17 people (Parsons and Haden, 2006). In addition, the fact that the tool only consists of one type of puzzles suggests the tool is somewhat limited in usefulness and could be improved on by expanding the types of puzzles available, of which the viewpoints are shared by 5 of the 17 students (Parsons and Haden, 2006).

**Lab 1**  
**Syntax practise**

Drag the statements on the right into the correct order. Do not use the incorrect statements. Should produce four lines of text- read them to get them in the correct order.

1	<code>WriteLn("This is the first line");</code>
2	<code>WriteLn("Don't use a line that's not correct as line 2");</code>
3	<code>WriteLn(5.67:2);</code>
4	<code>Begin InitOurCrt;</code>
5	<code>WriteLn("Don't use a line that's not correct as line 2");</code>
6	<code>WriteLn(5.67234:5:2);</code>
7	<code>Program Pick_the_correct_lines; {\$APPTYPE CONSOLE}</code>
8	<code>End.</code>
	<code>End;</code>
	<code>uses SysUtils, OurCrt;</code>
	<code>WriteLn("The line below should show the number 5.67");</code>

Figure 3: An example puzzle of the 'PPP' software (Source: Parsons and Haden, 2006)

In order to increase the problem-solving skills of engineering students, Merrick (2010) also suggested a puzzle-based approach to teaching. The approach simply involved the traditional approach with a twist of having all the examples be puzzle-related. To ensure motivation for the course, students were also required to solve a puzzle chosen by the student; the code used to solve the puzzle must satisfy certain conditions dictated by the marking scheme (Merrick, 2010). Based on the results given by Merrick (2010), the adoption of the method had an all-around

positive effect according to the students in comparison to the traditional method, though the results were only gathered for one year after the method had been adopted and so the results may have occurred by chance.

Unlike any of the previous studies, Figueiredo and García-Peñalvo (2019) proposed the use of several activities to increase the computational thinking of students. For example, the research involved origami and punched hole activities (see Figure 4), where the student had to identify the shape of the unfolded piece of paper after the paper has been folded in various ways and hole-punched. Besides that, students were given and were asked to give numerous random instructions in order to improve reasoning abilities, as well as map design activities in order to increase the planning skills of the students (Figueiredo and García-Peñalvo, 2019). Though the number of participants was small for the study at 49 people, the preliminary results indicated that the ‘Follow and Give Instructions’ activities have a strong correlation with success in the programming course (Figueiredo and García-Peñalvo, 2019).

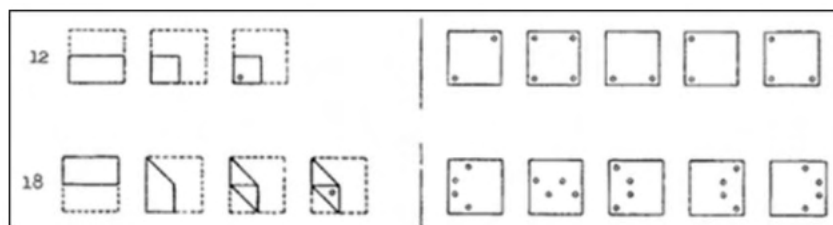


Figure 4: An example of the punched hole exercise (Source: Figueiredo and García-Peñalvo, 2019)

An alternative method by López-Pernas et al. (2019) consisted of a programming escape room in order to boost motivation and decrease the failure rate. The escape room included a “bomb” that all the participating students had to defuse, using a program written by the coordinators of the course; the program contained a number of bugs that the students had to debug, and a series of logical tasks were needed to obtain the correct commands for using the program (López-Pernas, 2019). Of the 124 students who participated, 84 students gave a mean score of 4.2 out of 5 when

asked if the escape room was fun, which suggested that the activity was motivating (López-Pernas, 2019). However, 82 of the students gave a mean score of 3.4 when asked if the activity improved knowledge of the content of the course, which suggests that although the escape room was motivating, utilising the activity as a method of teaching programming is unsuitable (López-Pernas, 2019).

### **3.4 Secondary School Level Teaching Method**

#### **3.4.1 Modifications to Traditional Method**

In 2019, a study by Sentance et al. described the use of the 'PRIMM' approach in order to teach secondary school students programming, with the research taking place due to wanting to find a method that could alleviate some of the identified problems of learning programming. Just like McDowell et al. (2002) and Nagappan et al. (2003), pair programming was identified as important and was incorporated into the method; for the students, the method involved having to predict what an existing piece of code does, followed by a demonstration by the teacher. Analysis of the code would then be carried out, followed by the students modifying the code for different given exercises (Sentance et al., 2019); at the end, students were told to use the same structure and create an entirely different piece of code. Utilising such a method would allow the students to build up confidence and skill (Lee et al., 2011). Using one group of students as the experimental group and another as the control group in a 673 student study, Sentance et al. (2019) concluded with the aid of the Mann-Whitney U test that the 'PRIMM' method had an undeniable positive effect on the scores, and the responses received from teachers were positive.

### **3.4.2 Scratch**

One of the more popular teaching methods of programming among secondary school teachers is the use of Scratch, evident by the numerous research journals on the topic.

For Coravu et al. (2015), the main focus for the research was to observe how useful Scratch as a tool would be for teaching programming since visual programming languages such as Scratch allows students to bypass having to learn the syntax and is easier to debug. Coravu et al. (2015) also identified that Scratch allow students to develop creative thinking skills and is considered enjoyable. Separating 210 students into two equal groups with one learning programming concepts from Scratch and another from C, the study concluded that students who learnt the topics on Scratch obtained better results and motivation than the other group, though due to the lack of statistics the results cannot be verified. However, a separate study by Yildiz Durak (2018) supported the use of Scratch for teaching programming after conducting an experiment with 62 secondary school students. With the aim of effectively teaching programming concepts, Yildiz Durak (2018) separated the students into two groups with one group learning programming via the conventional method and another group being taught via a digital story project; the students were asked to create a digital story entirely from scratch on Scratch, including the storyboard, vocals, and images. A pretest was conducted by the researcher beforehand to ensure that both of the groups were at the same level in terms of programming achievements (see Table 6). With a mean score of 81.56 for the group who learnt programming via the digital story approach and 73.17 for the conventional approach, Yildiz Durak (2018) performed ANCOVA tests and found that there were significant differences in the participation levels, the learning of the



concepts, and the grades given to the students, all favouring the digital story approach. Although no indication was given for how many students received subpar grades, increasing in the participation levels suggests that there was an increase in motivation in learning programming, which is considered to be a major barrier in many of the aforementioned studies. Two problems can be observed from the study: similar to the study conducted by Ortiz et al. (2017), the low number of participants in the study compared to some of the others mentioned suggests that the results can still vary, and further research would have to be carried out. Moreover, whilst the participants were secondary school students, the participants were of the younger years of secondary school, meaning that the comparison of the success rates of the study conducted by Yildiz Durak (2018) with the other aforementioned methods may be unfair, as the concepts that students had to learn whilst using the other teaching methods may have been more advanced.

*Table 6: Mean programming achievements before and after the experiment (source: Yildiz Durak, 2018)*

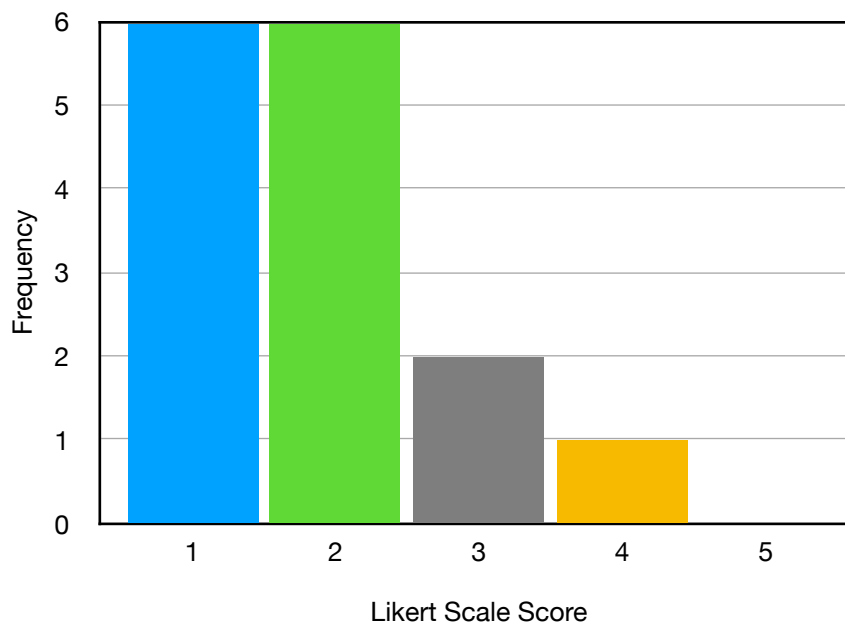
Groups	N	Pretest		Posttest	
		Mean	SD	Mean	SD
G1	32	40.16	17.34	81.56	15.68
G2	30	40.33	21.65	73.17	13.42

Another study relating to Scratch was done by Hermans and Aivaloglou (2017), who decided to combine Scratch and MOOC, which is a model similar to ‘Pex4Fun’ by Tillmann et al. (2013). According to the study, materials such as ‘videos, quizzes and forum interactions’ were provided, and the students had to program a game each week, assessed with 2 exams over the duration of the course. Due to the nature of MOOC, of the 2220 students who used the method, only 181 completed the course (Hermans and Aivaloglou, 2017); the fact that the course is a MOOC also meant the

data cannot be compared with the other courses described here. What is of interest however is that the mean grades of students who did answers questions had the lowest for programming concepts questions such as the use of comparison operators with 0.63 and the second-highest for debugging questions with a value of 0.85 out of 1, which supported the argument by Coravu et al. (2015) that Scratch allows student ease into the idea of debugging (Hermans and Aivaloglou, 2017).

## 4. Results and Discussion

Using the methodology detailed in section 2.2, a survey (see Appendix B) was sent out electronically to secondary school programming teachers across England and Wales, of which 15 responded. For Figures 5, 6, and 9, the Likert scale associated begins with strongly agree (1) to strongly disagree (5). For Figure 8, the scale begins with very easy (1) to very difficult (5).



*Figure 5: "Programming skills are important to have in general, regardless of profession"*

For question 1 (see Figure 5) there was an overwhelmingly positive response regarding the usefulness of programming as a skill in general where only one teacher responded in the negative, with a mean value of 1.87 using the Likert scale score. The purpose of this question was to see if teachers of programming would consider what is being taught useful; teachers who do not consider what is being taught to be useful would be less motivated to teach the subject, which would become a barrier for the students to learn to program. Without having such motivation and passion, teaching students would be more akin to 'enforcement and

obedience' according to Fried (1995, cited in Day, 2004). From the results gathered, the implication is that the lack of passion by teachers is not a barrier to learning programming. On the other hand, the fact that all the teachers who participated in the survey are programming teachers may have skewed the results more toward the positive than if the teachers were from all different kinds of disciplines.

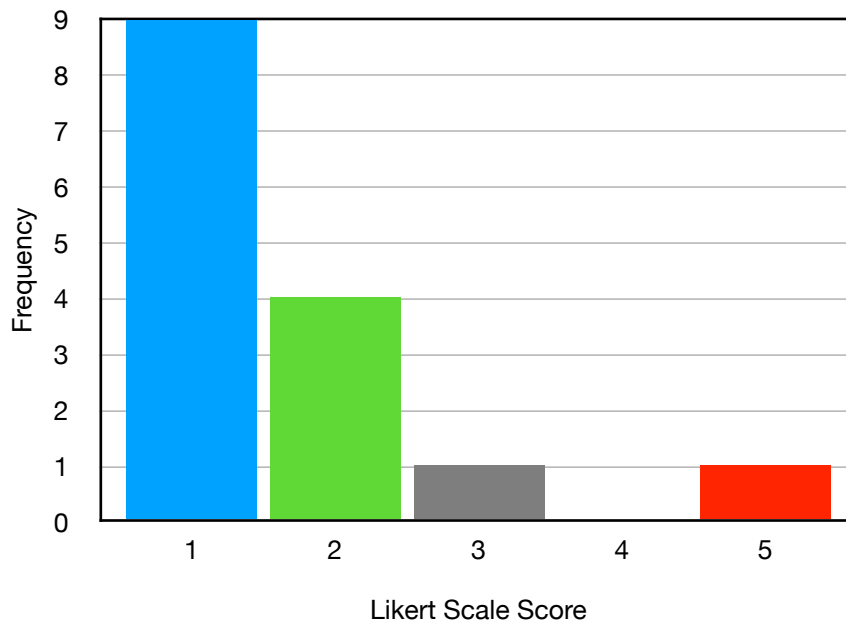
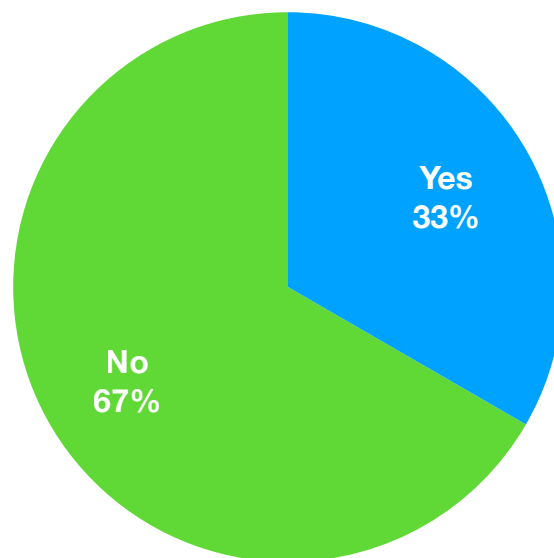


Figure 6: "Programming skills are integral to engineers"

The next question asked was the importance of programming according to the teachers (see Figure 6). Again, the results were overwhelmingly positive, with only one teacher who believed programming is not important to engineers, resulting in a mean value of 1.67. This is in agreement with Sun and Sun (2011) and Nikolic et al. (2018) who stated that programming is essential for engineers. The reasoning behind question 2 was if teachers disagreed that engineers require programming skills, teachers would not recommend learning programming to any students aspiring to become engineers, instead steering the students to more traditional subjects such as mathematics and physics. The resulting effect would be a barrier to learning to program as the students would not consider the skill to be important

for an engineering role. Similar to question 1, having only programming teachers fill the survey may have caused a bias towards the positive side of the scale. Additionally, one of the participants pointed out that different types of engineering may cause the answers to be different, which was not specified within the question. All the responses for question 3 can be viewed in Appendix C. The main responses by the participants for this question relate learning programming to learning problem-solving skills, one that is widely known as being vital to mechanical engineers. Several responses also noted the usefulness of programming for mechanical engineers due to the large use of models and simulations, many of which would benefit if the user understands programming techniques. One particular response mentioned that learning such a skill would potentially open up new job prospects for the student. To combat the lack of motivation in learning programming, which has been identified frequently in section 3, one may potentially use these answers as incentives.



*Figure 7: “Would you consider the students to be well-informed about what engineering entails when considering their future career options?”*

Figure 7 displays the results for question 4. The reasoning for the question is because the lack of knowledge of what engineering is may cause some students to not consider the subject as a possible future career despite having an interest in the field, which would be a barrier to engineering, and a barrier to engineering programming. Similar to the reasoning for question 2, students who are interested in engineering but not well-informed would not know that programming is important to engineers (Sun and Sun, 2011; Nikolic et al., 2018). When asked if the students know what engineering entails, 67% of teachers reported that the students do not know. This is in agreement with the study by Hirsch et al. (2007) which revealed that 62% of secondary school students did not know what the work for any type of engineering consists of despite the vast majority of students claiming beforehand that the student understood what engineers do. This suggests that despite only having teachers respond to the survey, the data gathered is likely more accurate than if only students responded to the survey as the teachers can predict what the students know and do not know better compared to the students themselves.

Question 5 (see Appendix C) was asked in order to identify possible barriers that students may have and is worded in such a way in order to not restrict the answers participants give. The aforementioned barriers such as low confidence from Lui et al. (2004) and lack of motivation were mentioned by several participants. In addition, a lack of interest in programming was also flagged as a barrier, which can be attributed to the subject being poorly taught previously or the lack of role models to look up to due to programming being a relatively new field. Some of the other possible barriers include the regarded difficulty of the subject and the lack of awareness.

