STEAM - EDUCATIONAL ROBOTICS: MOVING FROM SCRATCH TO PYTHON PROGRAMMING FOR ADVANCED STUDENTS.

by

EVANGELIA ANASTASAKI

Previous degree(s). B.A., University of Aegean, 2015

A THESIS

submitted in partial fulfilment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING

HELLENIC MEDITERRANEAN UNIVERSITY

2022

Approved by:

Vassilakis K.
Kalogiannakis M.
Papadakis S.

Abstract

The thesis studies the introduction of educational robotics within STEAM frameworks and investigates the case of smoothly shifting from Scratch to Python for STEAM robotics learning in advanced classrooms. Initially, we present an analysis of Scratch versus Python commands to integrate educational robotics into the school efficiently. Comparing and contrasting differences in the language rules of Scratch and Python was valuable because students can create more dynamic programs by understanding the coding process. It can become with physical computing hardware, such as LEGO WeDo 2.0, which can assist the students in noticing the abstract concepts of Scratch and Python programming in practice.

Furthermore, we constructed experimental functions/commands in Python programming language through a Raspberry Pi Platform, permitting a suitable connection to the WeDo 2.0 robot based on Scratch for advanced students. The main reasons for developing the commands are that Scratch language is a novice programming, and students gain incorrect perceptions of programming behaviour (Moors, Luxton-Reilly, & Denny, 2018). In contrast, Python is real-world programming, which students can utilise the language in future careers, and students can also create dynamic programs in Python using WeDo 2.0.

Additionally, we present some projects using the developed WeDo 2.0 Python functions compared with the WeDo Scratch programs as examples for activities in the classroom.

The limitation of the thesis was the lack of testing functions in actual instructive practice for data collection about the effectiveness of Python WeDo 2.0 commands in the classroom. The contribution of this thesis lies in the novelty framework of the development of WeDo 2.0 Python functions, which can be utilised in STEAM robotics advanced classrooms for learning in the fields of science, technology, engineering, the arts and mathematics.

Keywords: STEAM, Educational Robotics, Scratch, Python, WeDo 2.0, Raspberry Pi

Περίληψη

Η διατριβή μελετά την εισαγωγή της εκπαιδευτικής ρομποτικής στα πλαίσια STEAM και διερευνά την περίπτωση της ομαλής μετάβασης από το Scratch στην Python για εκμάθηση ρομποτικής STEAM σε προχωρημένες τάξεις. Αρχικά, παρουσιάζουμε μια ανάλυση των εντολών Scratch έναντι της Python για την αποτελεσματική ενσωμάτωση της εκπαιδευτικής ρομποτικής στο σχολείο. Η σύγκριση και η αντιπαράθεση διαφορών στους γλωσσικούς κανόνες του Scratch και της Python ήταν πολύτιμη επειδή οι μαθητές μπορούν να δημιουργήσουν πιο δυναμικά προγράμματα κατανοώντας τη διαδικασία κωδικοποίησης. Μπορεί να γίνει με physical computing, όπως το LEGO WeDo 2.0, το οποίο μπορεί να βοηθήσει τους μαθητές να παρατηρήσουν στην πράξη τις αφηρημένες έννοιες του προγραμματισμού Scratch και Python.

Επιπλέον, κατασκευάσαμε πειραματικές συναρτήσεις/εντολές σε γλώσσα προγραμματισμού Python μέσω μιας πλατφόρμας Raspberry Pi, επιτρέποντας την κατάλληλη σύνδεση με το ρομπότ WeDo 2.0 που βασίζεται στο Scratch για προχωρημένους μαθητές. Οι κύριοι λόγοι για την ανάπτυξη των εντολών είναι ότι η γλώσσα Scratch είναι ένας αρχάριος προγραμματισμός και οι μαθητές αποκτούν εσφαλμένες αντιλήψεις για τη συμπεριφορά προγραμματισμού (Moors, Luxton-Reilly, & Denny, 2018). Αντίθετα, η Python είναι προγραμματισμός πραγματικού κόσμου, τον οποίο οι μαθητές μπορούν να χρησιμοποιήσουν τη γλώσσα σε μελλοντικές σταδιοδρομίες και μπορούν επίσης να δημιουργήσουν πιο δυναμικά προγράμματα στην Python χρησιμοποιώντας το WeDo 2.0.

Επιπλέον, παρουσιάζουμε ορισμένα έργα που χρησιμοποιούν τις ανεπτυγμένες συναρτήσεις WeDo 2.0 Python σε σύγκριση με τα προγράμματα WeDo Scratch ως παραδείγματα για δραστηριότητες στην τάξη.

Ο περιορισμός της διατριβής ήταν η έλλειψη δοκιμής των συναρτήσεων στην πραγματικό περιβάλλον διδασκαλίας για τη συλλογή δεδομένων σχετικά με την αποτελεσματικότητα των εντολών Python WeDo 2.0 στην τάξη. Η συμβολή αυτής της διπλωματικής εργασίας έγκειται στο καινοτόμο πλαίσιο ανάπτυξης των συναρτήσεων WeDo 2.0 Python, οι οποίες μπορούν να χρησιμοποιηθούν σε προχωρημένες τάξεις ρομποτικής STEAM για μάθηση στους τομείς της επιστήμης, της τεχνολογίας, της μηχανικής, των τεχνών και των μαθηματικών.

Λέξεις Κλειδιά: STEAM, Εκπαιδευτική Ρομποτική, Scratch, Python, WeDo 2.0, Raspberry Pi

Acknowledgements

I want to thank Professors Vassilakis Kostas, Kalogiannakis Michail and Papadakis Stamatios for their guidance and support throughout the writing of my master thesis.

Also, thank you to them for offering me the opportunity to publish a part of my work in a foreign language journal article.

Preface

From my master thesis was completed a published manuscript peer-reviewed journal article. The journal is called “*Advances in Mobile Learning Educational Research*” in which Editor-in-Chief in Greece is Professor Stamatios Papadakis.

The citation information of the article is as follows:

Anastasaki, E., & Vassilakis, K. (2022). Experimental commands development for LEGO WeDo 2.0 in Python language for STEAM robotics advanced classes. *Advances in Mobile Learning Educational Research*, 2(2), 443-454.
https://doi.org/10.25082/AMLER.2022.02.013

Table of Contents

List of Figures

List of Tables

Chapter 1 - Introduction

Computer science is an integral part of modern-day life. The computer is now used for information retrieval, communication, data processing, and managing many situations in professional and everyday life. Computer programming is an integral part of computer science, and a pleasant object of study and study since the hardware and software are adapted to the user's requirements (Blanchard, Gardner-Mccune, & Anthony, 2019). The introduction of programming in the informatics curricula of all levels of the educational system shows us the great value of teaching its principles and using them to solve problems (Mayerové & Veselovská, 2017). Researchers and educators agree with Papert, who points out that “programming can be an educational tool for cultivating and developing mental skills in all students and motivates a structured way of thinking and dealing with problems in almost all cognitive objects” (Papert, 1980).