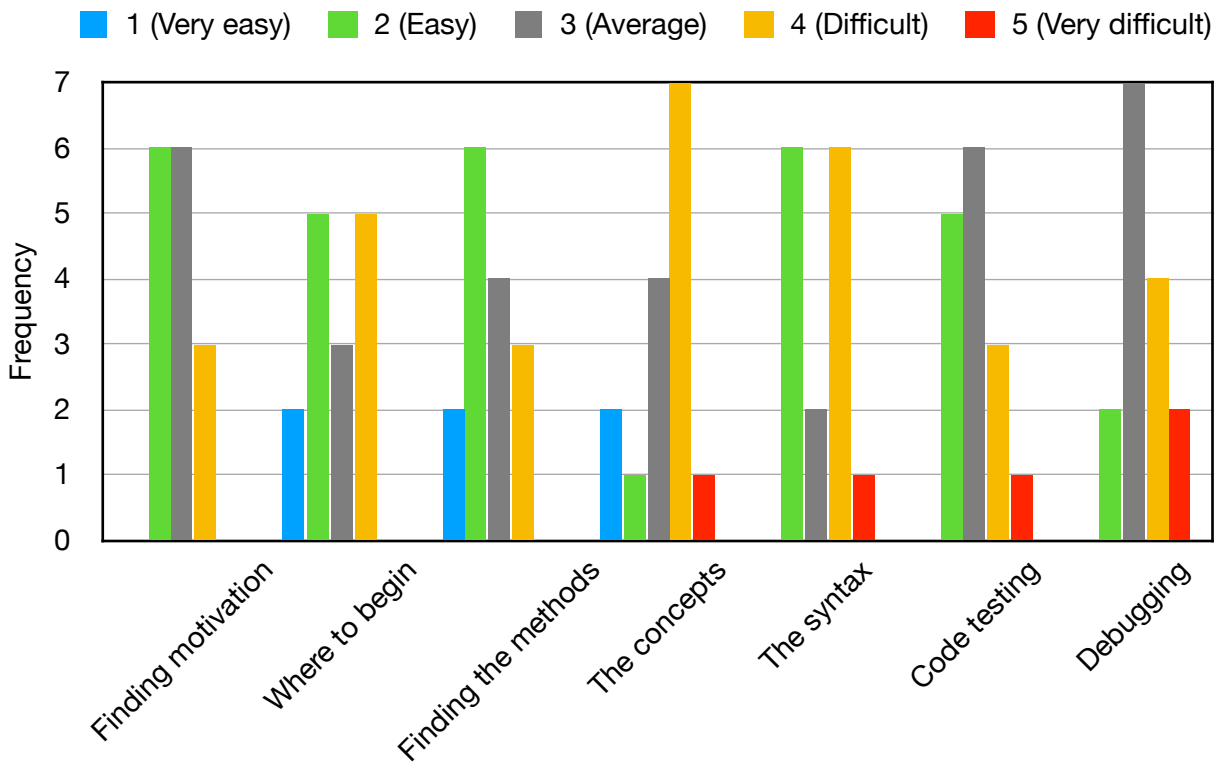


Figure 8: Whether teachers believe the given aspects are difficult for students

The results for question 6a are displayed in Figure 8. The purpose of question 6a was to find the most pressing barriers from the given examples. From left to right of the figure, the mean values for each of the examples are 2.80, 2.73, 2.53, 3.27, 3.13, 3.00, and 3.40 respectively. Comparing the mean values, the biggest barrier according to the participants is code debugging followed by understanding the concepts and syntax, whilst the easiest is finding the method to write programs followed by finding where to begin the programming process. The results shown in Figure 8 are in agreement with Coravu et al. (2015) and Sentance et al. (2019), who both stated that learning syntax is one of the bigger barriers to learning programming. Surprisingly, the mean values show that attempting to find the motivation to learn programming is considered to be somewhat easy according to the participants despite being one of the major focuses when attempting to find a more successful teaching method. When converted to a 7-point Likert scale from

the 5-point scale, the mean values for the syntax data and the debugging data become 4.20 and 4.60 respectively (Lewis and Sauro, 2020). Both sets of data then underwent the Kolmogorov-Smirnov test with p-values of 0.19298 and 0.16968 respectively, indicating that the data sets are sufficiently normally distributed (Kent State University, 2021; Stangroom, 2021); this was to ensure that a t-test could be performed on the data alongside the data gathered by Piteira and Costa (2013) to see if the barriers are similar for different countries. The t-values (see Appendix C) for the syntax and debugging sets of data were found to be 1.95 and 1.68 respectively (Kent State University, 2021). Since the syntax data set has a p-value of 1.725, the null hypothesis (see Appendix C) was rejected, meaning the mean values were not the same. On the other hand, the p-value for the debugging set of data 1.714 which means the hypothesis was not rejected. Since the mean values were likely not the same for the syntax data set, the implication is that the aforementioned barrier is prioritised differently for professionals in different countries; a problem may arise where research into the particular barrier is done at an inconsistent rate or is simply neglected by the professionals of particular countries. Whilst the null hypothesis of the debugging data set was not rejected, indicating a possibility that no relationship exists, the null hypothesis also implies that greater sample size is required.

Similar to question 5, question 6b was designed to identify possible barriers but was more orientated towards ones that students encounter during the process of learning. Several of the barriers in question 5 were reiterated in question 6b (see Appendix C). Nevertheless, the one which stood out was the difficulty in using pseudocode, whose main purpose when used correctly is to assist in planning and visualising programs. Social aspects were also flagged as barriers, including access



to equipment and resources, especially for secondary schools with less funding or students who are disadvantaged, as well as gender and racial stereotypes; a limited amount of research has been done to minimise the social problems, such as by Kamin (2007), though more research would need to be conducted in the field. A potential barrier that a participant identified is that some educators would focus on how to use functions in particular programming languages as opposed to the general concepts of programming.

■ 1 (Strongly Agree) ■ 2 (Agree) ■ 3 (Don't know) ■ 4 (Disagree) ■ 5 (Strongly Disagree)

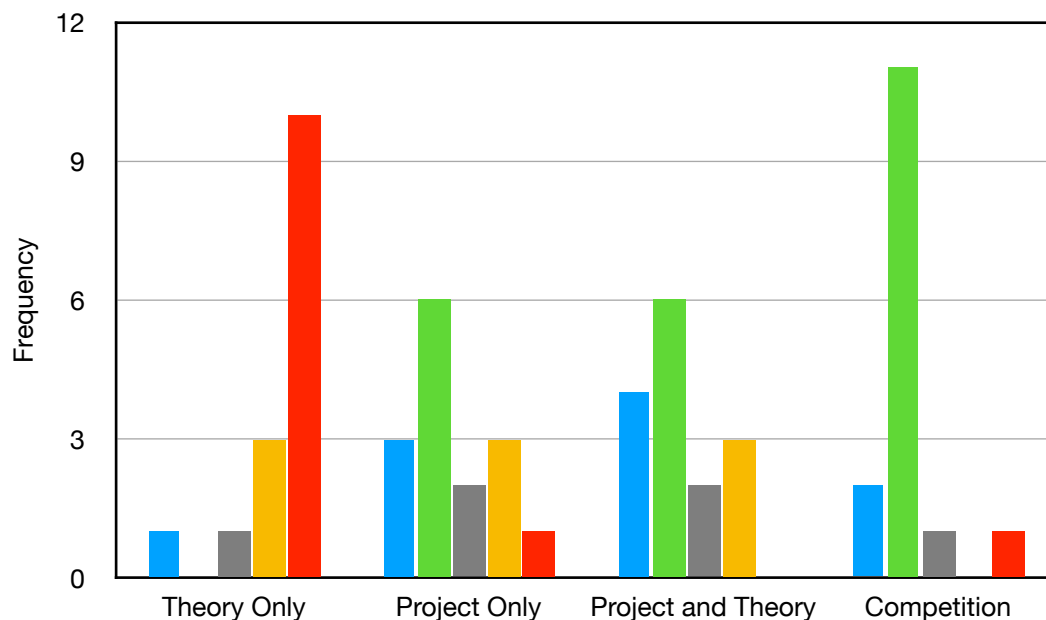


Figure 9: Whether teachers agree that the given method should be used to teach programming

For question 7a, participants were introduced to four teaching methods from section 3 and were asked if the teaching methods would be effective based on experience (see Figure 9). From left to right, the mean values based on the Likert scale were: 4.40, 2.53, 2.27, and 2.13. Unsurprisingly, the vast majority of participants believed that simply learning the theory is not a good strategy for teaching programming, which is a teaching method used to teach more traditional courses. Project and theory are akin to the traditional method described by

Blumenstein (2002) in section 3.2, and project only is based on the methods in section 3.3.3. Interestingly, based on the mean values of the two, participants of the survey believe that project and theory is a more suitable method of teaching programming, whilst researchers from section 3.3.3 such as Ortiz et al. (2017) found evidence that project-based teaching methods are better than the traditional method; one of the implications is that teachers are unable to catch up with current research. Alternatively, the observation suggests that secondary school teachers are unaware of the research since the targets for the journals are students at university rather than for primary or secondary school students. The lowest mean value for the data sets in Figure 9 was for competitions, which according to Wang (2001) was a method capable of decreasing the dropout rate while also maintaining the success rate, hence a safe strategy to use.

Finally, question 7b was asked to allow participants to introduce teaching methods that teachers may consider to be successful when it came to teaching programming. Some suggestions by participants include the use of physical devices such as Raspberry Pi, Edbot, and drones at a young age to increase relevance and allow students to physically see the results when the code is complete. One particular response suggested the use of unplugged activities similar to the method of Figueiredo and García-Peñalvo (2019). Another response suggested the use of simple programs that students can edit; students would need to know how each of the functions works before any improvements can be made. Several of the responses given were more towards improving the motivation and interest in the students and can be viewed in Appendix C.

Overall, what has been discovered is that programming is considered to be an important skill to have, though not necessarily a skill that students are aware of that

engineers need. The t-test is evidence of the fact that although barriers may be similar for different countries, the order of priority when attempting to solve the problem is different for different countries, and some barriers may simply be regarded as not sufficiently important to be looked into. Some of the barriers that have been identified through the survey include code debugging, learning the syntax, use of pseudocode, access to equipment and facilities, racism, and sexism. For code debugging, analysis of the research from section 3.4.2 had indicated that visual programming languages have the ability to ease students into the topic. In terms of teaching methods, competitions have been identified as the preferred method of programming teaching by the participants out of four of the more common methods of teaching, with project-based teaching using physical devices also being a popular option.

In the study, a number of limitations were identified. Firstly, the lack of participants for the survey has meant that the sample is less likely to represent the thoughts of the population. In addition, the participants being programming teachers in secondary schools has meant that the participants may have been biased when attempting to answer questions 1 and 2. As identified by one of the participants, had the question been worded slightly differently for question 2, the answers would likely have been different as well.

## **5. Cheatsheets and Software**

In line with the second aim set out at the start of the project, cheatsheets and software were generated and designed to assist students in learning how to program.

The software is programmed in Python and was inspired by the e-learning approaches evaluated in section 3.3.2, particularly 'SPOT' and 'PIT' where a broad range of questions was made and students can attempt a randomised question every time. E-learning approaches are also currently one of the more mature methods of teaching programming, evident by the number of studies conducted in the field compared to the other methods mentioned. As shown below, programming exercises designed to combat the more pressing barriers identified in question 6a of the survey were created.

## **5.1 Cheatsheets**

Since syntax was a major barrier according to results from the survey, cheatsheets were created to assist the students when programming, where the students can quickly check the syntax as well as check the purpose of some of the common functions used in a program. Two cheatsheets were created; one for Python and one for MATLAB (see Appendix D). Common variables of the programming languages are introduced at the top of the cheatsheets, followed by quick definitions of the listed common functions as well as syntax and parameters. Under the list of functions are extra pieces of information that students should be informed about prior to using the functions. Finally, references to notes and books that the cheatsheets were adapted from were included at the bottom of the cheatsheets so that students may do some further reading if interested.

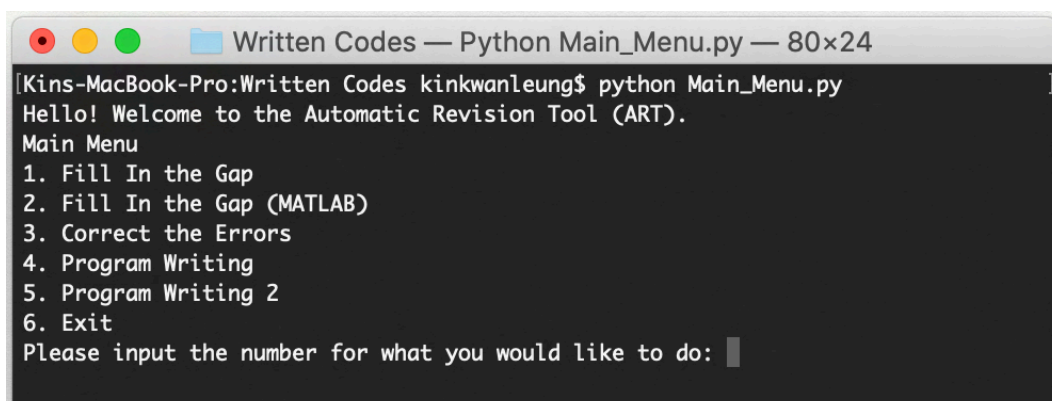
## **5.2 MATLAB Engine API**

To write the code for the software, research into Python was needed in order to find suitable functions such as Threading, Time and Random modules (Python Software

Foundation, 2021a; 2021b; 2021c); how the modules were used can be seen in Appendix F. An API by the name of MATLAB Engine was utilised in order to allow the execution of MATLAB functions in Python (Mathworks Inc., 2021). The use of the API allowed the software to evaluate answers given in both Python and MATLAB code, increasing the flexibility for teachers who use the software and allows the teachers to focus more on the concepts of programming rather than on any particular programming language, which was suggested as a barrier by a participant of the survey.

### 5.3 Software

The software program for the project is named Automatic Revision Tool (ART) and can be accessed by the command-line interface, which outputs the main menu (see Figure 10 and Appendix E for code). From the main menu, a number of programmed exercises can be accessed for revision via inputting the number associated. The code for each of the exercises is stored in a module by the name Questions.py (see Appendix F).

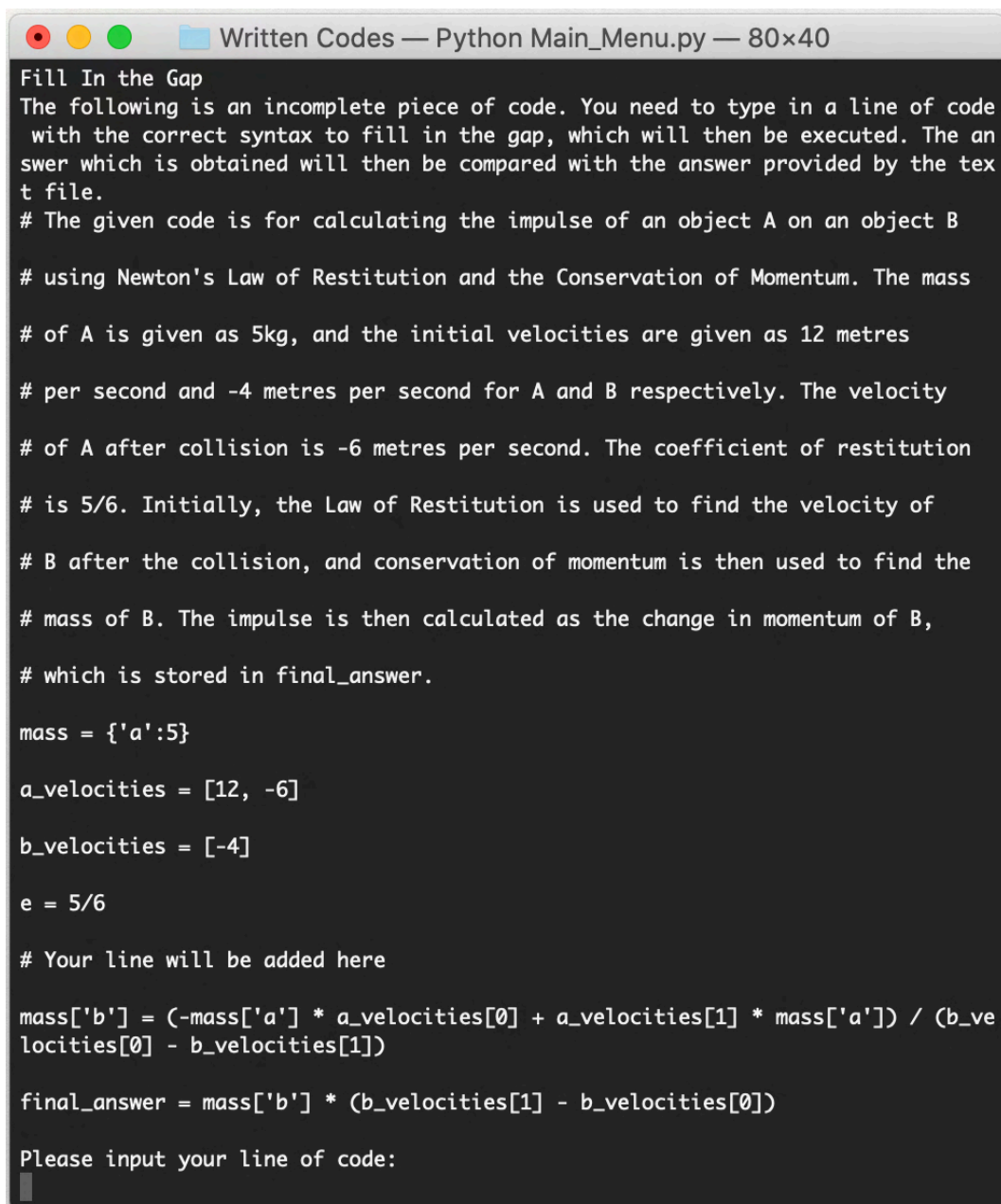


```
[Kins-MacBook-Pro:Written Codes kinkwanleung$ python Main_Menu.py
Hello! Welcome to the Automatic Revision Tool (ART).
Main Menu
1. Fill In the Gap
2. Fill In the Gap (MATLAB)
3. Correct the Errors
4. Program Writing
5. Program Writing 2
6. Exit
Please input the number for what you would like to do: █
```