Due to the cognitive object of informatics, programming is the object of study in the research community. More specifically, many research papers and publications are concerned with teaching the concepts of variable, recursion, selection, and repetition structures. Research has shown that programming often makes it difficult for students (Vihavainen, Paksula, & Luukkainen, 2011).

Conventional teaching methods do not solve students' problems, resulting in their negative attitude towards programming. STEAM education solves the problems created by long-established teaching approaches as it is an effective tool for cultivating and developing cognitive structures by students. STEAM is an approach to learning that employs Science, Technology, Engineering, the Arts, and Mathematics to direct pupil inquiry, dialogue, and critical thinking (Korkmaz, 2016).

“Scratch MIT” is a visual programming language that allows children to effectively create interactive content (Korkmaz, 2016; Papadakis, Kalogiannakis, Zaranis, & Orfanakis, 2016). “Python” is a “real” programming language with valuable data structures as a core part of the language; it gives them prominent names and makes them extraordinarily easy to use (Vega, 2018). Such usefulness, simplicity, and clarity remove barriers to programming, making the transition from Scratch language smoother.

The development set WeDo 2.0 permits pupils to construct and program basic LEGO models connected to a computer and can execute essential robotic assignments. This practice contains two stages, software, and hardware, which encourage the introduction of educational robotics into the classroom (Chalmers, 2018; Olabe, Olabe, Basogain, Maiz, & Castaño, 2011).

The programming can be too difficult and tedious when learned through the conventional "abstract" method. On the contrary, pupils learn what robots can and cannot do with direct experience and understanding by handling a physical robot and noticing its situation. By programming robots, pupils can discover if their aptitudes and interests correspond to those skills that will define the future job market, like programming, science, technology, or engineering (Kalovrektis et al., 2021).

1.1 Scope & Objectives

The thesis will study the introduction of educational robotics within STEAM frameworks and investigate the case of smoothly moving from Scratch to Python for STEAM robotics learning in advanced classrooms. Notably, an analysis of Scratch versus Python syntax languages will be presented. Also, functions/commands will be developed in Python programming language through a Raspberry Pi Platform, permitting a suitable connection to the LEGO WeDo 2.0 robot kit based on Scratch for advanced pupils to shift from Scratch to Python programming.

The research questions are as follows:

1) How do we move from Scratch programming to Python programming in advanced students' classes concerning learning outcomes, opinions, and interest in the field of educational robotics?

2) How does Lego WeDo 2.0 operate with Python compared to Scratch for robotics learning?

1.2 Thesis Structure

This thesis is structured in six chapters:

- The first chapter comprises an introduction as also the scope and objectives.
- The second chapter refers to the theoretical framework that drives this thesis. This chapter studies the fundamental learning theories and skills of STEAM Education and Educational Robotics.
- The third chapter mentions the two programming languages (Scratch and Python) used in this thesis.
- The fourth chapter concerns the modification techniques for smoothly integrating educational robotics in the classroom. This chapter studies the similarities and differences in syntax (language rules) of Scratch and Python Languages.
- The fifth chapter presents the STEAM implementation using the LEGO WeDo 2.0 robot kit with Scratch and Python. This chapter describes this robot kit's electronics tools and Scratch commands. Also, this chapter presents the constructed experimental Python commands through a Raspberry Pi to interact with WeDo 2.0 for advanced robotic learning in the classroom compared with the Scratch commands of this robot kit.
- At last, the sixth chapter comprises the conclusions and opinions for future work on this subject.

Chapter 2 - Theoretical Framework

2.1 STEAM Education

STEAM is an integrated learning approach that requires a link between standards, assessments, and course design/implementation. STEAM experiences include standards from Science, Technology, Engineering, Mathematics, and the Arts that must be taught and evaluated together. STEAM is an evolution of the original acronym STEM (figure 1), plus one extra element: "Art". The art could offer the students "socioemotional" and "holistic development" (Ampartzaki, Kalogiannakis, Papadakis, & Giannakou, 2022). Integrating the skills into STEM learning has enabled teachers to extend the benefits of hands-on training and collaboration in various areas, fostering creativity and curiosity at the core. Research, study, collaboration, and process-based learning are at the heart of the STEAM approach. Utilising the arts' integrity is essential to an authentic STEAM initiative (Sutradhar & Naraginti, 2022).

There are five steps to creating a STEAM-focused learning environment:

- Focus: This step selects a substantive question to solve an answer or problem. It is essential to focus on how this question or issue relates to STEM and the designated arts content areas (Herro, Quigley, & Cian, 2018).
- Detail: The information contributing to the problem or question is searched during the detail phase (Judy, 2011). The correlations with other areas come with discovering much basic knowledge, skills, and procedures students follow to answer the question (Herro et al., 2018).
- Discovery: The discovery concerns exclusively active research and deliberate teaching. In this step, students explore current solutions and what does not work based on existing solutions (Herro et al., 2018).
- Application: This is where the fun takes place. Once students are deeply immersed in a problem or question and have analysed current solutions, they apply the skills, procedures, and knowledge learned at the discovery stage (Herro et al., 2018).

- Presentation: Once students have constructed their solution or composition, it is time to share it. The work must be presented for feedback and expression based on a student's perspective on the question or problem they face (Herro et al., 2018).

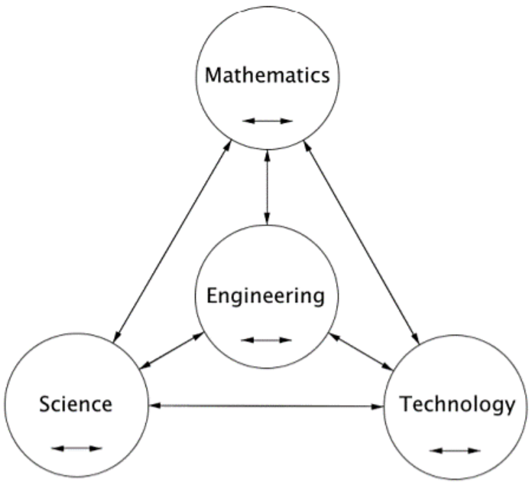


Figure 1: STEM Translation Model

Each of the five steps of the STEAM educational framework is important. As a result, students learn daily as they investigate, play in an active learning environment, and join innovative experiences (Ampartzaki et al., 2022; Zaher & Hussain, 2020). STEAM education is founded as an educational method in which students illustrate advanced concepts and reflect creatively while tackling association issues between these steps (Przybylla & Romeike, 2014). STEAM education improves educational outcomes and accomplishments and empowers learning opportunities, the pupil's logical and analytical skills, creative abilities, learning experiences, and student evolution to solve real-world issues (Dell'Erba, 2019).

STEAM's educational approach to instructing speaks to advancement towards an inventive demonstration that upgrades the learning preparation and progresses learning results utilising the central process (An, 2020). Students learning experiences include two or more STEAM standards, and learning innovation ordinarily abuses the art frame (Erba, 2019). In addition, the advanced STEAM method of education demands the proper instruments and resources. Before instruction, the teacher should preview the full range of available STEAM products (Erba, 2019).