Figure 10: The main menu of ART displayed on the command-line interface

The first two options on the main menu access an exercise that requires the user to input a line of code, one for Python and one for MATLAB. The purpose of the exercise is to assist the students in remembering the syntax of programming

languages, which was found to be a major barrier to learning programming in the survey. The program reads a text file with an incomplete pre-written set of code, the numerical answer if the code was executed with the correct statements, and the line that needs to be filled by the user; an example of which is given in Appendix G. The software outputs a description of what the exercise entails as well as the incomplete set of code with its specification (see Figure 11). As an example, a question about the conservation of momentum is given as mechanics is a relevant topic of both mechanical engineering and A-Level Further Mathematics (WJEC, 2019d).



```
Fill In the Gap
The following is an incomplete piece of code. You need to type in a line of code
with the correct syntax to fill in the gap, which will then be executed. The an
swer which is obtained will then be compared with the answer provided by the tex
t file.
# The given code is for calculating the impulse of an object A on an object B
# using Newton's Law of Restitution and the Conservation of Momentum. The mass
# of A is given as 5kg, and the initial velocities are given as 12 metres
# per second and -4 metres per second for A and B respectively. The velocity
# of A after collision is -6 metres per second. The coefficient of restitution
# is 5/6. Initially, the Law of Restitution is used to find the velocity of
# B after the collision, and conservation of momentum is then used to find the
# mass of B. The impulse is then calculated as the change in momentum of B,
# which is stored in final_answer.

mass = {'a':5}

a_velocities = [12, -6]

b_velocities = [-4]

e = 5/6

# Your line will be added here

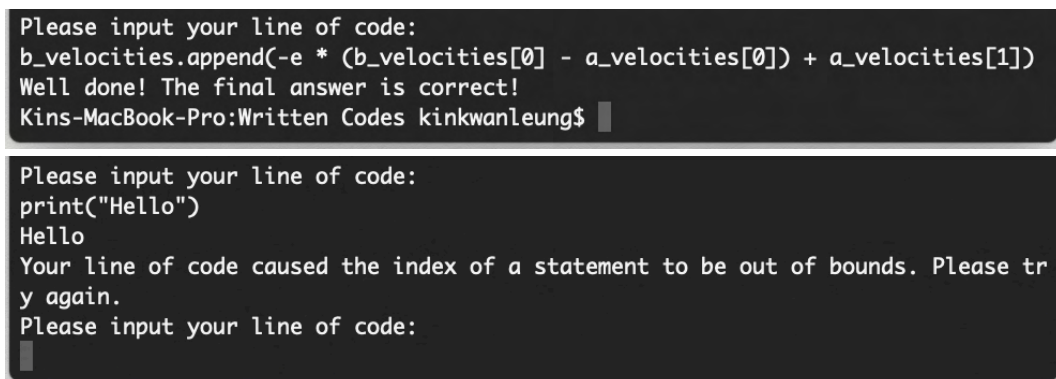
mass['b'] = (-mass['a'] * a_velocities[0] + a_velocities[1] * mass['a']) / (b_ve
locities[0] - b_velocities[1])

final_answer = mass['b'] * (b_velocities[1] - b_velocities[0])

Please input your line of code:
```

Figure 11: The command-line interface after choosing option 1 (Fill In the Gap)

A prompt would appear for when the line of code needs to be entered, and the software automatically injects the line of code provided by the user into the incomplete set of code; once completed ART attempts to execute the code and retrieve a numerical value, which is then compared to the aforementioned numerical answer. If the numerical value was equal, then a message would appear to congratulate the user and terminate the software (see Figure 12a). Conversely, if the answer was incorrect then a message would appear to clarify the problem encountered when attempting to execute the code and compare the expected answer with the actual answer; the program would then prompt the user to try again (see Figure 12b).



```
Please input your line of code:
b_velocities.append(-e * (b_velocities[0] - a_velocities[0]) + a_velocities[1])
Well done! The final answer is correct!
Kins-MacBook-Pro:Written Codes kinkwanleung$
```

```
Please input your line of code:
print("Hello")
Hello
Your line of code caused the index of a statement to be out of bounds. Please try again.
Please input your line of code:

```

*Figures 12a (top), 12b (bottom): The command-line interface after entering the correct line of code (top) and after entering an incorrect line of code (bottom) for Fill In the Gap exercise*

One of the special features of Fill In the Gap and Fill In the Gap (MATLAB) is that with different text files, teachers can add different questions as long as the text files adhere to a certain syntax, allowing flexibility for the teachers (see Appendix G).

For option 3 from the main menu (see Figure 10), a similar process to options 1 and 2 occur, where the software reads a text file and displays on the command-line prompt the description of the exercise as well as the piece of code that the exercise would use. Nevertheless, the main difference of option 3 is that the piece of code which is displayed on the command-line prompt is faulty and does not run, and was

designed as a result of identifying debugging as one of the biggest barriers to learning programming. Instead, the software requires the user to edit the piece of code on a separate text editor until all the errors and exceptions are fixed. Once the piece of code runs, an integer will be given which the user enters into the command-line prompt of the software; if the answer is correct, a message of congratulation would appear. Otherwise, the software would prompt the user to try again and enter a different value (see Figure 13).

```
Please input the final answer: 1
Unfortunately, your answer was not correct. Please try again.
Please input the final answer: 9744803452
Well done! The final answer is correct!
Kins-MacBook-Pro:Written Codes kinkwanleung$
```

Figure 13: The command-line interface after entering an incorrect followed by a correct numerical answer for the Correct the Error exercise

A separate function was not required for the MATLAB version of the exercise since similar to Fill In the Gap exercises, different text files with different programming languages can be used as long as the syntax is adhered to, which is similar to the syntax shown in Appendix G though without the ">gap<" line of code.

```
Python Program Writing
-----
| M |--> F,P
-----
Consider a two locomotive train. Let F and P be the two forces generated by the
engines of the two locomotives in Newtons and M be the combined mass of the loco
motives in kilograms. There will be three different values for each of F, P and
M. Write a program that can evaluate all the possible values of the acceleration
in metres per second squared. You will have 20 seconds to input all the numbers
in and 15 more seconds to answer the three randomised questions at the end.
Please enter "Continue" when you are ready:

Python Program Writing 2
A car travelled s metres in a straight line at an initial velocity of u metres p
er second in t seconds. There will be three different values for each of s, u an
d t. Write a program that can evaluate all the possible values of the accelerati
on in metres per second squared. You will have 15 seconds to input all the numbe
rs in and 15 more seconds to answer the three randomised questions at the end.
Please enter "Continue" when you are ready:
```

Figure 14: The descriptions for both of the Program Writing exercises



Finally, choosing options 4 and 5 shown in the main menu (see Figure 10) causes the software to display a description of the exercise, both of which are displayed in Figure 14; the descriptions were specifically chosen due to the relevance of the problems to both mechanical engineering and A-Level Mathematics and Further Mathematics. In addition, choosing a topic that is related to mechanical engineering may inspire passion in programming as long as the student is interested in the subject (WJEC, 2019d; 2019e).

The software requires the user to work out and write a script that solves the problem, assisting in overcoming the vast majority of the barriers described by question 6a of the survey. For each of the exercise, 9 values which are determined dynamically would be outputted by the software, and in both instances, the script written by the user must be able to take in all the values given and calculate all possible values of the acceleration. Once the allocated time has elapsed, 3 questions based on random combinations of the 9 values would be asked, and the user must answer all 3 of the questions in order for the software to mark the scores. Otherwise, the software would abort after the 15 seconds have elapsed (see Figure 15a). Should any of the answers be incorrect, the software would output the correct answers so that the user can review and debug the script (see Figure 15b). If the answers were all correct then a message of congratulation would be printed on the command-line interface (see Figure 15c).

To ensure that the Program Writing exercises were independent of programming languages, the answers calculated by the software rounds to the nearest two decimal places to make sure that other programming languages with less accurate mathematical functions can still utilise the software.

```
What is the acceleration when s is 39.35, u is 37.8 and t is 21.27?
Time is up! Press the "Enter" key to continue

Unfortunately, you did not answer the questions on time.
Kins-MacBook-Pro:Written Codes kinkwanleung$ █

What is the acceleration when s is 36.31, u is 19.59 and t is 25.34? 50
What is the acceleration when s is 46.01, u is 41.41 and t is 33.1? 10
What is the acceleration when s is 19.04, u is 19.59 and t is 33.1? 4
Marking... Please wait...
Your marks are 0 out of 3
The correct answer for question 1 was -1.43 whilst you inputted 50.0
The correct answer for question 2 was -2.42 whilst you inputted 10.0
The correct answer for question 3 was -1.15 whilst you inputted 4.0
Kins-MacBook-Pro:Written Codes kinkwanleung$ █

What is the acceleration when s is 19.96, u is 12.53 and t is 10.96? -1.95
What is the acceleration when s is 46.29, u is 12.53 and t is 10.96? -1.52
What is the acceleration when s is 11.99, u is 15.29 and t is 10.96? -2.59
Marking... Please wait...
Your marks are 3 out of 3
Well done!
Kins-MacBook-Pro:Written Codes kinkwanleung$ █
```

Figures 15a (top), 15b (centre) and 15c (bottom): Command-line interface if the answers were not provided (top), incorrect (centre) and correct (bottom) for the Program Writing exercise

The software was tested for bugs using a series of test data, and the resulting outcome was noted and can be viewed in Appendix H; The test data was designed to test all the error cases in the code as well as undesired inputs. Several bugs were identified as a result. Examples include:

- The software not comparing numerical values and subsequently crashes
- Inputting values at particular points during the Program Writing exercise causes the software to bypass the questions being asked
- Index issues as MATLAB begin an index with the numerical value 1, whereas Python begins an index with 0
- Functions causing the type of the variables to change

Once the bugs were identified, fixes were made immediately.

## 6. Conclusions

As stated in the introduction, the purpose of the study was to identify successful teaching methods and barriers that students of programming may face, particularly for secondary school engineering students. Based on the pedagogical literature analysed in the literature review as well as the data found from the various examination boards in the UK, the biggest problem that is observed is the high rates of dropout and failure (Vihavainen et al., 2011; Elnagar and Ali, 2012). Further analysis of the problem revealed that motivation is a significant factor for the students when deciding whether to drop out (Yacob and Saman, 2012). Barriers such as a lack of problem-solving skills, which was further reinforced by the survey, all contribute to the problem (Merrick, 2010; Yacob and Saman, 2012). Other results from the survey implied that the lack of motivation could be due to the lack of role models as well as a lack of interest in the topic. As well, the lack of knowledge that programming is necessary for mechanical engineers serves as a barrier. Social aspects such as economic, racial, and gender stereotypes have also been flagged up as barriers to learning programming. The most pressing barriers according to the survey are debugging, followed by learning the concepts and syntax.

With the disparity between the research into university level and secondary school level teaching methods, there is a clear knowledge gap in suitable teaching methods for secondary school students, particularly for engineering students, which shows that further research into the field is required. The literature review and survey conducted for the study is evidence that traditional approaches are not suitable for future teaching of programming (Vihavainen et al., 2011). Additionally, e-learning, according to the literature review, is considered to be one of the better methods of teaching for university-level students. Therefore, there was an attempt

at creating an e-learning software for secondary school students by the author. In terms of the survey, the participants agreed that a competition-based teaching method should be used to teach programming.

As discussed in section 5.2, the deployment of MATLAB was done via an API by the name of MATLAB Engine. Threading, Time and Random modules were used to program the software (see Appendix F), and three main exercises with variations were coded to tackle barriers such as debugging, syntax problems, concepts problems, and a lack of interest; to garner interest in programming for secondary school students interested in engineering, the problems were placed in a mechanical engineering context, and the questions were based on A-Level problems. The software has also been tested by a series of test data (see Appendix H) in order to root out the bugs. To assist the secondary school students with the exercises, cheatsheets have also been designed (see Appendix D).

For the first aim, the first objective was completed entirely since a questionnaire was created (see Appendix B), distributed and analysed by teachers from secondary schools, and the responses allowed the author to identify certain barriers that would have otherwise been left out. The survey also had a section dedicated to the methods of teaching programming as well. For the second aim, however, whilst a number of pedagogical literature have been studied and critically evaluated (see section 3), the variety of teaching methods was not large, and several academic journals were not evaluated.

For the second aim, learning all of Python and all of its API would require a much larger timeframe than given for the Individual Project, and hence only a fraction of the first objective was properly carried out. In terms of the second objective, exercises on oversights such as syntax, error checking, and correct application of

concepts were included in the software, and so the objective was fulfilled; since the software was programmed using Python, had the capability to call MATLAB and had questions based on the A-Level specification for mathematics, which contains mechanics, the third objective fulfilled (WJEC, 2019d; 2019e). However, the software did not contain any questions regarding the GCSE specification and was therefore partially fulfilled. In terms of the final two objectives of the second aim, the cheatsheets have been designed in full and can be observed in Appendix D, while the test data along with the outcomes can be observed in Appendix H, hence both of the objectives were fully fulfilled.

In terms of identifying barriers that secondary school students have to face and successful teaching methods that should be used for teaching programming, the immediate impact of the project is not large. However, the long-term impact for the project would become more obvious as time moves on due to the fact that the project raises awareness of the barriers that currently plague secondary school programming students. As a result of the project, more researchers would begin research on the aforementioned barriers as well as teaching methods that would be able to bypass the problems which have been identified.

## **6.1 Limitations**

Although a few limitations have already been discussed in the previous sections, not all of the flaws of the project have been discussed. To begin, the data from a few of the literature may no longer be accurate due to how old the studies were on a topic that is considered to be relatively new, including the study whose data was used to conduct the t-test. Also, many of the studies discussed researched into successful teaching methods of computer science rather than programming, which is much

broader and encompasses content that is irrelevant to programming. Merely a fraction of all the research on the topic is included in the literature review, hence there is a slim chance that other learning approaches such as competitions and puzzles are more widespread than e-learning.

As identified at the end of section 4, the low number of participants has meant that the results are less likely to be repeatable than if the number of participants was greater. Furthermore, the participants being programming teachers have meant that the data may be skewed towards a particular direction than if the participants were teachers of different disciplines. Also as mentioned in section 4, some of the questions for the survey were ambiguous to the point where a participant made a comment.

For the software, common oversights such as index being out of bounds are caught by the software, but the feedback given by the software is vague, which is unsuitable for revision; the software is merely a prototype and therefore has several flaws, including that the software has security flaws when using the exec function, which can potentially delete all files of a computer if abused by the user (Programiz, 2021). From a different perspective, the types of exercises and questions currently available are somewhat limited and so the impact that the current iteration of the software has on overcoming the identified barriers is weak. Lastly, the software has yet to be tested by end-users such as teachers or students.

## **6.2 Future Work**

Further research will need to be carried out on some of the identified barriers such as racial and gender stereotypes in order to find methods that could minimise the impact.

Even though the software is currently considered weak for overcoming barriers, the code that has already been written serves as a foundation for something which can easily be used by secondary school teachers and students. Nevertheless, the software in the current form would need to be tested by secondary school students to ensure that the software is appropriate and can be used in revision, which can be done. In order to make the software more accessible to people who have no experience with programming, the main menu should be converted into a GUI via the use of frameworks. Additionally, the cheatsheets in Appendix D should be incorporated into the software as one of the options to choose from in the main menu so that easy access by the students is possible. According to section 6.1, improvements on the software can also be made by upping the number of questions that can be asked by the software by taking advantage of the versatility of the text file reading as well as increasing the types of exercises available to better combat the barriers associated with learning programming. Akin to ‘Coursemarker’, ‘PIT’ and ‘SPOT’ mentioned in section 3.3.2, more tools geared towards in-person classes may alleviate problems the teacher has, who can then plan activities designed to overcome barriers that the particular class is prone to have (Higgins et al., 2005; El-Zein et al., 2009; Rehberger et al., 2013). An example would be online and competition features due to the results from the survey suggesting competitions as a possibly good method of teaching; each person in a class has to write and submit a script for a particular problem to the software; furthermore, each student has to provide an input which the student believes would crash the script of other students in the class, allowing the teacher to score the scripts based on whether the script of the student solves the problem and whether the script is robustness.

## 7. References

AQA (2019a). *AS and A-Level Computer Science*. Available at: <https://filestore.aqa.org.uk/resources/computing/specifications/AQA-7516-7517-SP-2015.PDF> (Accessed: 20/4/21).

AQA (2019b). *Exam results statistics - June 2019*. Available at: [https://filestore.aqa.org.uk/over/stat\\_pdf/AQA-GCSE-STATS-JUN-2019.PDF](https://filestore.aqa.org.uk/over/stat_pdf/AQA-GCSE-STATS-JUN-2019.PDF) (Accessed: 20/4/21).

AQA (2020). *GCSE Computer Science*. Available at: <https://filestore.aqa.org.uk/resources/computing/specifications/AQA-8520-SP-2016.PDF> (Accessed: 20/4/21).

AQA (2021). *Teaching Resources*. Available at: <https://www.aqa.org.uk/subjects/computer-science-and-it/gcse/computer-science-8520/teaching-resources> (Accessed: 20/3/21).

Azemi, A. and Pauley, L. L. (2008). 'Teaching the introductory computer programming course for engineers using Matlab'. In: *2008 38th Annual Frontiers in Education Conference*, 22-25 Oct. 2008 2008. pp. T3B-1-T3B-23.

BBC (2021a). *User interfaces*. Available at: <https://www.bbc.co.uk/bitesize/guides/zwb4jxs/revision/1#:~:text=A%20command%2Dline%20interface%20allows,PCs%20used%20command%2Dline%20interfaces>. (Accessed: 6/5/21).

BBC (2021b). *Development and testing*. Available at: <https://www.bbc.co.uk/bitesize/guides/z8n3d2p/revision/7> (Accessed: 8/5/21).

Blumenstein, M. (2002). 'Strategies for improving a Java-based, first year programming course'. In: *International Conference on Computers in Education*,



2002. Proceedings., 3-6 Dec. 2002 2002. pp. 1095-1099 vol.2 (Accessed: 11/11/20).

Buyrukoglu, S., Batmaz, F. & Lock, R. (2016). '*Semi-Automatic Assessment Approach to Programming Code for Novice Students*', Proceedings of the 8th International Conference on Computer Supported Education, Rome, Italy: SCITEPRESS - Science and Technology Publications, Lda, pp. 289–297.

Buyrukoglu, S., Batmaz, F. & Lock, R. (2019). '*Improving marking efficiency for longer programming solutions based on a semi-automated assessment approach*', Computer Applications in Engineering Education, 27(3), pp. 733-743.

Christensson, P. (2013). *Framework Definition*. Available at: <https://techterms.com> (Accessed: 14/11/20).

Coravu, L., Marian, M. & Ganea, E. (2015). 'Scratch and recreational coding for kids'. In: *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, 24-26 Sept. 2015 2015. pp. 85-89 (Accessed: 11/11/20).

Cyr, M., Miragila, V., Nocera, T. & Rogers, C. (1997). 'A Low Cost, Innovative Methodology for Teaching Engineering Through Experimentation', *Journal of Engineering Education*, 86, pp. 167-171.

Dawson, J. Q., Allen, M., Campbell, A. & Valair, A. (2018). 'Designing an Introductory Programming Course to Improve Non-Majors' Experiences', *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, Baltimore, Maryland, USA: Association for Computing Machinery, pp. 26–31.

Day, C. (2004). 'A Passion for Teaching', *A Passion for Teaching*, pp. 1-170.

Department of Education (2013a). *National curriculum in England: computing programmes of study*. Available at: <https://www.gov.uk/government/publications/>

[national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study](#) (Accessed: 12/11/20).

Department of Education (2013b). National curriculum in England: design and technology programmes of study. Available at: <https://www.gov.uk/government/publications/national-curriculum-in-england-design-and-technology-programmes-of-study/national-curriculum-in-england-design-and-technology-programmes-of-study> (Accessed: 12/11/20).

Elnagar, A., Ali, M. (2012). 'A modified team-based learning methodology for effective delivery of an introductory programming course', *Proceedings of the 13th annual conference on Information technology education*, Calgary, Alberta, Canada: Association for Computing Machinery, pp. 177–182.

El-Zein, A., Langrish, T. & Balaam, N. (2009). 'Blended Teaching and Learning of Computer Programming Skills in Engineering Curricula', *Advances in Engineering Education*, 1.

Erwin, B., Cyr, M. & Rogers, C. (2000). 'LEGO Engineer and RoboLab: Teaching Engineering with LabVIEW from Kindergarten to Graduate School', *International Journal of Engineering Education*, 16.

Esteves, M., Fonseca, B., Morgado, L. & Martins, P. (2011). 'Improving teaching and learning of computer programming through the use of the Second Life virtual world', *British Journal of Educational Technology*, 42(4), pp. 624-637.

Figueiredo, J., García-Peñalvo, F. J. (2019). 'Teaching and learning strategies of programming for university courses', *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality*, León, Spain: Association for Computing Machinery, pp. 1020–1027.

Gill, G. & Holton, C. F. (2006). 'A Self-Paced Introductory Programming Course', *Journal of Information Technology Education: Research*, 5(1), pp. 95-105.

Guo, P. J. (2018). 'Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities', *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, Montreal QC, Canada: Association for Computing Machinery, p. Paper 396.

Hermans, F. & Aivaloglou, E. (2017). 'Teaching Software Engineering Principles to K-12 Students: A MOOC on Scratch'. In: *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, 20-28 May 2017 2017. pp. 13-22.

Higgins, C. A., Gray, G., Symeonidis, P. & Tsintsifas, A. (2005). 'Automated assessment and experiences of teaching programming', *J. Educ. Resour. Comput.*, 5(3), pp. 5–es.

Hirsch, L. S., Carpinelli, J. D., Kimmel, H., Rockland, R. & Bloom, J. (2007). 'The differential effects of pre-engineering curricula on middle School Students' attitudes to and knowledge of engineering careers'. In: *2007 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, 10-13 Oct. 2007 2007. pp. S2B-17-S2B-21 (Accessed: 30/4/21).

Hogle, J. G. (1995). *Computer Microworlds in Education [microform] : Catching Up with Danny Dunn / Jan G. Hogle*. [Washington D.C.]: Distributed by ERIC Clearinghouse.

Jonassen, D., Spector, M. J., Driscoll, M., Merrill, M. D. & van Merriënboer, J. (2008). *Handbook of Research on Educational Communications and Technology: A Project*

of the Association for Educational Communications and Technology: Taylor & Francis.

Kent State University (2021). *SPSS TUTORIALS: INDEPENDENT SAMPLES T TEST*. Available at: <https://libguides.library.kent.edu/SPSS/IndependentTTest> (Accessed: 4/5/21).

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J. & Werner, L. (2011). 'Computational thinking for youth in practice', *ACM Inroads*, 2(1), pp. 32–37.

Lewis, J., Sauro, J (2020). *HOW TO CONVERT BETWEEN FIVE- AND SEVEN-POINT SCALES*. Available at: <https://measuringu.com/convert-point-scales/> (Accessed: 4/5/21).

López-Pernas, S., Gordillo, A., Barra, E. & Quemada, J. (2019). 'Examining the Use of an Educational Escape Room for Teaching Programming in a Higher Education Setting', *IEEE Access*, 7, pp. 31723-31737.

Lui, A. K., Kwan, R., Poon, M. & Cheung, Y. H. Y. (2004). 'Saving weak programming students: applying constructivism in a first programming course', *SIGCSE Bull.*, 36(2), pp. 72–76.

Mathworks Inc. (2021). *Install MATLAB Engine API for Python*. Available at: [https://uk.mathworks.com/help/matlab/matlab\\_external/install-the-matlab-engine-for-python.html](https://uk.mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html) (Accessed: 29/4/21).

McDowell, C., Werner, L., Bullock, H. & Fernald, J. (2002). 'The effects of pair-programming on performance in an introductory programming course', *SIGCSE Bull.*, 34(1), pp. 38–42.

Merrick, K. E. (2010). 'An Empirical Evaluation of Puzzle-Based Learning as an Interest Approach for Teaching Introductory Computer Science', *IEEE Transactions on Education*, 53(4), pp. 677-680.

Mulesoft (2020). *What is an API? (Application Programming Interface)*. Available at: <https://www.mulesoft.com/resources/api/what-is-an-api> (Accessed: 14/11/20).

Nagappan, N., Williams, L., Wiebe, E., Miller, C., Balik, S., Ferzli, M. & Petlick, J. (2003). 'Pair Learning: With an Eye Toward Future Success'. In: *Maurer, F. & Wells, D., eds. Extreme Programming and Agile Methods - XP/Agile Universe 2003, 2003//2003 Berlin, Heidelberg. Springer Berlin Heidelberg. pp. 185-198* (Accessed: 30/3/21).

Nikolic, S., Ros, M. & Hastie, D. B. (2018). 'Teaching programming in common first year engineering: discipline insights applying a flipped learning problem-solving approach', *Australasian Journal of Engineering Education*, 23(1), pp. 3-14.

OCR (2019). *Final Results Statistics*. Available at: <https://www.ocr.org.uk/Images/552371-gcse-cambridge-nationals-and-other-level-2-final-exam-statistics-june-2019.pdf> (Accessed: 20/4/21).

OCR (2020a). *GCSE (9-1) Specification Computer Science*. Available at: <https://www.ocr.org.uk/Images/225975-specification-accredited-gcse-computer-science-j276.pdf> (Accessed: 20/4/21).

OCR (2020b). *AS Level Specification Computer Science*. Available at: <https://www.ocr.org.uk/Images/170845-specification-accredited-as-level-gce-computer-science-h046.pdf> (Accessed: 20/4/21).

OCR (2020c). *A Level Specification Computer Science*. Available at: <https://www.ocr.org.uk/Images/170844-specification-accredited-a-level-gce-computer-science-h446.pdf> (Accessed: 20/4/21).

OmniSci (2020). *Graphical User Interface*. Available at: <https://www.omnisci.com/technical-glossary/graphical-user-interface> (Accessed: 14/11/20).

Ortiz, O. O., Franco, J. Á. P., Garau, P. M. A. & Martín, R. H. (2017). 'Innovative Mobile Robot Method: Improving the Learning of Programming Languages in Engineering Degrees', *IEEE Transactions on Education*, 60(2), pp. 143-148.

Parsons, D. & Haden, P. (2006). 'Parson's programming puzzles: a fun and effective learning tool for first programming courses', *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, Hobart, Australia: Australian Computer Society, Inc., pp. 157–163.

Pearson (2019a). *Pearson BTEC Certificates for the Level 3 Software Development Technician Apprenticeship Standard*. Available at: [https://qualifications.pearson.com/content/dam/pdf/BTEC-Specialist-Qualifications/Programming/2019/specification-and-sample-assessments/9781446960226\\_BTEC\\_Splt\\_L3\\_SoftDevTech.pdf](https://qualifications.pearson.com/content/dam/pdf/BTEC-Specialist-Qualifications/Programming/2019/specification-and-sample-assessments/9781446960226_BTEC_Splt_L3_SoftDevTech.pdf) (Accessed: 20/4/21).

Pearson (2019b). *GCSE (9-1) Specifications Grade Statistics (Final)*. Available at: <https://qualifications.pearson.com/content/dam/pdf/Support/Grade-statistics/GCSE/grade-statistics-june-2019-final-gcse-9-1-specifications.PDF> (Accessed: 20/4/21).

Pearson (2020). *GCSE (9-1) Computer Science*. Available at: [https://qualifications.pearson.com/content/dam/pdf/GCSE/Computer%20Science/2020/specification-and-sample-assessments/GCSE\\_L1\\_L2\\_Computer\\_Science\\_2020\\_Specification.pdf](https://qualifications.pearson.com/content/dam/pdf/GCSE/Computer%20Science/2020/specification-and-sample-assessments/GCSE_L1_L2_Computer_Science_2020_Specification.pdf) (Accessed: 20/4/21).

Piteira, M. & Costa, C. 'Learning computer programming: study of difficulties in learning programming', *Proceedings of the 2013 International Conference on*

*Information Systems and Design of Communication*, Lisboa, Portugal: Association for Computing Machinery, pp. 75–80.

Programiz (2021). *Python exec()*. Available at: <https://www.programiz.com/python-programming/methods/built-in/exec> (Accessed: 9/5/21).

Python Software Foundation (2021a). *time — Time access and conversions*. Available at: <https://docs.python.org/3/library/time.html> (Accessed: 30/4/21).

Python Software Foundation (2021b). *threading — Thread-based parallelism*. Available at: <https://docs.python.org/3/library/threading.html> (Accessed: 30/4/21).

Python Software Foundation (2021c). *random — Generate pseudo-random numbers*. Available at: <https://docs.python.org/3/library/random.html> (Accessed: 30/4/21).

Python Software Foundation (2021d). *6. Modules*. Available at: <https://docs.python.org/3/tutorial/modules.html> (Accessed: 6/5/21).

Rehberger, S., Frank, T. & Vogel-Heuser, B. (2013). 'Benefit of e-learning teaching C-programming and software engineering in a very large mechanical engineering beginners class'. In: *2013 IEEE Global Engineering Education Conference (EDUCON)*, 13-15 March 2013 2013. pp. 1055-1061 (Accessed: 8/3/21).

Rubio, M. A., Hierro, C. M. & Pablo, A. (2013). 'Using arduino to enhance computer programming courses in science and engineering'. In: *Proceedings of EDULEARN13 conference, 2013*. IATED Barcelona, Spain. pp. 1-3 (Accessed: 4/3/21).

Sentance, S., Waite, J. & Kallia, M. (2019). 'Teaching computer programming with PRIMM: a sociocultural perspective', *Computer Science Education*, 29(2-3), pp. 136-176.

Stangroom, J. (2021). *The Kolmogorov-Smirnov Test of Normality*. Available at: <https://www.socscistatistics.com/tests/kolmogorov/default.aspx> (Accessed: 4/5/21).

Sun, W., Sun, X. (2011). 'Teaching Computer Programming Skills to Engineering and Technology Students with a Modular Programming Strategy', *2011 ASEE Annual Conference & Exposition*, Vancouver, BC, 2011/06/26.

Tillmann, N., Halleux, J. d., Xie, T., Gulwani, S. & Bishop, J. (2013). 'Teaching and learning programming and software engineering via interactive gaming'. In: *2013 35th International Conference on Software Engineering (ICSE)*, 18-26 May 2013 2013. pp. 1117-1126 (Accessed: 11/11/20).

Tsai, C. Y. (2019). 'Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy', *Computers in Human Behavior*, 95, pp. 224-232

Vihavainen, A., Paksula, M. & Luukkainen, M. (2011). 'Extreme apprenticeship method in teaching programming for beginners', *Proceedings of the 42nd ACM technical symposium on Computer science education*, Dallas, TX, USA: Association for Computing Machinery, pp. 93-98.

Vihavainen, A., Airaksinen, J. & Watson, C. (2014). 'A systematic review of approaches for teaching introductory programming and their influence on success', *Proceedings of the tenth annual conference on International computing education research*, Glasgow, Scotland, United Kingdom: Association for Computing Machinery, pp. 19-26.

Wang, E. (2001). 'Teaching freshmen design, creativity and programming with LEGOs and Labview'. In: *31st Annual Frontiers in Education Conference*. Impact on



Engineering and Science Education. Conference Proceedings (Cat. No.01CH37193), 10-13 Oct. 2001 2001. pp. F3G-11 (Accessed: 11/11/20).

WJEC (2019a). *WJEC GCSE in Computer Science*. Available at: <https://www.wjec.co.uk/media/jymdzl0a/wjec-gcse-comp-science-spec-2017-e-04-05-2020.pdf> (Accessed: 20/4/21).

WJEC (2019b). *WJEC Eduqas GCE A Level in Computer Science*. Available at: <https://www.eduqas.co.uk/media/zhpapwjj/eduqas-a-level-computer-science-spec-from-2015-e-24-01-2020.pdf> (Accessed: 20/4/21).

WJEC (2019c). *WJEC Legacy and Revised New Wales Final GCSE Results*. Available at: <https://www.wjec.co.uk/media/a1ep0gcx/final-results-june-2019.pdf> (Accessed: 20/4/21).

WJEC (2019d). *WJEC GCE AS/A Level in Further Mathematics*. Available at: <https://www.wjec.co.uk/media/hcpnsu41/wjec-gce-further-maths-spec-from-2017-e.pdf> (Accessed: 7/5/21).

WJEC (2019e). *WJEC GCE AS/A Level in Mathematics*. Available at: <https://www.wjec.co.uk/media/lm3fegtu/wjec-gce-maths-spec-from-2017-e.pdf> (Accessed: 7/5/21).

WJEC (2020). *WJEC Eduqas GCE AS in Computer Science*. Available at: <https://www.eduqas.co.uk/media/wc2fgbyp/eduqas-as-computer-science-spec-from-2015-e-090119.pdf> (Accessed: 20/4/21).

WJEC (2021). *All resources by subject*. Available at: <https://resources.wjec.co.uk/Pages/ResourceByArgs.aspx?subid=6&lvlid=2> (Accessed: 20/4/21).

Woodley, M., Kamin, S. N. (2007). 'Programming studio: a course for improving programming skills in undergraduates', SIGCSE Bull., 39(1), pp. 531–535.

Yacob, A. & Saman, M. Y. M. (2012). 'Assessing level of motivation in learning programming among engineering students'. *In: The International Conference on Informatics and Applications (ICIA2012)*. Malaysia:[sn], 2012. pp. 425-432.

Yildiz Durak, H. (2018). 'Digital story design activities used for teaching programming effect on learning of programming concepts, programming self-efficacy, and participation and analysis of student experiences', *Journal of Computer Assisted Learning*, 34(6), pp. 740-752.

## **Appendices**

### **Appendix A - Reflection and Project Management**

Looking back to the first semester of the year, the focus of the author was mainly on the other units such as Operations Management tests and coursework, or Design 3 reports as opposed to the Individual Project. As such, only two-thirds of the research was carried out in terms of pedagogical literature alongside the preliminary plans for the survey during the first semester; research into Python and MATLAB code began alongside the brainstorming of the software exercises. At the end of the first semester, a reassessment of the objectives was carried out which ultimately meant that the idea of a GUI for the software was dropped. Because the GUI was dropped, research into GUI frameworks was also dropped.

At the beginning of the second semester, the generation of the cheatsheets began; following that, the programming of the software also began, and the bulk of the code was completed within 2 weeks, which although started later than expected was finished a lot sooner than expected as well. At roughly the same time, however, the decision to abandon the ethical approval was made as there was no response from the ethical team, which was predicted in the project plan. Instead, a contingency plan from the project proposal was used, whereby the analysis period for the survey was cut down and pushed to a later date in order to compensate for the delay. In addition, only secondary school teachers would be allowed to respond to the survey as opposed to both teachers and students, allowing the author to bypass the ethical approval stage. The survey was finalised and was sent to both Dr. Simmons and Technocamps for distribution.

The most time-consuming part of the project was attempting to find pedagogical literature and reviewing the academic journals, which took 2 months during the

second semester. During the two months, minor tweaks and engineering contexts were added to the software as well as finalisations to the cheatsheets; the MATLAB engine API was incorporated into one of the exercises of the software and a main menu was generated. Foreseeing that the code would be difficult to test in the project plan, the idea of splitting up the code into functions was followed, which made testing simpler as only individual sections needed to be tested; a series of test data was used in order to accomplish the goal. Once all the other tasks were completed, the analysis of the survey began, and the conclusion, analysis of limitations, and reflections began.