2.2 Learning Theories & Skills

STEAM with Lego WeDo 2.0 is distinguished by constructionism, active learning, computational thinking, creative thinking, and problem-solving.

2.2.1 Constructionism

Constructionism advances social and communication skills by creating a classroom environment that emphasises collaboration and the exchange of thoughts. Pupils must learn to verbalise their opinions clearly and collaborate on tasks successfully by sharing in group projects (Evripidou et al., 2020). Papert's constructionism focuses on learning in the circumstances “rather than looking at them from a distance, that connectedness rather than separation are powerful means of gaining understanding” (Ackermann, 2001). This implies that new experiences are the entirety of a single experience made by applying existing knowledge to enhance it. In this manner, “hands-on exercises are the best for the classroom applications of constructionism, critical considering and learning” (Ackermann, 2001). Each person has a different perception and construction of the knowledge process (Khanlari, 2014).
The theory of constructivism argues that students do not passively accept knowledge but are more active in the learning process. They build on experience to understand what they are learning. They are more involved in creating meaning and understanding. This leads to a student-centred approach in which the student guides their learning through various technological tools (Benotti, Gómez, & Martínez, 2017).

2.2.2 Active Learning

Active learning is an instructional approach that effectively engages students with the course material through discussions, problem-solving, case studies, role plays, and other strategies (Ampartzaki et al., 2022; Sisman & Kucuk, 2019). Active learning approaches put a more prominent degree of obligation on the learner than passive approaches such as lectures, but educators' direction is still significant within the active learning classroom (Zaher & Hussain, 2020). Active learning exercises may range in length from some minutes to entire course sessions or take place over numerous course sessions (Zaher & Hussain, 2020).

2.2.3 Computational thinking

Computational thinking is a set of aptitudes to solve problems (Wing, 2006); it is a thought process (Psycharis, 2018), or it is a problem-solving process (Papadakis & Kalogiannakis, 2022). Computational thinking allows us to analyse a complex problem, understand it and develop possible solutions. Thus, we manage to present these solutions so that the computer and its operator can understand the problem to look for ways to solve it. The four basic steps in the process of applying computational thinking are (Psycharis, 2018):

- <u>Decomposition:</u> breaking down a complex and challenging problem into smaller ones that can be more manageable.
- <u>Identification of similarities</u>: the search for similarities between minor problems.
- <u>Abstract process:</u> focus only on important information, and ignore irrelevant details.
- <u>Algorithms:</u> develop a solution to the problem (step by step) or classify the rules to solve it.

Each stage of the above is critical to continue to the next. The correct application of all four steps will help solve the complex problem best. Computational thinking is not computer programming; it is devising solutions proper for computers. Programming guides the computer on what to do and how to do it. So, computational thinking enables us to work on the computer, just as we say what we want it to do (Catlin & Woollard, 2014).

Seymour Papert first used the term Computational Thinking in 1980. The key features of the Computational Thinker are (Psycharis, 2018):

- The formation of structures.
- The combination of mathematical thinking and techniques from the science of engineering.
- Computational thinking leads to “ideas” and not “artefacts”.

The pedagogy of science, technology, engineering, the arts, and mathematics (STEAM) can be quickly developed using robotic and computational thinking tools (Psycharis, Kalovrektis, & Xenakis, 2020). The STEAM approach followed in the educational process is essential for connecting computational thinking with problem-solving. This strategy allows students to turn complex problems into a more straightforward process, following the phases of a teaching scenario (Psycharis et al., 2020; Wing, 2008).

Teachers who use STEAM techniques in the teaching process, including computational thinking, allow students to practice problem-solving by trying and making mistakes (Ampartzaki et al., 2022; Kim & Han, 2018; Wing, 2008). A highlight of STEAM is that inquiry-based learning is adopted, abandoning teacher-centred teaching, with pupils' inclusion in dynamic problem-solving through a project-based plan, development, and project-based learning, taking into consideration conditions, variables, and all sorts of imperatives, such as social, environmental and technical (Pham, Misra, Huynh, & Ahuja, 2019).

2.2.4 Creative thinking

Creative thinking can be interpreted as thinking that can connect or see things from a new perspective. While the characteristics of creative people are curiosity, being resourceful, having the desire to find, choosing difficult jobs, enjoying solving problems, having the dedication to work, thinking flexibly, asking lots of questions, giving better answers, being able to synthesise, see new implications, and have a broad knowledge (Murcia, Pepper, Joubert, Cross, & Wilson, 2020). Generating new ideas and ways to produce a product is thinking creatively, namely leads to gaining new insights, approaches, perspectives, and practices when dealing with issues (McGregor, 2007). It is a skill in carrying out a mindset based on a deep understanding of the concepts that an individual has previously mastered. That mindset will then influence the individual's mind to make changes. The general problem for a teacher may be unusual for a student. Therefore, teachers must develop learning methods or strategies that can create the creative thinking power of their students (Noviani & Wangid, 2018). Also, creative thinking skills include fluency, flexibility, creativity, and elaboration abilities and are suitable for collaboration with 21st-century learning, namely STEAM learning (Noviani & Wangid, 2018).

2.2.5 Problem–Solving

The 21st century requires a few preparations. One of the preparations must be having problem-solving ability. Solving problems is fundamental in numerous sciences, math, and engineering classes. Assume an objective of a course is for students to develop the ability to solve new sorts of issues or utilise new problem-solving strategies (Huei, 2015). In that case, students require various opportunities to create the abilities essential to approach and reply to diverse problems. The skill of problem-solving can be reflected in how somebody resolves issues precisely and reasonably (Astuti et al., 2021).

2.3 STEAM & Educational Robotics

Seymour Papert, in 1969, developed the Logo programming language and “robot turtles”. Papert considers that people learn according to mental models and learning environments. Also, he envisioned the ability of computers to engage learners through turtle programming in active learning (Catlin & Woollard, 2014). Papert's “turtles” are constructions programmed by learners to explore the world around them. The turtle is an “educational robot” on the ground that moves based on commands given by the computer. Based on Logo, various environments have been developed, called “logo-like environments’, and they are also used in physical computing, such as Scratch, EV3/Lego programming, etc. (Catlin & Woollard, 2014; Kalovrektis et al., 2021).