Whilst a few objectives have had to change due to a review or due to external factors, the methodology that was actually used in the project did not vary much from the methodology written in the project plan.

Throughout the Individual Project, being stressed was a major issue for the author due to being unfamiliar with doing large projects with much less guidance in comparison to the other units in the past. Additionally, adhering to either of the project plans were difficult since both of the plans were too optimistic; the first plan did not account for coursework and reports from other units of the course, and the second project plan did not account for the responsiveness of other people such as the ethics team and the participants of the study as well as the reading and writing skills of the author. As such, time management was a major problem throughout the project leading to stress, and many of the milestones were only reached closer towards the end of the second semester. Problems with time management have also meant that much of the time spent in the second semester was on the project as opposed to revision for units in the second semester. Being unable to go to the university to complete the project had meant that more time was spent

procrastinating as well, despite having had social media websites disabled for all devices.

Excitement and uncertainty were feelings felt by the author at the start of the project, as while the author had done mentoring and programming prior to the project and would like to know methods that could assist in both learning and teaching programming, the author also did not know what would be required in order to complete the project. Moreover, the COVID-19 pandemic had only exacerbated the feeling of uncertainty as actions such as distributing surveys and communicating with the supervisor regarding the project was made more difficult.

Especially during the first 2 months of the second semester, a sense of dispiritment was felt as the research and literature review seemed repetitive, and the pandemic has meant that exchanging ideas with peers in order to keep the reviewing part of the project fascinating had become impossible. Furthermore, spending a large portion of time at home was somewhat demotivating.

When Technocamps agreed to assist in distributing the survey, a sense of surprise was felt and the motivation for completing the project was slowly renewed; the feeling has amplified the closer towards the deadline of the project. As the project nears its final stages, a sense of relief and proudness can also be felt, as the Individual Project is one that has spanned many months and is one of the longest projects the author has completed to date.

Through the Individual Project, the author has learnt that having a separate work only account of a computer and disabling social media on devices are crucial in minimising procrastination, which is something that the author will use in future projects and essays. As well, the facilities open to students at the university are vital for long essays and projects due to the more serious environment, allowing

students to gain motivation. Additionally, the use of remote learning for long projects such as the Individual Project is demoralising, and steps should be taken to ensure that such effects are minimised. One method of improving such a situation is to have some sort of reward once certain milestones have been achieved.

Reflecting on the Individual Project, one of the main areas which could have been improved was the utilisation of even more connections to distribute the survey, as is one of the main things to be wary of should a similar project be carried out in the future. As mentioned previously, using the university facilities or different environments to work on long projects would be beneficial in combatting repetitiveness and motivation problems. Finally, despite having mentioned the use of pseudocode in the project plan, pseudocode was not used in the programming of the software. In future programming projects, the author will bear in mind to use such a tool to save time and effort.

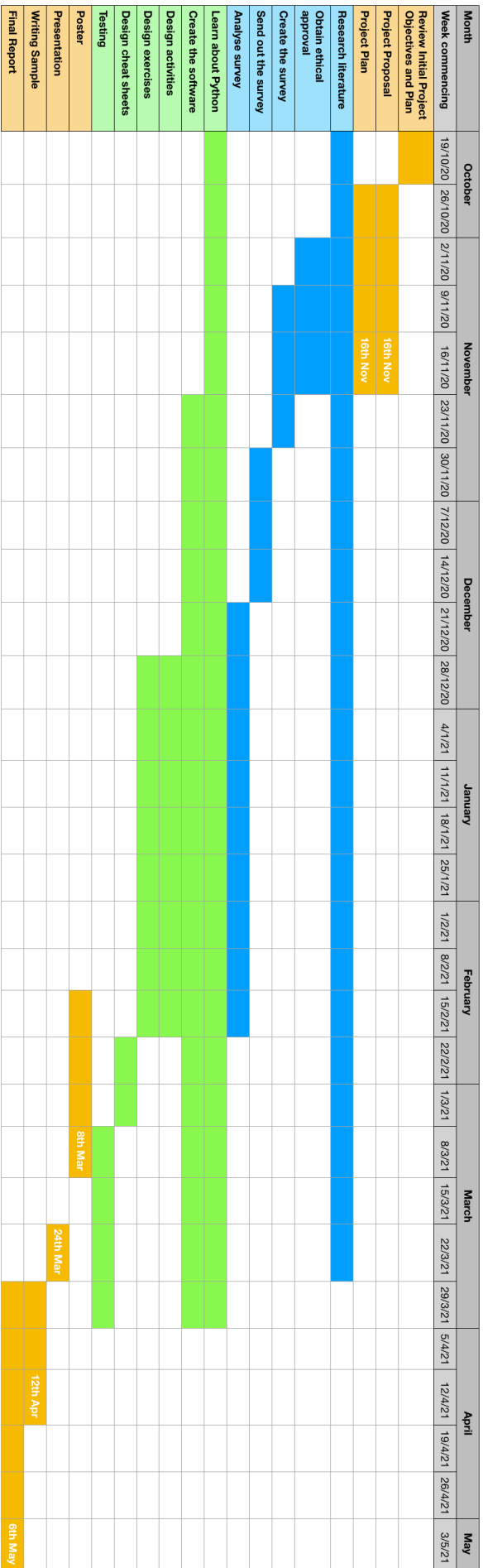


Figure 16: Initial Project Plan

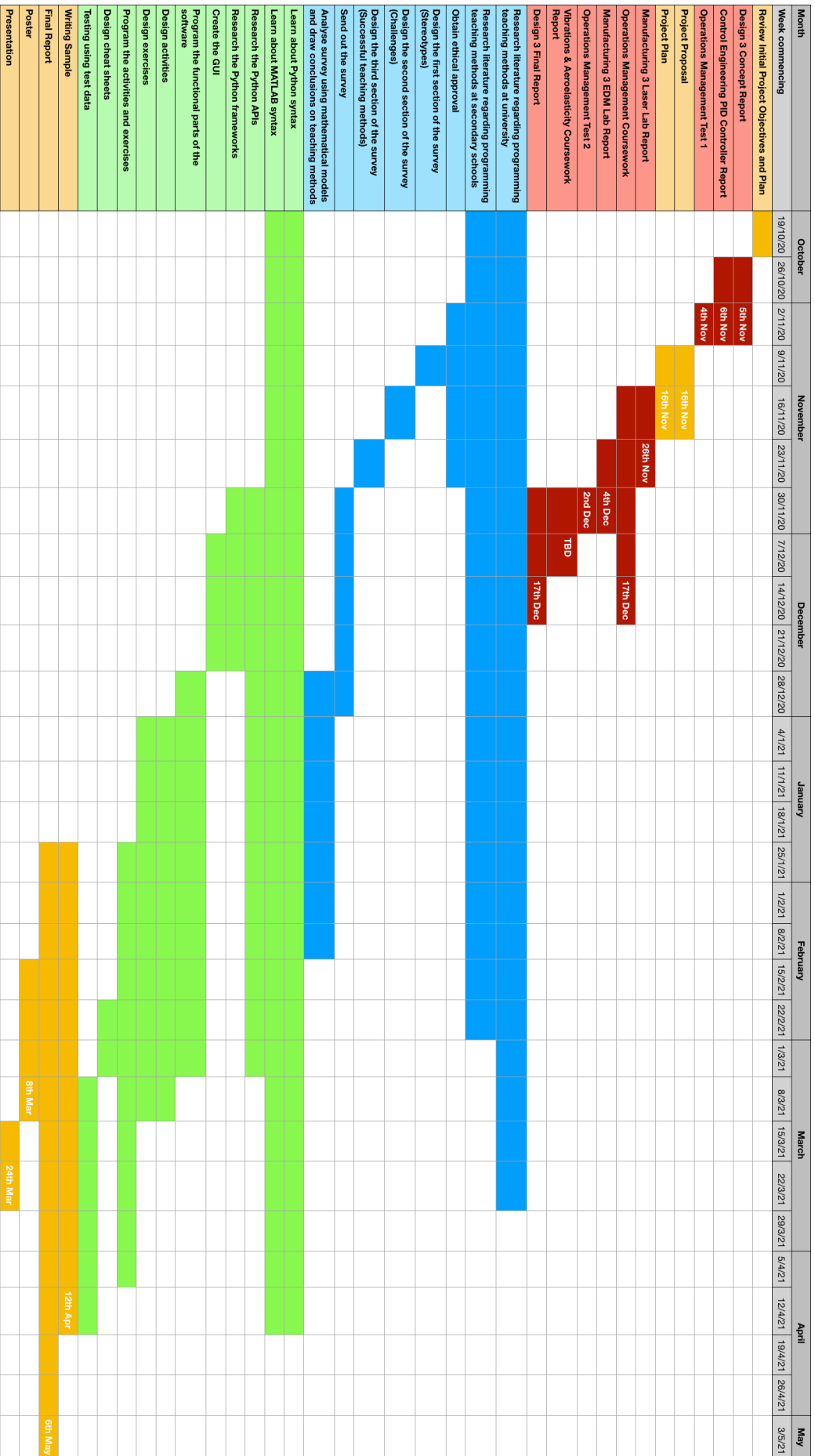


Figure 17: Revised Project Plan



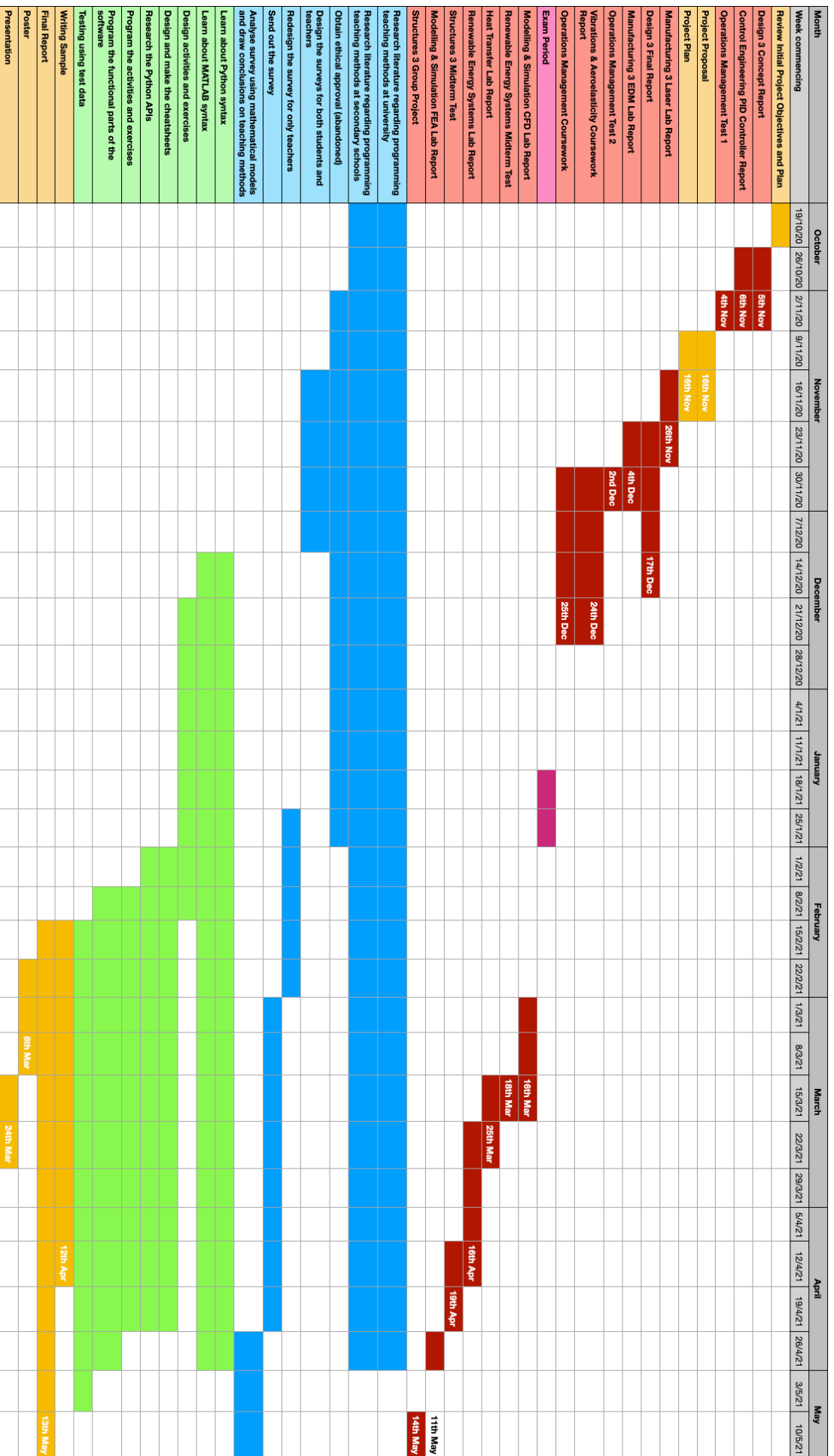


Figure 18: Actual Project Timeline

## Appendix B - Survey

# Engineering Programming Questionnaire

The questionnaire is mainly aimed for teachers and professionals working within secondary schools. The purpose of this is to help understand programming in secondary schools and its implications for the future of digital engineering. The results from this questionnaire will then be used in a 3rd year university Individual Project.

No personal data will be collected, and so all submissions are anonymous.

Thank you in advance!

\* Required

1. From 1 to 5, how much do you agree with the statement that "Programming skills are important to have in general, regardless of profession"? \*

	1	2	3	4	5	
Strongly agree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly disagree

2. From 1 to 5, how much do you agree with the statement that "Programming skills are integral to engineers"? \*

	1	2	3	4	5	
Strongly agree	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Strongly disagree

3. Please provide a reason for your answer to question 2. \*

Your answer

---

## Future Careers Options

This section is concerned with the future careers of secondary school students and what factors play a role when they are considering what career options to pursue.

4. Would you consider the students to be well-informed about what engineering entails when considering their future career options? \*

Yes

No

5. What would you consider to be the main factors that would affect a student's decision to learn programming or pursue a programming-related career? \*

Your answer

---

## Barriers with Learning Programming

This section is mainly concerned with all the different types of barriers that secondary school students may encounter when attempting to learn programming.

6a. Based on your experience (or students' experiences whilst attempting to learn programming), how easy or difficult would you consider the following to be in a programming context? \*

	1 (Very easy)	2 (Easy)	3 (Average)	4 (Difficult)	5 (Very difficult)
Finding motivation to learn programming	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Finding where to begin	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Finding methods to write a program	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The concepts (what each thing does)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The syntax (the structure of the code)	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Testing the code	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Debugging the code	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

6b. Apart from the aforementioned, what other aspects would you consider to be difficult when attempting to learn programming?

Your answer

---

## Methods of Teaching Programming

This section is concerned with the different types of programming teaching methods that have been implemented by various people in the past.

7a. From 1 to 5, how much do you agree with the following statements? \*

	1 (Strongly Agree)	2 (Agree)	3 (Don't know)	4 (Disagree)	5 (Strongly Disagree)
"Students should learn programming by going through theory only (reading online resources, books etc.) and then do a written exam"	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
"Students should learn programming by each coming up with and doing a programming-based project"	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
"Students should learn programming by doing a programming-based project and also go through the theory and sit an exam"	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
"Students should learn programming by doing a school programming-based competition"	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7b. Apart from the aforementioned, what other methods would you consider to be an effective method of teaching students (either in programming or in general)?

Your answer

---

*Figures 19a-19e: Screenshots of the survey*

## Appendix C - Participants' Paragraph Answers and T-Test Values

### Question 3

- Maths is needed for engineering and is covered in the CS modules
- Even if programming skills are not relevant to an engineer's current job, they could be relevant to future jobs, so having those skills gives an engineer more options in terms jobs that they can apply for and jobs that they can do.
- all engineering/stem jobs will not involved computer and programming to take advantage of the technology available to us in the sector
- Engineers need to understand what is happening with within the process, so need to understand inputs/outputs and programming helps to get a further understanding of that
- Problem solving etc
- Quite subjective as this may not apply to all engineering professions such as mechanical engineering. Problem solving skills and the ability to break a problem down would be more relevant. If the question indicated which engineering group, a more specific response could be given. E.g. Electrical engineering -"Strongly Agree".
- The quickest way to analyse data nowadays is usually to throw it into a computer model using something like the Matplotlib or Pandas library in Python
- I don't believe one's value is tied directly to their tangible skills. There are swathes of people who specialise in theoretical computer science and are the brains behind development teams at University level. Whilst it's quite important (more so as the years pass) that they have a basic understanding of programming principles, the actual formal ability to program in a given language is not as necessary. Is it a good career choice - yes.

Must you understand functions and programming paradigms and patterns - yes. There is space for anything the world needs and whilst programming is more and more prevalent, the success of engineers and software engineers is simply correlation not causation. Who is to say that windmill repair doesn't become the next big thing :)

In short, programming is a medium for people to solve problems within a clearly defined syntax and set of rules. This makes understanding problems simpler by abstracting the nuance of language - it is as integral as learning another language (arguably more so) but ultimately we aren't at a stage where we should make everyone a programmer.

- Engineering is about solving a problem.

When someone is programming (especially at a GCSE level) they are solving simple problems computational

- As engineers are making robots etc understanding how that robot is made to function using programming is essential. Also autonomous factories have engineers design the equipment and they need to understand how they can make it function when making something for a customer/client
- Programming teaches you to break problems down into smaller parts. A process that we call decomposing the problem. This is a useful life skill regardless of your profession.
- Because engineering contains a load of calculation, modelling, simulations etc that a knowledge of programming would help with.
- Because being able to think computationally improves your problem solving skills as well as being able to model, design using computational models, etc.