Educational robotics within STEAM frameworks can attain an attractive strategy that changes boring concepts into a fun learning preparation. STEAM robotics instruction gives imaginative challenges and openings for school-level learners to create one-of-a-kind concepts and advanced learning skills (Afari & Khine, 2017). STEAM with educational robotic kits is an efficient approach because they encourage the ease with which pupils can make connections among STEAM disciplines (Plaza et al., 2020; Plaza, Sancristobal, Carro, Castro, & Blazquez, 2018). Robotic kits like LEGO's, connected with a block-based programming language like Scratch, can help the pupil's fundamental programming learning (Dorling & White, 2015; Ruzzenente, Koo, Nielsen, Grespan, & Fiorini, 2012). The LEGO robot kits are appropriate tools that include pupils with hands-on learning through building with LEGO bricks and programming in a block-based programming environment (Chalmers, 2018; Elkin, Sullivan, & Umashi Bers, 2014), such as Scratch or with text-based programming (Armoni, Meerbaum-Salant, & Ben-Ari, 2015; Weintrop & Wilensky, 2017), such as Python. Educational Robotics with Python Language is used for teaching at various levels of education (Blank, Meeden, & Kumar, 2003; Khamphroo, Kwankeo, Kaemarungsi, & Fukawa, 2017; Vega, 2018). Many research articles reported positive results of using robots with Python to aid students in understanding programming concepts and writing programs (Khamphroo et al., 2017; Weintrop & Wilensky, 2017). By understanding the qualities and limits of block-based environments, teachers can work towards actualising programming tools that are most advantageous for beginners and shifting to text-based languages (Weintrop & Wilensky, 2017). Moreover, to encourage and entice learners in STEAM education, instructors utilise hands-on projects, called physical computing, that move the computational concepts from the screen into the real world to permit learners to interact with them (Juškevičienė, Stupurienė, & Jevsikova, 2020).

2.4 Physical Computing & LEGO

"Physical computing covers the design and realisation of interactive objects and installations and permits students to develop tangible, concrete products of the real world, which get up the learners' imagination" (Przybylla & Romeike, 2014). Students who utilised LEGO robots when comprehending programming had better test results in standardised programming than students who learned to program using traditional methods. When constructing an artefact with physical computing, it is already an entire computing system (Huang, K.H., Yang, T.M., & Cheng, Huang, Yang, & Cheng, 2013). Physical computing applies creative arts and engineering design processes and combines parts (such as sensors, LEDs, etc.) and software elements (Juškevičienė et al., 2020). Also, it brings computational thinking and programming closer to the students' daily lives. They can construct systems that are currently environmentally familiar to them and find it stimulating to comprehend more (Przybylla & Romeike, 2014). However, a common problem when instructing text-based programming languages is that the priority is learning specific syntax or semantics rather than comprehending the process. When students understand the process of coding, they can create more dynamic programs (Vihavainen et al., 2011). Physical objects can provide exceptional feedback to students. When utilising physical objects with computing, it is more effortless for students to notice whether the code is operating. They get direct feedback from the object itself. Students can instantly detect this when the system fails to meet expectations, for instance, due to unsuitable sensors or delayed responsiveness (Przybylla & Romeike, 2014). The feedback from objects will encourage students to create potent, expanded, entire, and valid programming and willingly spend lots of time debugging programs (Lawhead et al., 2003). With physical computing hardware, they have a "real" object that assists in noticing the abstract concepts with traditional programming languages in practice

(Hodges, Sentance, Finney, & Ball, 2020). Moreover, physical computing motivates learners to utilise their imagination and creativity, focusing more on ideas than technical restrictions. Students have the opportunity to develop projects that are meaningful to them (Przybylla & Romeike, 2014).

Chapter 3 – Programming Languages Used

In the framework of the project, we used two programming languages: Scratch and Python.

3.1 Scratch Programming

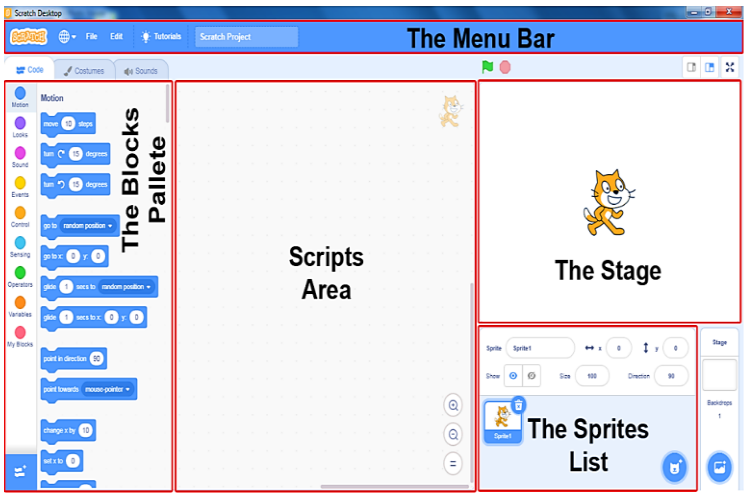


Figure 2: Scratch 3.0 development environment

"Scratch" is an interpreted, dynamic, high-level block-based visual programming language. Being passionate permits code changes even during program execution. It aims to teach programming concepts to children and teens and enable them to create games, videos, and music. It can be downloaded for free and utilised in various activities inside and outside the school worldwide (Plaza et al., 2018).

The Scratch interface (Figure 2) is separated into three main sections: a scene area, a block palette, and a coding region to place and arrange the blocks into scripts executed by pressing the green flag or clicking the code. Users can also construct their code blocks in "My Blocks" (Korkmaz, 2016).

The development environment includes the animation, graphics, and thumbnail sprites listed at the bottom. Utilises stage `x` and `y` coordinates, with `0.0` being the central stage. With a spirit selected at the bottom of the staging area, command blocks can be applied by dragging them from the block palette to the coding region. The Costumes tab permits users to modify the look of the sprite to create various effects, including animations. The sounds tab allows us to join sounds and music to a spirit. When making a sprite and background, users can design their sprite manually, select a Sprite from the library or upload an image. Users can also create their code blocks, appearing in 'My Blocks". (Lamb & Johnson, 2011).

3.2 Python Programming

"Python" is an "open-source", "interpreted", "general-purpose", and "high-level" programming language. It belongs to the wording of imperative programming and supports both procedural and object-oriented programming paradigms. Its dynamically typed programming language supports garbage collection (G.C.) (Wikipedia, n.d.).

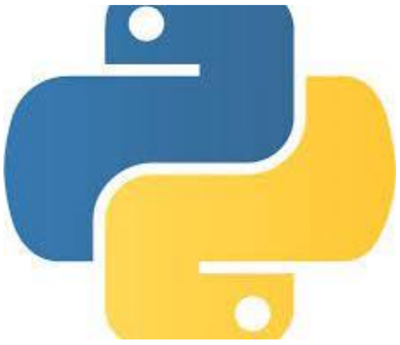
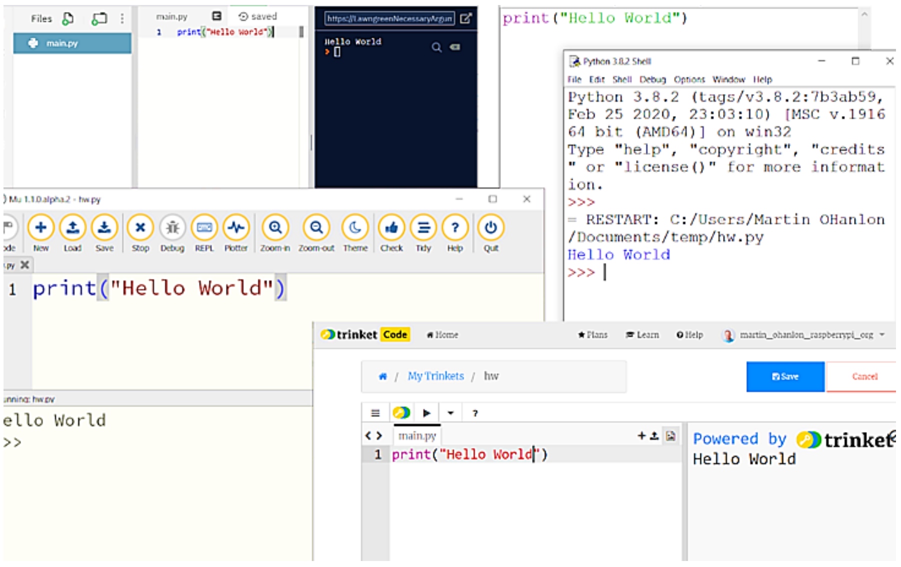


Figure 3: Logotype of Python

It was created by the Dutchman Guido van Rossum at the Centrum Wiskunde & Informatica (CWI) research centre in 1989 and was first published in 1991. Its main goal is its code's readability and ease of use. Its syntax allows developers to express concepts in fewer lines of code than possible in languages such as `C++` or `Java`. It is distinguished by its many libraries that facilitate many ordinary tasks and its learning speed. It has the disadvantage that because it is interpretable, it is slower than compiled languages such as `C` and `C++` (Bart, Tibau, Tilevich, Shaffer, & Kafura, 2017; Wikipedia, n.d.).

For this reason, it is not suitable for writing operating systems. Python interpreters are available for installation on multiple operating systems, allowing Python to execute code on various designs. Using third-party tools such as Py2exe or Pyinstaller, CPython can be packaged in standalone applications for some of the most prevalent operating systems, permitting the distribution of Python-based software for use in these environments without the need for installation Python interpreter. It is developed as open-source software and is managed by the non-profit organisation Python Software Foundation. The code is distributed under the GPL-compliant Python Software Foundation License. The name of the language comes from the comedian "Monty Python" and has nothing to do with the python snake, although its logo suggests something like that (Almeida, De Netto, & Rios, 2017; Wikipedia, n.d.).



```
print("Hello World")
```

```
Hello World
>
```

```
print("Hello World")
```

```
Python 3.8.2 (tags/v3.8.2:7b3ab59,
Feb 25 2020, 23:03:10) [MSC v.1916
64 bit (AMD64)] on win32
Type "help", "copyright", "credits
" or "license()" for more informat
ion.
>>>
= RESTART: C:/Users/Martin OHanlon
/Documents/temp/hw.py
Hello World
>>> |
```

```
print("Hello World")
```

```
ello World
>>
```

```
print("Hello World")
```

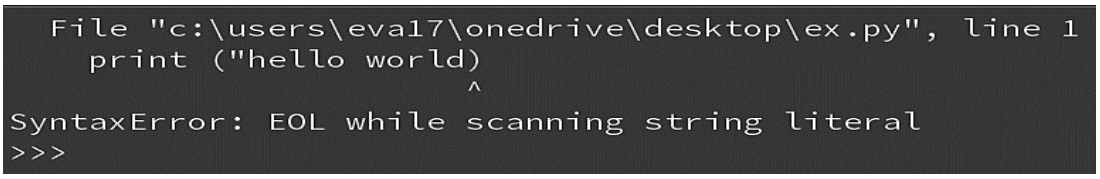
```
Hello World
```

Figure 4: Various Python IDLEs

Chapter 4 - Modification Techniques

To smoothly move from Scratch to Python for STEAM educational robotics in the classroom, it is valuable to compare and contrast differences in syntax (language rules) of both languages to assist students effectively.

Numerous students battle with text-based languages' viewpoints to understand the particular syntax required. Errors in a program are regularly due to non-compliance with these rules, and these mistakes are called syntax mistakes. In this manner, it is valuable to appear the parallels and contrasts between a language that a student has already mastered and one they are attempting to learn. When programming utilising a text-based language such as Python, you will soon encounter an error message like Figure 5 below:



```
  File "c:\users\eva17\onedrive\desktop\ex.py", line 1
    print ("hello world)
                       ^
SyntaxError: EOL while scanning string literal
>>>
```

Figure 5: Error Example (Missing the second quote) in "Mu" Python IDE

The example above is a syntax error. In general, every programming language has a different set of syntax rules. Hence, when the program does not follow these rules, a message appears, which tells it to be made an error. In the case of Figure 5 above, a second quote is not included in the command.

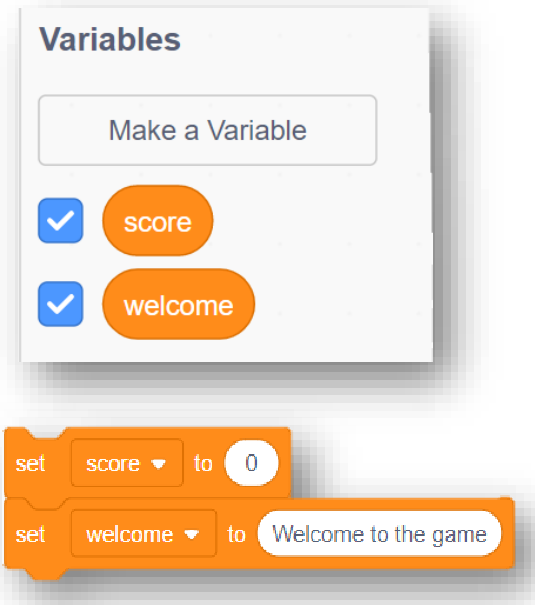
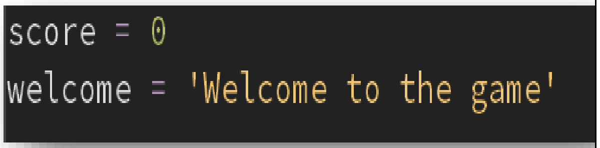
4.1 Basic Syntax

4.1.1 Variable Assignment

In Scratch, a variable should be created before assigning or setting its value to be given a variable. In Python, a variable is completed as soon as a value is assigned. Also, Python should be enclosed strings of text in a single (' ') or double quotes (" ").

Table 1: Variable Assignment in Scratch & Python

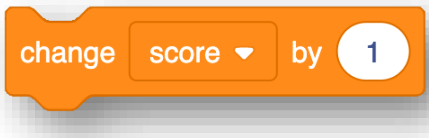
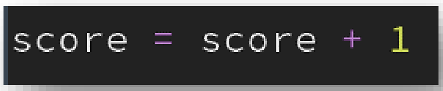
SCRATCH	PYTHON

 set score to 0 set welcome to Welcome to the game	 `score = 0` `welcome = 'Welcome to the game'`

4.1.2 Increment a Variable

In Scratch, to be incremented a value, we can use the change block to either increase or decrease a block's value. Whereas in Python, a new value must first be calculated by adding or subtracting an amount to the value. Then that new value is assigned back to the variable.