- Machine code skills and C programming language is very useful towards programming CAD/CAM and CNC milling machines, as well as programming robotic arms.
- Understanding the program or design you are going to use will help collate information for a better understanding.

### **Question 5**

- Perceived difficulty of the subject and not enough time spent at KS3
- Interest and motivation - if this is an area that a pupil is not interested in, then they are unlikely to learn programming, or pursue programming-related careers. Also, even though SO MANY people in today's World are extremely reliant on computers (phones, laptops, tablets, etc...), are in love with their computers, and could not function without their computers, their interest is in the using of the computers, and not how they work and how they are made to work - e.g. many school pupils would rather play a computer game and understand how the game that they love works. They are stuck in the Matrix, but they cannot see the wood for the trees!

Lack of understanding and awareness - many pupils do not know the opportunities that could be available to them, if they learn programming skills (i.e. all of the different jobs that it can be used in, and all of the future jobs that will require it).

Fear of the subject and lack of literacy skills.

Lack of role models in the field - in certain areas/communities/families, pupils do not know someone involved in programming-related jobs, so they do not 'aspire' to be like someone who does that job.

Sometimes, it is also possible for educators to be unable to see things from the pupils' points of view - e.g. I have seen people who are really good at programming not being able to understand how pupils cannot do/learn some of the most basic programming techniques, and so struggling to pitch activities/teaching at the correct level for the pupils to access it.

Sometimes, schools introduce Computer Science to the curriculum, and teachers are expected to teach Computer Science and programming, but they are not given the time and the training needed for them to develop their own skills. There are an awful lot of skills and knowledge needed for the subject, and these skills and knowledge are not things that teachers should just be expected to 'have' or to develop in their own time. Teachers need to be given time and resources to develop their own skills, before they can be expected to teach them to the pupils.

- how much it is used in all sciences, media, problem solving, design, everything! and HOW it is used.
- Stereotypes
- Enjoyment & money
- Passion for the subject. Genuine interest.
- Confidence, prior attainment (e.g. success in coding lower down the school)
- Their exposure to good teaching and supportive teachers who introduce the subject and modules in an engaging manner is the main factor. Our decisions (scientifically) are dictated by our genetics and our environment - the ability for programming/engineering principles to be applied to every field can mean that even someone who is not inclined to enter the industry can see it's beauty from a different angle.

The underlying thing here is motivation - this can be made extrinsic using the high

salaries and competitions that are prevalent currently, but I don't think this is an ideal means of affecting a student's decision.

- Programming is such a small part of the GCSE computer science specification.  
Programming is essentially learning a new language, it takes hard work, and dedication to master it. Without an interest or natural ability for it, it's hard, the level of difficulty puts people off.  
Not all Computer Science teachers are aware of the Labour Market information relating to programming related careers, and so can not share that with their students.
- They think a programming career involves sitting in front of a computer all the time not understanding the impact the use of such a skill can have on people's lives
- A genuine interest in coding and solving problems.
- How the topic of programming is presented at a younger age. If it is dry and not for a purpose they are interested in, they won't be interested. If it is in a fun context or a context that they decide on themselves like a personal project, then they will become more interested and more likely to learn it.
- They perceive it as difficult and/or boring, mainly to the way that it is taught and the content of the qualifications.
- The money and the challenge!
- Capabilities would be the biggest factor or barrier as it can become very stressful but achieving the end result will encourage students to gain self belief and have enjoyment

### Question 6a

H<sub>0</sub>: The mean of the survey data set is the same as the mean of the data gathered by Piteira and Costa (2013) for the teachers (different countries)

H<sub>1</sub>: The mean of the survey data set is NOT the same as the mean of the data gathered by Piteira and Costa (2013) for the teachers (different countries)

Table 7: Numbers and calculations for the T-test

T-test	Syntax Data Set	Debugging Data Set
Mean (Likert seven-point scale)	$\frac{(2.5 \times 6) + (4 \times 2) + (5.5 \times 6) + (7 \times 1)}{15} = 4.2$	$\frac{(2.5 \times 2) + (4 \times 7) + (5.5 \times 4) + (7 \times 2)}{15} = 4.6$
Standard Deviation	1.590148	1.365388
T-value	1.94797	1.68349
Degree of Freedom	20.245 $\approx$ 20	22.995 $\approx$ 23
P-value at 10% $\alpha$ -level	1.725	1.714

### Question 6b

- Pseudocode

- Knowing where to start.

Knowing what language to start with.

Understanding the purpose of what you are learning.

Understanding what you are learning can be used for.

Seeing where these skills can lead and take you.

Lack of ability to focus and concentrate on something that is not big, loud and entertaining.

Access to equipment and resources for learning.

- getting over the assumption that it is hard, complicated and harder than other subjects

- real world examples that they can relate to.
- Previous experiences, exposure to the subject, regular access to resources.
- Media stereotyping of STEM practitioners can put off e.g. girls and people of colour
- In most cases there is a central source for good resources that is tailored to engage KS3 - for example BBC Bitesize. Whilst with CS we have W3Schools and some others, they are more documentation-style websites which require a student to already have motivation for learning and they are less likely to encourage or nurture a students early learning.
- I find that students often find it difficult to apply what they have learn't to decomposing other problems. Often they have the skills but don't necessarily realise that they can be applied to a given situation. A lot of this stems from the fact that they don't always code between lessons. Which is vital in this subject. Plus the fact that with the WJEC we are trying to cram three modules into two lessons per week. Three into two simply won't go.
- Focusing on the general programming concepts as opposed to how to implement them in a specific language.
- The computational thinking behind it needs to be taught and understood first.
- The difference between procedural and object orientated
- Ensuring all students understand the functions and operational components that are taking place

### **Question 7b**

- Make the programming relevant ie use Edbots and DJI drones along with other practical examples.

- Introduce them to the basics of the subject and lead them to achieve success with the basic skills.

Try to find areas of the subject that interest them.

Introduce some of the interesting history of the subject.

Give the pupils a greater awareness of the purpose of what they are learning and why it could be really useful for them (i.e. make them understand what you are teaching them - don't just teach it to them).

Give the pupils a variety of experiences with the subject (e.g. programming games, programming maths tools, debugging, programming with different languages, complete a programming-based project, learn theory, exams, etc...).

Give the pupils more of an idea of what jobs the subject could lead them to in the future.

Understand that we are all different, so what works (in terms of learning) for one pupil, will not necessarily work for another - there is no one set way of learning/doing something.

Understand that some pupils are just not interested in programming, and never will be, so as long as you do your job and give them a basic introduction and try to help them to find interest in the subject, then you are teaching them in the right way.

- start them young. start using tools and apps for programming at a young age and keep that momentum throughout their education. make it second nature, just like using a computer for PowerPoint or writing an essay is already
- small programs that they are given they edit and improve them to be able to see them working

- Unplugged activities. Not everything in CS needs to be in front of a computer screen. Not all students learn in the same way and or retain information. Many programming concepts can be taught through alternative interactive tasks to gain the core basic understanding before trying to apply it to code.

Teachers need to be able to adapt and be dynamic when delivering complex topics and especially when teaching programming skills. Failure to do so results in loss of interest and motivation from students. Being able to think creatively and logically as well as knowing your class when planning programming lessons is essential to its success.

- Project-based learning e.g. build a product
- Fixed (university-style) assignments whereby the project is determined by the assessor and they are introduced to the concept of a marking rubric, as they develop the stringent requirements of the rubric slowly become more vague until they are at a stage where they understand how to break down an idea into requirements using SDLC practices.
- Tutorials are important. The students need to learn the coding techniques, syntax etc but also understand the theory behind why they do things the way they do. Plus, as stated previously they really need to code as a hobby in between lessons. Not just do a lesson and leave it for a week until the next one.
- Using physical devices such as microbits, raspberry pi, etc. Programming for a real purpose that the students can get excited about. Too much of the qualification content is based on things that the students don't care about and cannot relate to - i.e. accounting, stock control, etc.
- Show and copy or ensure loads of practise is given to the student! In addition, if they fail then praising them and encourage them to try again

# Python Statements

## Variables:

`_` - Answer to the previous operation  
`[]` - Empty list  
`()` - Empty tuple  
`{}` - Empty dictionary

## Functions:

`help()` - Initiates the Python help utility  
`print(<string>)` - Outputs the given string. Multiple variables with different types can be printed on the same line by adding a comma between each variable  
`+ - * / %` - Mathematical operators for expressions (Add, subtract, multiply, divide, modulus). `+` is also used for concatenation  
`for <variable> in <iterable>`: - Stores each item from the Iterable into the given variable one at a time. For each time that this occurs, execute the indented block of code below  
`range(<initial value>, <final value>, <increment>)` - Creates a list (which is an Iterable) of all the values from the initial value to the final value in the given increments  
`> >= < <= != ==` - Comparison operators (Greater than, greater than or equal to, less than, less than or equal to, not equal, equal)  
`if <condition>`: - If the condition returns true, execute the indented block of code below. The block of code may attach an `elif` statement or an `else` statement at the end. Other conditional statements attached to this `if` statement are ignored once the block of code is executed  
`elif <condition>`: - If the conditions from previous `if` and `elif` statements return false and the conditions for this statement returns true, then the indented block of code below gets executed. Other conditional statements attached to this `elif` statement are ignored once the block of code is executed  
`<list variable name>.append(<value>)` - Appends the given value into the list as the final element  
`<list variable name>.insert(<index>, <value>)` - Inserts the given value at the `<index>+1` position of the list  
`len(<list variable name>)` - Returns the length of the given list  
`while <condition>`: - If the condition returns true, execute the indented block of code below. This is repeated until the condition no longer returns true  
`input(<string>)` - Outputs the given string and waits for an input from the user. The input given by the user is then returned in the form of a string  
`float(<input>)` - Converts the type of the input to a float number  
`int(<input>)` - Converts the type of the input to an integer  
`long(<input>)` - Converts the type of the input to a long number  
`str(<input>)` - Converts the type of the input to a string  
`open(<filename>, <r/w>')` - Depending on whether read/write is chosen ('r' for read and 'w' for write), the file with the given filename is opened for reading/writing  
`<file variable name>.write(<string>)` - Writes the string into the file with the given filename. The file must be open with the "write" modifier  
`<file variable name>.read()` - Returns the content of the entire file as a string. The file must be open with the "read" modifier  
`<file variable name>.readline()` - Returns the next line of the file as a string. The file must be open with the "read" modifier  
`<file variable name>.readlines()` - Returns the lines of the file as a list. The file must be open with the "read" modifier  
`<file variable name>.close()` - Closes the connection Python has with the file if the file is open



## Extra:

# is used at the start of a line to indicate that the line is used for commenting

pass is null statement, meaning that Python does nothing when executing this statement

**<list variable name>**[n-1] points to the nth element of the list, which can be used to return the value at that position or assign a value to that position of the list. Similarly, **<list variable name>**[a-1:b-1] points to the sublist between and including a and b in the given list. An element can be deleted by using del **<list variable name>**[**<element to delete-1>**]. A list can be created by using the square brackets [ and ].

A tuple is very similar to a list, with the main difference being that a list is mutable whereas a tuple is immutable (cannot be changed after creation). This means that a lot of the functions and manipulations for lists also applies to tuples except for deletion and reassignment of values to tuples. A tuple is created using the round brackets ( and ).

For dictionaries, the values are indexed by keys rather than by position numbers. They are initialised by supplying both the key and the corresponding value (i.e. {**<key>**:**<value>**}). **<dictionary variable name>**[**<key>**] points to the value with the given key, which can be used to return the value of the given key or assign a new value to that key. **<key>** in **<dictionary variable name>** checks the dictionary for the given key and returns a boolean on whether the key exists.

**<string variable name>**.split() splits a string into a list with each space signifying a new element.

def **<new function name>**(**<arguments>**): allows the user to create a new function with the required arguments listed. The body of the function is defined in the indented block below this statement. The function can return values using the statement 'return' followed by the variable values to return.

A try/except block is used to catch possible exceptions and then define what to do if such an exception is caught.

## References:

Beazley, D. M. (2000). 'An Introduction to Python', *O'Reilly Open Source Conference (17/7/21)*. Available at: [http://www.dabeaz.com/python/tutorial/beazley\\_intro\\_python/Slides/SLIDE001.HTM](http://www.dabeaz.com/python/tutorial/beazley_intro_python/Slides/SLIDE001.HTM) (Accessed: 10/4/21)

Fernandes, A. A. A. (2017). 'A Brief Python Tutorial', COMP18112: *Fundamentals of Distributed Systems*. University of Manchester. Unpublished.

Wichmann, M. (2019). *For Loops*. Available at: <https://wiki.python.org/moin/ForLoop> (Accessed: 10/4/21).

# MATLAB Statements

## Variables:

**ans** → answer to the previous statement  
**pi** → Value of pi  
**Inf** → Infinity (values larger than  $1.7 \times 10^{308}$ )  
**NaN** → Not a number  
**i** → Imaginary number  
**j** → Imaginary number (same as for i)

## Functions:

**sin(x), cos(x), tan(x)** → Trigonometric functions. 'x' is in radians  
**asin(x), acos(x), atan(x)** → Inverse trigonometric functions in radians  
**sqrt(x)** → Square root of 'x'  
**abs(x)** → absolute value of 'x'  
**exp(x)** → exponential function, where the function returns  $e^x$   
**log(x)** → natural logarithm of 'x'  
**log10(x)** → logarithm to base 10 of 'x'  
**rand** → Random number between 0 and 1  
**disp(x)** → Print out 'x', where 'x' could be a string or a number (remember to add **num2str(y)** or **int2str(y)** for a number y in a string. Also, square brackets ([]) are used to concatenate strings together)  
**sum(x)** → Calculates the sum of all the elements of the vector 'x'  
**mean(x)** → Calculates the mean of the elements of the vector 'x'  
**min(x)** → Finds the minimum element of the vector 'x'  
**max(x)** → Find the maximum element of the vector 'x'  
**linspace(x, y, z)** → Creates exactly 'z' amount of numbers between 'x' and 'y' (inclusive)  
**length(x)** → Returns the length of the vector 'x'  
**end** → Used as **x(end)**, where end takes the index of the last element of the vector that it is currently being used in (i.e. last index of the vector 'x')  
**ones(x, y)** → Creates an array with x rows and y columns, all with the value '1'  
**plot(x, y)** → Creates a graph where 'y' is plotted against 'x' on the current figure (**title(x)**, **ylabel(x)** and **xlabel(x)** can only be used after the graph has been plotted)  
**title(x)** → Sets the title of the graph of the current figure with the string 'x'  
**ylabel(x)** → Sets the y axis label of the graph of the current figure with the string 'x'  
**xlabel(x)** → Sets the x axis label of the graph of the current figure with the string 'x'  
**roots(x)** → Returns the roots of the vector 'x', where 'x' is an equation in vector form  
**poly(x)** → Returns the equation which has roots given by the vector 'x' in vector form (basically does the opposite of **roots(x)**)  
**semilogx** → Logs the x axis  
**semilogy** → Logs the y axis  
**loglog** → Logs both of the axes  
**find(x)** → Returns a vector of indices of values that satisfy 'x', where 'x' is a series of conditions for one or more vectors. The vectors considered are included in the condition (e.g. **y < 0** searches for values within the vector 'y' that are less than 0)  
**inv(x)** → Returns the inverse of the matrix 'x'

## Loops and Conditional Statements:

**for a=x** → A for loop where 'a' is a variable and 'x' is a vector, typically a vector of 1:y, where 'y' is an integer. The loop body is iterated 'y' times in this case. The loop body needs to be ended with **end**

**if <a condition>** → An if statement. The relational operators that can be used for the condition is ==, >, >=, <, <=, ~=, | and & can also be used, but each condition would need to be in separate brackets. The if statement may also have an else statement. If that is the case, there only needs to be an **end** after the else statement; otherwise, there needs to be an **end** after the if statement

**While <a condition>** → A while loop. The same operators can be used for the condition in the while loop as for the if statement condition. There needs to be an **end** after the while loop

## Others:

To create a string, use apostrophes (e.g. 'Hello') to surround the characters. To add an apostrophe (') to a string, use two apostrophes ('').

A semicolon at the end of the statement prevents Matlab from printing out the answer of the variable that was just assigned a value (e.g. **a=10;**).

A variable can be assigned an empty array/vector (e.g. **a=[]**). A vector is assigned by surrounding the values with square brackets (e.g. **a=[1 2 3 4]**). Assigning a column vector is done similarly, but with semi-colons in between each value (e.g. **a=[1; 2; 3; 4]**). For matrices, a similar method is used (e.g. [1 2 3 4; 5 6 7 8] for a 2x4 matrix with the elements specified).

A vector/matrix can be transposed by using the .' operator. .\* and ./ are the element-wise operators for vectors/matrices. Using \* for vectors is equivalent to using vector multiplication, and using \* for matrices is equivalent to using matrix multiplication. Vectors can be raised to a power by using the .^ operator. For matrices, .^ is used for raising each element of the matrix by a power, whereas ^ is used for raising the entire matrix by a power.

Vectors containing all the integers from 'x' to 'y' can be created by using x:y (e.g. **a=1:10** for a vector containing all integers starting from 1 to 10). An increment 'z' can be added using x:z:y (e.g. **a=1:2:10** creates a vector with all the odd numbers from 1 to 10, starting with 1).

The element at index 'x' of a vector 'v' can be accessed by using **v(x)**. Multiple elements can be accessed at the same time by having the index be a vector (e.g. **v(x)** or **v([1 2])**). Using the same method, multiple elements can be assigned the same value (e.g. **v([1 2 3]) = 1**). An element can be deleted by assigning the empty array ([]) to that element.