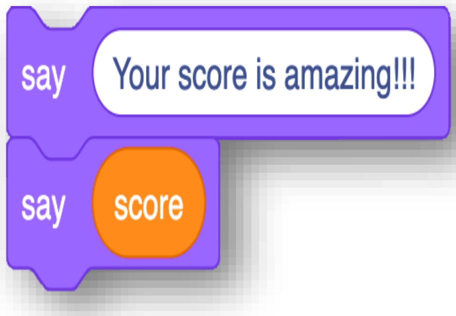
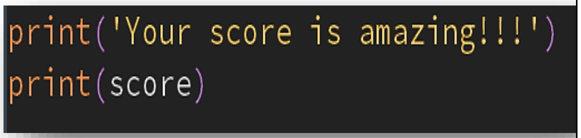
Table 2: Increment a Variable in Scratch & Python

SCRATCH	PYTHON
 change score by 1	 `score = score + 1`

4.1.3 Simple Output

Scratch to output to the screen should create a sprite talk by utilising the say block. An equivalent in Python would be the print statement, which will output text, numbers, or symbols to the screen. Nevertheless, we must also recall utilising single or double quotes when using text strings in Python.

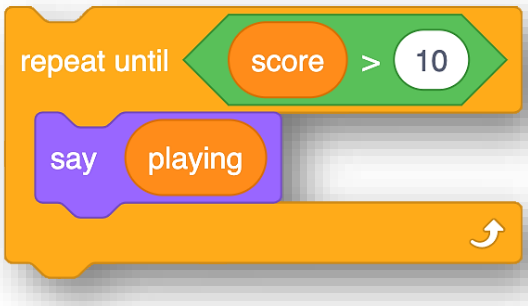
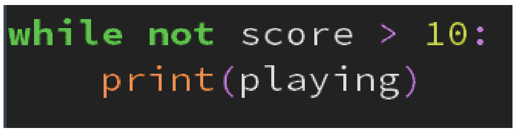
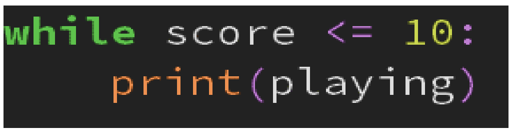
Table 3: Simple Output in Scratch & Python

SCRATCH	PYTHON
	 `print('Your score is amazing!!!')` `print(score)`

4.1.4 Conditional loops

One of the most significant differences is when utilising conditional loops. In Scratch, a conditional loop repeats until a statement is "`True`". In Python, a conditional loop repeats as long as a statement is true. Furthermore, we state how Python requires a colon (:) at the end of the statement and how a code part of the loop is indented. We can compare this to how the say block is indented as part of the repeat until the block in Scratch.

Table 4: Conditional Loops in Scratch & Python

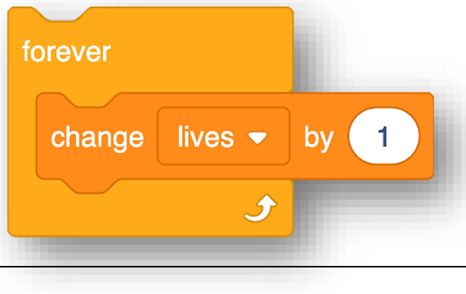
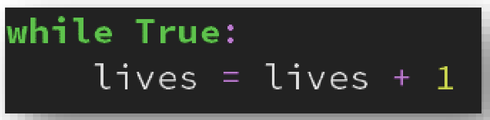
SCRATCH	PYTHON
	 `while not score > 10:` `    print(playing)` **or**  `while score <= 10:` `    print(playing)`

Consequently, explaining the codes above, in Scratch, we say, “repeat until the score is greater than ten and keep running this code until the variable score is greater than 10”. In Python, we say, “keep running this code while the score is not bigger than 10” or “keep running this code while the score is less than or equal to 10”.

4.1.5 Infinite loops

Scratch has a block type for making infinite loops, called a “forever block”. There is no equivalent, no clear forever block in Python, but a conditional loop is used, constantly evaluating “`True`”. So, while “`True`”, running this code is equivalent to forever.

Table 5: Infinite Loops in Scratch & Python

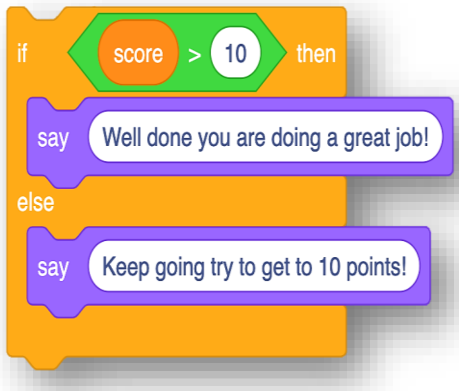
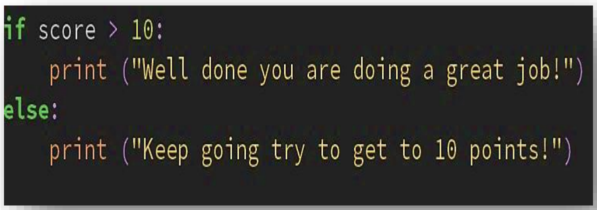
SCRATCH	PYTHON
	 `while True:` `    lives = lives + 1`

4.1.6 Conditions and If statements

When utilising selection in Scratch, there are two blocks: an “if” and an “else”. Python has a similar syntax: “If this condition executes this code, else execute this code”. Also, we utilise a colon (:) at the end of the statement and indentation to denote the code, which is part of this condition.

Table 6: Simple If statement in Scratch & Python

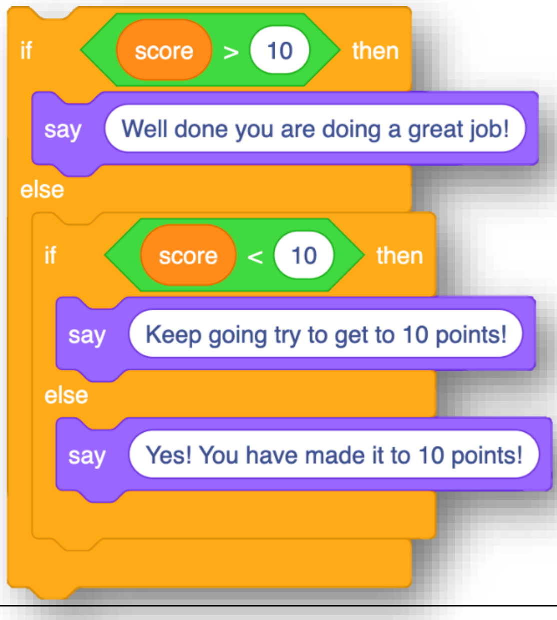
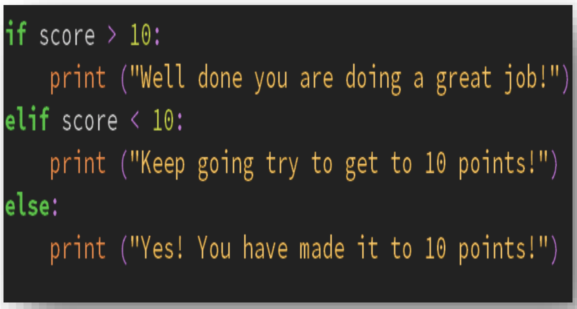
SCRATCH	PYTHON

 if score > 10 then say Well done you are doing a great job! else say Keep going try to get to 10 points!	 `if score > 10:` `print ("Well done you are doing a great job!")` `else:` `print ("Keep going try to get to 10 points!")`

```
if score > 10:
    print ("Well done you are doing a great job!")
else:
    print ("Keep going try to get to 10 points!")
```

However, if multiple conditions are required in Scratch, statements must be nested, one inside the other. Python has three conditional statements– `if`, `elif`, and `else`. Also, note how colon (:) is used and indentation again.

Table 7: Multiple if Statement in Scratch & Python

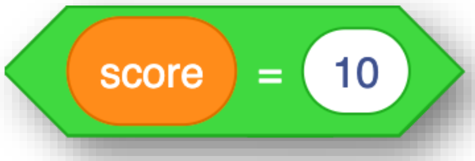
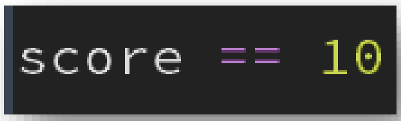
SCRATCH	PYTHON
 if score > 10 then say Well done you are doing a great job! else if score < 10 then say Keep going try to get to 10 points! else say Yes! You have made it to 10 points!	

```
if score > 10:
    print ("Well done you are doing a great job!")
elif score < 10:
    print ("Keep going try to get to 10 points!")
else:
    print ("Yes! You have made it to 10 points!")
```

4.1.7 Testing for equality

In Scratch test equality, we can utilise an operator block and a single equals sign. In Python, a single equals sign is reserved for variable assignment. So, a double-equals sign is used to test for equality.

Table 8: Test for Equality in Scratch & Python

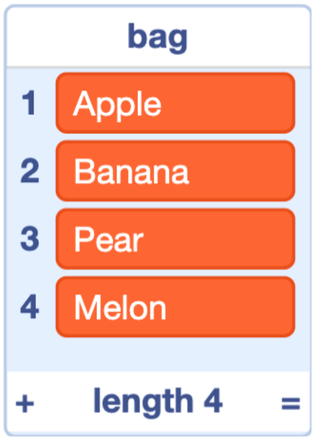
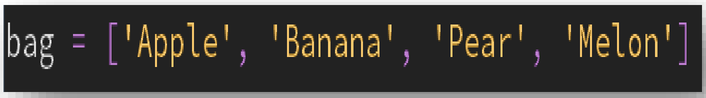
SCRATCH	PYTHON
	 `score == 10`

4.2 Advanced Syntax

4.2.1 Lists

When utilising a list in Scratch, they are very resemblance to variables. Scratch must be made a list and added items to the list. Whereas in Python, square brackets ([ ]) are utilised to create a list, and the items in the list are separated using commas.

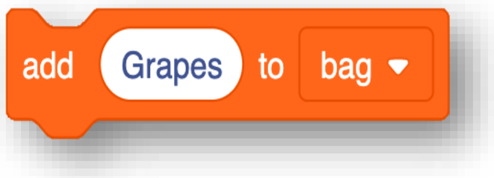
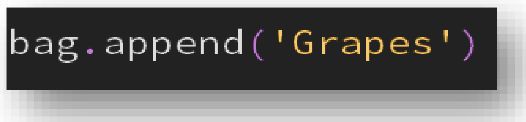
Table 9: Make a List in Scratch & Python

SCRATCH	PYTHON
 bag 1 Apple 2 Banana 3 Pear 4 Melon + length 4 =	 `bag = ['Apple', 'Banana', 'Pear', 'Melon']`

In Scratch, the “add block” adds an item to a list. Whereas in Python, it would be used the *“append”* method.

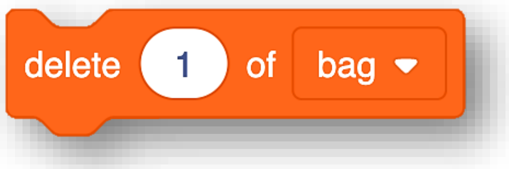
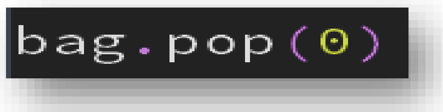
Table 10: Add an item to a list in Scratch & Python

SCRATCH	PYTHON

 add Grapes to bag	 `bag.append('Grapes')`

Removing items from lists is becoming in both languages using the delete or pop function. In Scratch, the first item in a list is at position `1`. In Python, it always starts counting from `0` (zero).

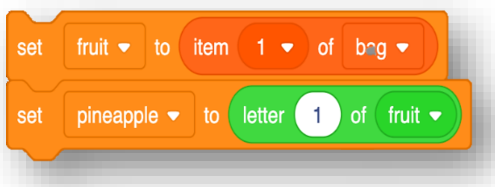
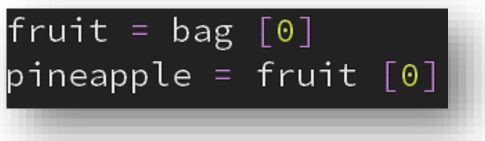
Table 11: Delete an item from a list in Scratch & Python

SCRATCH	PYTHON
 delete 1 of bag	 `bag.pop(0)`

4.2.2 Indexing

Searching for an item in a list or string is done in both languages using its index. Scratch uses the object block; in Python, it is used inside square brackets, the item number.

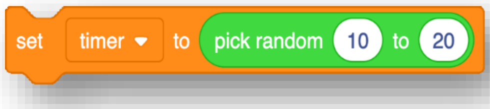
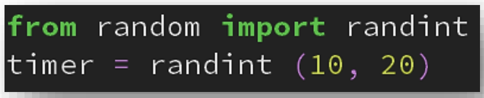
Table 12: Searching for an item in a list in Scratch & Python

SCRATCH	PYTHON
 set fruit to item 1 of bag set pineapple to letter 1 of fruit	 `fruit = bag [0]` `pineapple = fruit [0]`

4.2.3 Randomness

A "random block selection" could use random numbers in the program Scratch, which creates a random number in an area. In the case of Table 13 below, between 10 and 20. Python's similar function called "randint" (random integer) expresses a range. However, the "randint function" and "random module" must first be imported to use random processes in Python programs. The "`randint`" function stores the code that returns an integer number from a specified range. The "`random`" module is a part of the "standard library" with Python.