Concatenation is done using square brackets.

**Figure** is used to generate a new figure, and **hold on** is used to plot more curves on the same figure without overwriting the previous graph. **grid on** is used to turn on the grid of the graph. **[a, b]=ginput** is used to pinpoint the coordinates of a point of the graph by clicking on said point and pressing 'Enter' after that. The x and y coordinates are then saved onto the variables **a** and **b** respectively.

For a condition, vectors as well as scalars can be compared; for vectors, the elements at the corresponding index are compared (e.g. [a<sub>1</sub> a<sub>2</sub> a<sub>3</sub>] > [b<sub>1</sub> b<sub>2</sub> b<sub>3</sub>] would mean that Matlab would compare a<sub>n</sub> > b<sub>n</sub> for n=1, 2, 3). This also means that only vectors with the same number of elements can be compared.

To create a function, the script must begin with the line **function x = y(v<sub>1</sub>, v<sub>2</sub>, ... , v<sub>n</sub>)**, where 'x' is the output variable (which could be a vector. In that case, it would be [x<sub>1</sub>, x<sub>2</sub>, ... , x<sub>n</sub>]), 'y' is the name of the function and v<sub>1</sub>, v<sub>2</sub>, ... , v<sub>n</sub> are the parameters passed on to this function. The script file has to have 'y' as its name (the name of the function).



## References:

- Bonello, P. (2018). 'Tutorial 1a: Performing some simple tasks', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished.
- Bonello, P. (2018). 'Tutorial 1b: More about variables', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished.
- Bonello, P. (2018). 'Tutorial 2: Vectors', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished.
- Bonello, P. (2018). 'Tutorial 3: Use of vectors', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished.
- Bonello, P. (2018). 'Tutorial 4: Loops and conditional statements', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished.
- Bonello, P. (2018). 'Tutorial 5: Writing your own functions – an Introduction', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished.
- Bonello, P. (2018). 'Tutorial 6: Introduction to matrices', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished.
- Leung, K. K. (2018). 'Matlab Statements', *MACE12101: Tools for Engineers*. University of Manchester. Unpublished essay/assignment

## Appendix E - Main Menu Code

```
# Created by Kin Leung (27th April of 2021) for the Individual Project
# titled 'Programming for Engineers: Removing Barriers and Improving Outcomes'.
# This piece of code incorporates all the separate pieces of code into one,
# forming the full software. (Python Software Foundation, 2021).
```

```
import Questions
```

```
# Define a function for printing out all the list of actions available to the
# user.
```

```
def print_actions():
```

```
    print("1. Fill In the Gap")
```

```
    print("2. Fill In the Gap (MATLAB)")
```

```
    print("3. Correct the Errors")
```

```
    print("4. Program Writing")
```

```
    print("5. Program Writing 2")
```

```
    print("6. Exit")
```

```
# Begin by welcoming the user to the software and reveal to them the possible
# actions they can choose.
```

```
print("Hello! Welcome to the Automatic Revision Tool (ART).")
```

```
print("Main Menu")
```

```
print_actions()
```

```
# Let the user choose.
```

```
chosen_action = input("Please input the number for what you would like")
```

```
    " to do: ")

# Evaluate which number the student choose. Invoke the corresponding
# actions that has been chosen.

if chosen_action == '1':

    Questions.Fill_In_The_Gap()

elif chosen_action == '2':

    Questions.Fill_In_The_Gap_MATLAB()

elif chosen_action == '3':

    Questions.Correct_The_Errors()

elif chosen_action == '4':

    Questions.Program_Writing()

elif chosen_action == '5':

    Questions.Program_Writing_Two()

elif chosen_action == '6':

    exit()

# Python Software Foundation (2021). 6. Modules. Available at:
# https://docs.python.org/3/tutorial/modules.html#modules (Accessed: 29/4/21).
```

## Appendix F - Questions Module Code

```
# Created by Kin Leung (27th April of 2021) for the Individual Project titled
# 'Programming for Engineers: Removing Barriers and Improving Outcomes'. This
# module encompasses several types of questions (in the form of functions) that
# can be used to help the student revise.
```

```
# Created by Kin Leung (8th February of 2021) for the Individual Project titled
# 'Programming for Engineers: Removing Barriers and Improving Outcomes'.
# This piece of code takes a text file which contains a specification of what
# it does at the start of the file as well as an incomplete piece of code. What
# the student has to do is input the correct Python code which gets executed to
# find the correct value (which is also contained in the file). This allows
# this particular piece of code to be reusable for different text files.
# Create a variable to store the text file's name, which is defaulted to be
# InputFile.txt.
```

```
def Fill_In_The_Gap():
```

```
    text_file_name = "InputFile.txt"
```

```
    # Tell the students what the purpose of this code is for.
```

```
    print("Fill In the Gap")
```

```
    print("The following is an incomplete piece of code. You need to type in a",
```

```
          "line of code with the correct syntax to fill in the gap, which will",
```

```
          "then be executed. The answer which is obtained will then be compared",
```

```
          "with the answer provided by the text file.")
```

```

# Open the text file and take all the lines (W3Schools, 2021).
text_file = open(text_file_name, 'r')
text_lines = text_file.readlines()
text_file.close()

# Create variables to store the answer and the line at which the answer is
# given as well as the line which contains the >gap< tag and the line where the
# gap is.
given_answer = None
answer_line = None
line_to_fill = None
gap_line = None

# Check the lines for the >ans< and >gap< tags, which denotes the expected
# final answer and the line which the student needs to fill respectively.
for index in range(len(text_lines)):
    if ">ans<" in text_lines[index]:
        # Extract the final answer from the line and keep track of which line
        # this is.
        given_answer = text_lines[index].replace(">ans<", "")
        answer_line = index
    if ">gap<" in text_lines[index]:
        # Do the same for the gap.
        line_to_fill = text_lines[index].replace(">gap<", "")
        gap_line = index

```



```

# Assuming that the given values are integers, convert both of the extracted
# values to int type. line_to_fill needs to be an index, hence needs to
# subtract by 1.
given_answer = int(given_answer)
line_to_fill = int(line_to_fill) - 1

# Delete the lines containing the >ans< and >gap< tags.
del text_lines[answer_line]
if answer_line < gap_line:
    del text_lines[gap_line - 1]
else:
    del text_lines[gap_line]

# Adjust the value in line_to_fill as two lines have now been deleted.
if answer_line < line_to_fill and gap_line < line_to_fill:
    line_to_fill = line_to_fill - 2
elif ((answer_line < line_to_fill < gap_line)
      or (gap_line < line_to_fill < answer_line)):
    line_to_fill = line_to_fill - 1

# Output the remainder of the text file to the student.
for index in range(len(text_lines)):
    print(text_lines[index])

```

```

# Eliminate the comments from the list by searching for # at the beginning
# of the line starting from the end of the list so that the indices do not
# change when the comments are being deleted.
for index in range(len(text_lines) - 1, -1, -1):
    if text_lines[index].find("#") == 0:
        del text_lines[index]

# Create a variable to store a boolean for the while loop which allows multiple
# attempts of the same problem. Defaults to false.
answer_is_correct = False

# Use a while loop to allow multiple submissions.
while not answer_is_correct:
    # Ask the student for the omitted line of code, appending a newline
    # character at the end.
    student_answer = input("Please input your line of code:\n") + '\n'

# Create a variable to store the string containing the list statements as a
# single string, which is initially empty.
answer_string = ""

# Convert the list into a single string.
for index in range(len(text_lines)):
    # If the index is the same as the line_to_fill value, then inject the
    # student's code before adding the remainder of the statements from the

```

```

# list.

if line_to_fill == index:

    answer_string = answer_string + student_answer

answer_string = answer_string + text_lines[index]

# Try to execute the string (khelwood, 2018).

try:

    exec(answer_string, globals())

    # If the code was successfully executed, then the final answer is in

    # final_answer. Compare that value to given_answer.

    if final_answer == given_answer:

        # Congratulate the student for giving the correct statement and set

        # the boolean answer_is_correct to true.

        print("Well done! The final answer is correct!")

        answer_is_correct = True

    # If the answer was not correct, then report that to the student.

    else:

        print("Unfortunately, your answer did not provide the correct",

              "result after execution. Please try again.")

# If this causes an error, then the line of code (either syntax, variable

# name or otherwise is wrong). All the errors are caught here.

# (Python Software Foundation, 2021d).

except SyntaxError:

    print("The syntax for your line of code was not correct. Please try",

          "again.")

```

```

except NameError as error:

    print("A NameError has been raised by the parser. This was the",

          "message:")

    print(error)

    print("Please try again.")

except IndexError:

    print("Your line of code caused the index of a statement to be out of",

          "bounds. Please try again.")

# If there are other errors, then catch it and report that an unknown error
# has occurred. (Python Software Foundation, 2021d).

except Exception as error:

    print("An unknown error has occurred. This was the message passed by",

          "the parser:")

    print(error)

    print("Please try again.")

# Created by Kin Leung (8th February of 2021) for the Individual Project titled
# 'Programming for Engineers: Removing Barriers and Improving Outcomes'.
# This piece of code takes a text file which contains a specification of what
# it does at the start of the file as well as an incomplete piece of code. What
# the student has to do is input the correct Python code which gets executed to
# find the correct value (which is also contained in the file). This allows
# this particular piece of code to be reusable for different text files.
# (Mathworks Inc., 2021b).

def Fill_In_The_Gap_MATLAB():

```

```
import matlab.engine

# Create a variable to store the text file's name, which is defaulted to be
# InputFile.txt.
text_file_name = "InputFile template.txt"

# Tell the students what the purpose of this code is for.
print("Fill In the Gap (MATLAB)")
print("The following is an incomplete piece of code. You need to type in a",
      "line of code with the correct syntax to fill in the gap, which will",
      "then be executed. The answer which is obtained will then be compared",
      "with the answer provided by the text file.")

# Open the text file and take all the lines (W3Schools, 2021).
text_file = open(text_file_name, 'r')
text_lines = text_file.readlines()
text_file.close()

# Create variables to store the answer and the line at which the answer is
# given as well as the line which contains the >gap< tag and the line where the
# gap is.
given_answer = None
answer_line = None
line_to_fill = None
gap_line = None
```

```

# Check the lines for the >ans< and >gap< tags, which denotes the expected
# final answer and the line which the student needs to fill respectively.
for index in range(len(text_lines)):
    if ">ans<" in text_lines[index]:
        # Extract the final answer from the line and keep track of which line
        # this is.
        given_answer = text_lines[index].replace(">ans<", "")
        answer_line = index
    if ">gap<" in text_lines[index]:
        # Do the same for the gap.
        line_to_fill = text_lines[index].replace(">gap<", "")
        gap_line = index

# Assuming that the given values are integers, convert both of the extracted
# values to int type. line_to_fill needs to be an index, hence needs to
# subtract by 1.
given_answer = int(given_answer)
line_to_fill = int(line_to_fill) - 1

# Delete the lines containing the >ans< and >gap< tags.
del text_lines[answer_line]
if answer_line < gap_line:
    del text_lines[gap_line - 1]
else:

```

```

del text_lines[gap_line]

# Adjust the value in line_to_fill as two lines have now been deleted.
if answer_line < line_to_fill and gap_line < line_to_fill:

    line_to_fill = line_to_fill - 2
elif ((answer_line < line_to_fill < gap_line)
      or (gap_line < line_to_fill < answer_line)):

    line_to_fill = line_to_fill - 1

# Output the remainder of the text file to the student.
for index in range(len(text_lines)):

    print(text_lines[index])

# Eliminate the comments from the list by searching for # at the beginning
# of the line starting from the end of the list so that the indices do not
# change when the comments are being deleted.
for index in range(len(text_lines) - 1, -1, -1):

    if text_lines[index].find("#") == 0:

        del text_lines[index]

# Create a variable to store a boolean for the while loop which allows multiple
# attempts of the same problem. Defaults to false.
answer_is_correct = False

# Use a while loop to allow multiple submissions.

```

```

while not answer_is_correct:

    # Ask the student for the omitted line of code, appending a newline
    # character at the end.

    student_answer = input("Please input your line of code:\n") + '\n'

    # Create a variable to store the string containing the list statements as a
    # single string, which initially contains only the MATLAB function
    # declaration.

    answer_string = "function final_answer = studentanswer()\n"

    # Convert the list into a single string.

    for index in range(len(text_lines)):

        # If the index is the same as the line_to_fill value, then inject the
        # student's code before adding the remainder of the statements from the
        # list.

        if line_to_fill == index:

            answer_string = answer_string + student_answer

            answer_string = answer_string + text_lines[index]

    # Create a new file and store the contents of the answer_string to the
    # file. Afterward, close the link (W3Schools, 2021).

    testing_file = open("studentanswer.m", 'w')

    testing_file.write(answer_string)

    testing_file.close()

```



```

# Start the MATLAB engine (Mathworks Inc., 2021b).
engine = matlab.engine.start_matlab()

# Try to execute the file.
try:
    # (Mathworks Inc., 2021a).
    final_answer = engine.studentanswer()

    # If the code was successfully executed, then the final answer is in
    # final_answer. Compare that value to given_answer.
    if final_answer == given_answer:
        # Congratulate the student for giving the correct statement and set
        # the boolean answer_is_correct to true.
        print("Well done! The final answer is correct!")
        answer_is_correct = True

    # If the answer was not correct, then report that to the student.
    else:
        print("Unfortunately, your answer did not provide the correct",
              "result after execution. Please try again.")

    # If this causes an error, then the line of code (either syntax, variable
    # name or otherwise is wrong). All the errors are caught here.
    # (Python Software Foundation, 2021d).
    except SyntaxError:
        print("The syntax for your line of code was not correct. Please try",
              "again.")
    except NameError as error:

```

```

print("A NameError has been raised by the parser. This was the",
      "message:")
print(error)
print("Please try again.")
except IndexError:
    print("Your line of code caused the index of a statement to be out of",
          "bounds. Please try again.")
# If there are other errors, then catch it and report that an unknown error
# has occurred (Python Software Foundation, 2021d).
except Exception as error:
    print("An unknown error has occurred. This was the message passed by",
          "the parser:")
    print(error)
    print("Please try again.")
finally:
    # Exit the MATLAB engine no matter if the code was successfully
    # executed (Mathworks Inc., 2021a).
    engine.quit()

# Created by Kin Leung (14th February of 2021) for the Individual Project
# titled 'Programming for Engineers: Removing Barriers and Improving Outcomes'.
# This piece of code takes a text file which contains a specification of what
# it does at the start of the file as well as a piece of code which contains
# errors. What the student has to do is copy the code and then correct all the
# errors and exceptions that arise. A final answer will be given once all the

```

```
# errors and exceptions have been corrected, and the student has to input the
# final answer value. This will be used to determine if the student succeeded.
# This piece of code can be reused for different text files.
# Create a variable to store the text file's name, which is defaulted to be
# ErrorInputFile.txt.
def Correct_The_Errors():
    text_file_name = "ErrorInputFile.txt"

    # Tell the students what the purpose of this code is for.
    print("Correct the Errors")
    print("The following is an attempt at writing a piece of code for solving a",
          "problem. Sadly, the code would not run due to various errors and",
          "exceptions. Please check the specifications that have been given, copy",
          "the code into a separate file and fix it. Once the code runs, a",
          "final answer will be given. Input that final answer below for marking.")

    # Open the text file and take all the lines (W3Schools, 2021).
    text_file = open(text_file_name, 'r')
    text_lines = text_file.readlines()
    text_file.close()

    # Create variables to store the answer and the line at which the answer is
    # given.
    given_answer = None
    answer_line = None
```

```
# Check the lines for the >ans< tag, which denotes the expected final answer.
```

```
for index in range(len(text_lines)):
```

```
    if ">ans<" in text_lines[index]:
```

```
        # Extract the final answer from the line and keep track of which line
```

```
        # this is.
```

```
        given_answer = text_lines[index].replace(">ans<", "")
```

```
        answer_line = index
```

```
# Delete the line with the answer.
```

```
del text_lines[answer_line]
```

```
# Assuming that the given value is an integer, convert the extracted value
```

```
# to int type.
```

```
given_answer = int(given_answer)
```

```
# Output the remainder of the text file to the student.
```

```
for index in range(len(text_lines)):
```

```
    print(text_lines[index])
```

```
# Create a variable to store a boolean for the while loop which allows multiple
```

```
# attempts of the same problem. Defaults to false.
```

```
answer_is_correct = False
```

```
# Use a while loop to allow multiple submissions.
```

```

while not answer_is_correct:

    # Ask the student for the final answer.

    student_answer = input("Please input the final answer: ")

    # Check whether the answer is valid and whether the answer is correct.

    try:

        if int(student_answer) == given_answer:

            # Congratulate the student for giving the correct answer and set

            # the boolean answer_is_correct to true.

            print("Well done! The final answer is correct!")

            answer_is_correct = True

        # If the answer was not correct, then report that to the student.

        else:

            print("Unfortunately, your answer was not correct. Please try",

                  "again.")

        # If the answer did not have a correct type, then tell the student.

        # (Python Software Foundation, 2021d).

    except ValueError as error:

        print("You did not provide a valid answer. Remember to provide the",

              "final answer as digits to the nearest whole number. Please try",

              "again")

        # If there are other errors, then catch it and report that an unknown error

        # has occurred (Python Software Foundation, 2021d).

    except Exception as error:

        print("An unknown error has occurred. This was the message passed by",

```

```

        "the parser:")

    print(error)

    print("Please try again.")

# Created by Kin Leung (6th February of 2021) for the Individual Project titled
# 'Programming for Engineers: Removing Barriers and Improving Outcomes'.
# This piece of code generates random values of the forces and masses and tells
# the student to write a program based on the scenario to calculate the
# acceleration. A timer of 15 seconds to input all the forces and masses is
# set as well as another 15 seconds to answer three randomised questions.
# The timer can be adjusted by changing the variables below. (Python Software
# Foundation, 2021a; 2021b; 2021c).

def Program_Writing():

    import random

    import threading

    import time

    # Define a function to get the answers from the student.

    def ask_for_answers(F_list, P_list, M_list, index_for_f, index_for_p,
                       index_for_m, given_answers, time_is_up_event):

        # Ask the student for the answers to the randomised questions and append it
        # to the supplied list.

        for index in range(3):

            (given_answers

             .append(input(("What is the acceleration when F is {0}, P is {1} and "

```

```

        "M is {2}? ").format(F_list[index_for_f[index]],
                            P_list[index_for_p[index]],
                            M_list[index_for_m[index]])))))

# If the deadline as already passed, then terminate the thread.

# (Python Software Foundation, 2021b).

if time_is_up_event.is_set():
    exit()

# Specify that the timer for inputting the forces and masses is 20 seconds

# and the timer for answering the three questions is 15 seconds.

first_timer_time = 20

second_timer_time = 15

# Introduce to the student the problem and the specification that they need to
# write their program.

print("Program Writing")

print("-----")

print("| M |--> F,P")

print("-----")

print("Consider a two locomotive train. Let F and P be the two forces",
      "generated by the engines of the two locomotives in Newtons and M be the",
      "combined mass of the locomotives in kilograms. There will be three",
      "different values for each of F, P and M. Write a program that can",
      "evaluate all the possible values of the acceleration in metres per",
      "second squared. You will have 20 seconds to input all the numbers in",

```

```
"and 15 more seconds to answer the three randomised questions at the",  
"end.")
```

```
# Introduce a string variable for checking the input, which is an empty string.
```

```
input_string = ""
```

```
# The program will only continue if the student types in "Continue".
```

```
while input_string!="Continue":
```

```
    input_string = input("Please enter \"Continue\" when you are ready: ")
```

```
# Create lists to store the three values each of F, P and M.
```

```
F = []
```

```
P = []
```

```
M = []
```

```
# Generate 6 random values between 10000 and 500000 and and 3 random  
values
```

```
# between 90000 and 200000, and assign them to be either F, P or M. Round  
each
```

```
# value to two decimal place. (Python Software Foundation, 2021c;
```

```
# Worldwide Rails, 2021).
```

```
for index in range(3):
```

```
    F.append(round(random.uniform(10000,500000), 2))
```

```
    P.append(round(random.uniform(10000,500000), 2))
```

```
    M.append(round(random.uniform(180000,400000), 2))
```



```

# Create three lists to store the indices that will be used to generate the
# three randomised questions and another list to store the answers.

f_index = []
p_index = []
m_index = []
acceleration = []

# Generate 9 random integers between 0 and 2 to generate the three randomised
# questions to ask the student. Calculate the answer using the indices and
# append it to the acceleration list to two decimal places (Python Software
# Foundation, 2021c).
for index in range(3):
    f_index.append(random.randint(0, 2))
    p_index.append(random.randint(0, 2))
    m_index.append(random.randint(0, 2))
    answer = (F[f_index[index]] + P[p_index[index]]) / M[m_index[index]]
    acceleration.append(round(answer, 2))

# Reveal the numbers to the student and tell them to not press anything.
print("You have 20 seconds to enter the numbers into your program. The",
      "numbers are:")
print("F1:", F[0], " F2:", F[1], " F3:", F[2])
print("P1:", P[0], " P2:", P[1], " P3:", P[2])
print("M1:", M[0], " M2:", M[1], " M3:", M[2])

```

```

print("Please do NOT input anything here.")

# Sleep for 20 seconds before continuing (Python Software Foundation, 2021a).
time.sleep(first_timer_time)

# Create a list variable to store the answers given by the student, which is
# empty at the start. Create an Event to store a boolean which is used to
# check if the 15 seconds were up first or if the student inputted the answers
# first (Python Software Foundation, 2021b).
student_answers = []
time_is_up_event = threading.Event()

# Start a thread which asks the student for the answers.
# (Python Software Foundation, 2021b).
answer_thread = threading.Thread(target = ask_for_answers,
                                args = (F, P, M, f_index, p_index,
                                        m_index, student_answers,
                                        time_is_up_event))
answer_thread.start()

# Create a variable to store the current time and add 15 to it for comparison.
deadline = time.time() + 15

# Constantly check the time to ensure that the inputs are given in before the
# 15 seconds have elapsed (Python Software Foundation, 2021b).

```

```

while answer_thread.is_alive():

    # If it has been 15 seconds, then set the boolean to true and wait for the
    # thread to terminate. Tell the student to terminate the thread.

    # (Python Software Foundation, 2021a; 2021b).

    if deadline < time.time():

        time_is_up_event.set()

        print("\nTime is up! Press the \"Enter\" key to continue")

        answer_thread.join()

# If the answers were not answered on time, tell the student that it was not
# done on time (Python Software Foundation, 2021b).

if time_is_up_event.is_set():

    print("Unfortunately, you did not answer the questions on time.")

# Otherwise, mark the questions for the student.

else:

    print("Marking... Please wait...")

    # Check that the answers given are float numbers and convert them.

    try:

        for index in range(3):

            student_answers[index] = float(student_answers[index])

            # Create a list to keep track of which questions were answered
            # correctly. By default, all values are false. Additionally, create a
            # variable to keep track of the marks. By default, the marks are 0.

            correct_answers = [False, False, False]

            marks = 0

```

```

# Check the student's answers compared to the calculated answers.
# Allow a tolerance of +/-0.01 for each of the answers.
for index in range(3):
    if ((acceleration[index] - 0.01) <= student_answers[index]
        <= (acceleration[index] + 0.01)):
        correct_answers[index] = True
        marks = marks + 1

# Report the marks to the student as well as which answers were
# incorrect if they had any. Congratulate them if all the answers were
# correct.
print("Your marks are", marks, "out of 3")
if correct_answers[0] and correct_answers[1] and correct_answers[2]:
    print("Well done!")
else:
    for index in range(3):
        if not correct_answers[index]:
            print(("The correct answer for question {0} was {1} whilst"
                " you inputted {2}")
                .format(index + 1, acceleration[index],
                    student_answers[index]))

# If they are not float numbers, tell the student that the answers are not
# valid (Python Software Foundation, 2021d).
except ValueError:
    print("You did not give valid answers.")

# If there are other errors, then catch it and report that an unknown error

```

```

# has occurred (Python Software Foundation, 2021d).
except Exception as error:
    print("An unknown error has occurred. This was the message passed by",
          "the parser:")
    print(error)
    print("Please try again.")

# Created by Kin Leung (16th February of 2021) for the Individual Project
# titled 'Programming for Engineers: Removing Barriers and Improving Outcomes'.
# This piece of code generates random values of the displacement (s), initial
# velocity (u) and time (t) and tells the student to write a program based on
# the scenario to calculate the acceleration. A timer of 15 seconds to input
# all the displacements, initial velocities and times is set as well as another
# 15 seconds to answer three randomised questions. The timer can be adjusted
# by changing the variables below (Python Software Foundation, 2021a; 2021b;
# 2021c).
def Program_Writing_Two():
    import random
    import threading
    import time

    # Define a function to get the answers from the student.
    def ask_for_answers(s_list, u_list, t_list, index_for_s, index_for_u,
                       index_for_t, given_answers, time_is_up_event):
        # Ask the student for the answers to the randomised questions and append it

```

```

# to the supplied list.
for index in range(3):
    (given_answers
     .append(input(("What is the acceleration when s is {0}, u is {1} and "
                    "t is {2}? ").format(s_list[index_for_s[index]],
                                         u_list[index_for_u[index]],
                                         t_list[index_for_t[index]]))))

# If the deadline as already passed, then terminate the thread.
# (Python Software Foundation, 2021b).
if time_is_up_event.is_set():
    exit()

# Specify that the timers for inputting the displacements, initial velocities
# and times and for answering the three questions are 15 seconds.
first_timer_time = 15
second_timer_time = 15

# Introduce to the student the problem and the specification that they need to
# write their program.
print("Program Writing 2")
print("A car travelled s metres in a straight line at an initial velocity of u",
      "metres per second in t seconds. There will be three different values",
      "for each of s, u and t. Write a program that can evaluate all the",
      "possible values of the acceleration in metres per second squared. You",
      "will have 15 seconds to input all the numbers in and 15 more seconds",

```

"to answer the three randomised questions at the end.")

# Introduce a string variable for checking the input, which is an empty string.

```
input_string = ""
```

# The program will only continue if the student types in "Continue".

```
while input_string!="Continue":
```

```
    input_string = input("Please enter \"Continue\" when you are ready: ")
```

# Create lists to store the three values each of s, u and t.

```
s = []
```

```
u = []
```

```
t = []
```

# Generate 9 random values between 5 and 50 and assign them to be either s, u

# or t. Round each value to two decimal place (Python Software Foundation,

# 2021c).

```
for index in range(3):
```

```
    s.append(round(random.uniform(5,50), 2))
```

```
    u.append(round(random.uniform(5,50), 2))
```

```
    t.append(round(random.uniform(5,50), 2))
```

# Create three lists to store the indices that will be used to generate the

# three randomised questions and another list to store the answers.

```
s_index = []
```

```

u_index = []
t_index = []
acceleration = []

# Generate 9 random integers between 0 and 2 to generate the three randomised
# questions to ask the student. Calculate the answer using the indices and
# append it to the acceleration list to two decimal places (Python Software
# Foundation, 2021c).
for index in range(3):
    s_index.append(random.randint(0, 2))
    u_index.append(random.randint(0, 2))
    t_index.append(random.randint(0, 2))
    answer = ((2 * (s[s_index[index]] - u[u_index[index]] * t[t_index[index]]))
              / (t[t_index[index]] * t[t_index[index]]))
    acceleration.append(round(answer, 2))

# Reveal the numbers to the student and tell them to not press anything.
print("You have 15 seconds to enter the numbers into your program. The",
      "numbers are:")
print("s1:", s[0], " s2:", s[1], " s3:", s[2])
print("u1:", u[0], " u2:", u[1], " u3:", u[2])
print("t1:", t[0], " t2:", t[1], " t3:", t[2])
print("Please do NOT input anything here.")

# Sleep for 15 seconds before continuing (Python Software Foundation, 2021a).

```



```

time.sleep(first_timer_time)

# Create a list variable to store the answers given by the student, which is
# empty at the start. Create an Event to store a boolean which is used to
# check if the 15 seconds were up first or if the student inputted the answers
# first (Python Software Foundation, 2021b).
student_answers = []
time_is_up_event = threading.Event()

# Start a thread which asks the student for the answers.
# (Python Software Foundation, 2021b).
answer_thread = threading.Thread(target = ask_for_answers,
                                args = (s, u, t, s_index, u_index,
                                         t_index, student_answers,
                                         time_is_up_event))
answer_thread.start()

# Create a variable to store the current time and add 15 to it for comparison.
deadline = time.time() + 15

# Constantly check the time to ensure that the inputs are given in before the
# 15 seconds have elapsed (Python Software Foundation, 2021b).
while answer_thread.is_alive():
    # If it has been 15 seconds, then set the boolean to true and wait for the
    # thread to terminate. Tell the student to terminate the thread.

```

```

# (Python Software Foundation, 2021a; 2021b).

if deadline < time.time():

    time_is_up_event.set()

    print("\nTime is up! Press the \"Enter\" key to continue")

    answer_thread.join()

# If the answers were not answered on time, tell the student that it was not
# done on time (Python Software Foundation, 2021b).

if time_is_up_event.is_set():

    print("Unfortunately, you did not answer the questions on time.")

# Otherwise, mark the questions for the student.

else:

    print("Marking... Please wait...")

    # Check that the answers given are float numbers and convert them.

    try:

        for index in range(3):

            student_answers[index] = float(student_answers[index])

            # Create a list to keep track of which questions were answered
            # correctly. By default, all values are false. Additionally, create a
            # variable to keep track of the marks. By default, the marks are 0.

            correct_answers = [False, False, False]

            marks = 0

            # Check the student's answers compared to the calculated answers.

            # Allow a tolerance of +/-0.01 for each of the answers.

            for index in range(3):

```

```

if ((acceleration[index] - 0.01) <= student_answers[index]
    <= (acceleration[index] + 0.01)):
    correct_answers[index] = True
    marks = marks + 1

# Report the marks to the student as well as which answers were
# incorrect if they had any. Congratulate them if all the answers were
# correct.

print("Your marks are", marks, "out of 3")

if correct_answers[0] and correct_answers[1] and correct_answers[2]:
    print("Well done!")

else:
    for index in range(3):
        if not correct_answers[index]:
            print(("The correct answer for question {0} was {1} whilst"
                " you inputted {2}")
                .format(index + 1, acceleration[index],
                    student_answers[index]))

# If they are not float numbers, tell the student that the answers are not
# valid (Python Software Foundation, 2021d).

except ValueError:
    print("You did not give valid answers.")

# If there are other errors, then catch it and report that an unknown error
# has occurred (Python Software Foundation, 2021d).

except Exception as error:
    print("An unknown error has occurred. This was the message passed by",

```

```
    "the parser:")
print(error)
print("Please try again.")

# khelwood (2018). Setting variables with exec inside a function. Available at:
# https://stackoverflow.com/questions/23168282/setting-variables-with-exec
# -inside-a-function (Accessed: 29/4/21).
# Mathworks Inc. (2021a). Call User Scripts and Functions from Python. Available
# at: https://uk.mathworks.com/help/matlab/matlab_external/call-user-script-and
# -function-from-python.html (Accessed: 29/4/21).
# Mathworks Inc. (2021b). Install MATLAB Engine API for Python. Available at:
# https://uk.mathworks.com/help/matlab/matlab_external/install-the-matlab
# -engine-for-python.html (Accessed: 29/4/21).
# Python Software Foundation (2021a). time — Time access and conversions.
# Available at: https://docs.python.org/3/library/time.html (Accessed: 30/4/21).
# Python Software Foundation (2021b). threading — Thread-based parallelism.
# Available at: https://docs.python.org/3/library/threading.html
# (Accessed: 30/4/21)
# Python Software Foundation (2021c). random — Generate pseudo-random
numbers.
# Available at: https://docs.python.org/3/library/random.html
# (Accessed: 30/4/21).
# Python Software Foundation (2021d). 8. Errors and Exceptions. Available at:
# https://docs.python.org/3/tutorial/errors.html (Accessed: 29/4/21).
# Worldwide Rails (2021). How Much Do Locomotives Cost?. Available at:
```

# <https://worldwiderails.com/how-much-do-locomotives-cost/>  
#:~:text=How%20much%  
# 20does%20a%20locomotive,also%20heavier%20than%20DC%20locomotives.  
# (Accessed: 30/4/21).  
# W3Schools (2021). Python File Write. Available at:  
# [https://www.w3schools.com/python/python\\_file\\_write.asp](https://www.w3schools.com/python/python_file_write.asp) (Accessed: 29/4/21).

## Appendix G - InputFile.txt Code

```
# The given code is for calculating the impulse of an object A on an object B
# using Newton's Law of Restitution and the Conservation of Momentum. The mass
# of A is given as 5kg, and the initial velocities are given as 12 metres
# per second and -4 metres per second for A and B respectively. The velocity
# of A after collision is -6 metres per second. The coefficient of restitution
# is 5/6. Initially, the Law of Restitution is used to find the velocity of
# B after the collision, and conservation of momentum is then used to find the
# mass of B. The impulse is then calculated as the change in momentum of B,
# which is stored in final_answer.

>ans< 90

>gap< 5

mass = {'a':5}

a_velocities = [12, -6]

b_velocities = [-4]

e = 5/6

# Your line will be added here

mass['b'] = (-mass['a'] * a_velocities[0] + a_velocities[1] * mass['a']) / (b_velocities[0]
- b_velocities[1])

final_answer = mass['b'] * (b_velocities[1] - b_velocities[0])
```

## Appendix H - Test Data and Outcome

### Fill In the Gap:

- `->` Error
- `1` `->` IndexError
- `Test` `->` NameError
- `print "Test"` `->` SyntaxError
- `open("Test")` `->` Error
- `b_velocities.append(-e * (b_velocities[0] - a_velocities[0]) + a_velocities[1])`  
`->` Correct answer

### Fill In the Gap (MATLAB):

- `->` Error
- `1` `->` Error
- `Test` `->` Error
- `disp("Test")` `->` Error
- `c=sqrt(4)` `->` Incorrect answer statement
- `c=b*30` `->` Correct answer statement

### Correct the Errors:

- `->` Not valid statement
- `1` `->` Answer incorrect statement
- `Test` `->` Not valid statement
- `print("Test")` `->` Not valid statement
- `3.5677` `->` Not valid statement
- `9744803452` `->` Correct answer statement

- 5.456465.4 → Not valid statement

### **Program Writing/Program Writing 2:**

→ Not valid answers statement

All correct answers → Correct answers statement

Two correct answers and one wrong answer → One correction statement

One correct answer and two wrong answers → Two corrections statement

1, 1, 1 → Incorrect answers statement

Test, 1, 1 → Not valid answers statement

1, Test, 1 → Not valid answers statement

1, 1, Test → Not valid answers statement

1, Test, Test → Not valid answers statement

Test, 1, Test → Not valid answers statement

Test, Test, 1 → Not valid answers statement

Test, Test, Test → Not valid answers statement