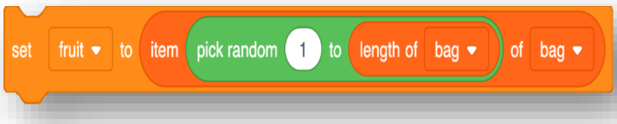
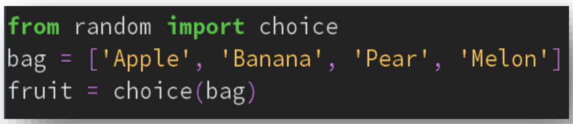
Table 13: Randomness in Scratch & Python

SCRATCH	PYTHON
	

```
from random import randint
timer = randint (10, 20)
```

In both languages, we can also select random items from a list. However, in Scratch, it has to be used the item "pick random" and the "length of blocks" to be able to find and "pick a random item" in a list. We can use the choice function to get a random item from a list in Python. The "choice function" is imported from the "random module".

Table 14: Selection of random items from a list in Scratch & Python

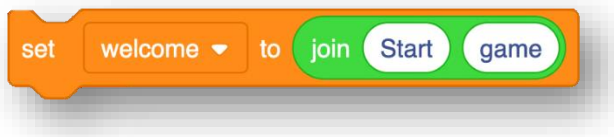
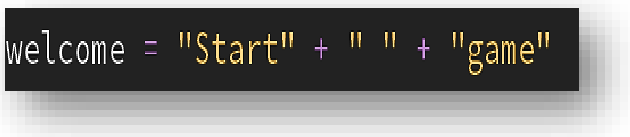
SCRATCH	PYTHON
	

```
from random import choice
bag = ['Apple', 'Banana', 'Pear', 'Melon']
fruit = choice(bag)
```

4.2.4 Concatenation

Scratch uses the “join operator block” to join items together, whereas Python uses the “addition operator”, a plus (+) sign, to join strings together. However, one of the critical differences is that space is automatically put between the two strings in Scratch, whereas it has to be entered in a further “space” in Python.

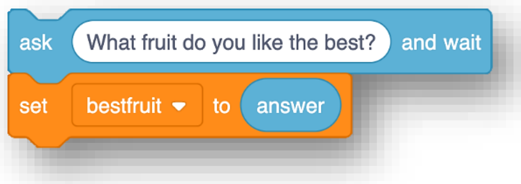
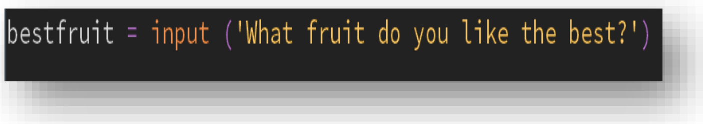
Table 15: Concatenation in Scratch & Python

SCRATCH	PYTHON
	 `welcome = "Start" + " " + "game"`

4.2.5 Input

Getting input from a user, in Scratch, we can use the “ask block”, which will prompt the user to ask a question, wait for a response to be answered, and then put that value into a particular variable called “answer”. A function in Python called “`input()`” will prompt the user to enter some text, and when enter is pressed, it will be assigned to a variable.

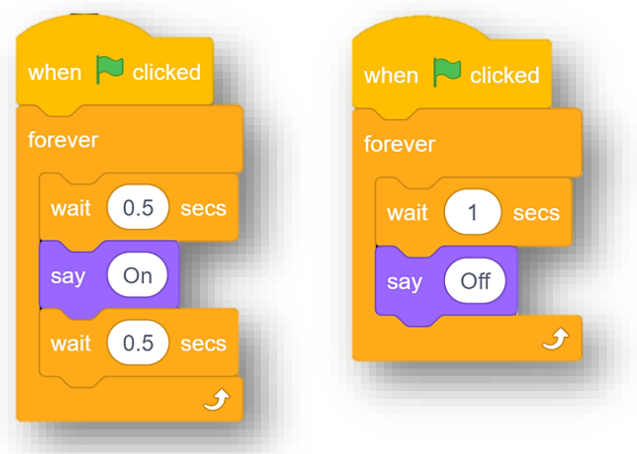
Table 16: Getting input from a user in Scratch & Python

SCRATCH	PYTHON
 ask What fruit do you like the best? and wait set bestfruit to answer	 `bestfruit = input ('What fruit do you like the best?')`

4.2.6 Concurrency

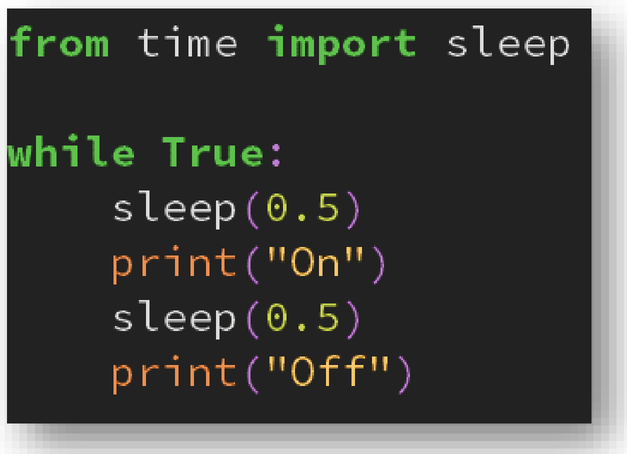
Scratch's most notable and valuable feature is its ability to execute multiple code blocks simultaneously. This means that the computer does not have to wait for one code block to finish running before executing the next block. This is known as concurrency. So, it uses the block "wait". This block is used to wait for [n] number of seconds, where [n] is any integer.

Table 17: Concurrency in Scratch

SCRATCH
 when clicked forever wait 0.5 secs say On wait 0.5 secs when clicked forever wait 1 secs say Off

Most text-based programming languages like Python do not allow the programmer to execute multiple parts of their program in parallel with such ease. The code must be condensed into a single loop to replicate the Scratch program and work in Python. The "sleep" function can be used to get the delay of the Python program. The "*sleep*" function is imported from the "*time*" module.

Table 18: Concurrency in Python

PYTHON


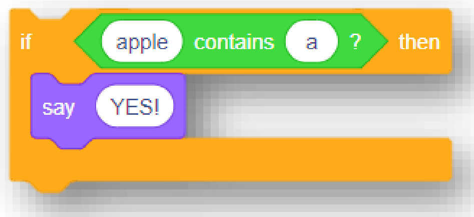
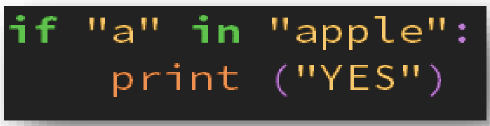
```
from time import sleep

while True:
    sleep(0.5)
    print("On")
    sleep(0.5)
    print("Off")
```

The concept of concurrency can sometimes be difficult for students to understand, so it is critical to tackling it reasonably early on when they transition to writing programs in a text-based language.

4.2.7 Membership Operator

Membership operators are used to testing if a sequence is presented in an object. In Scratch, it is used as "`contains`", whereas in Python ", `in`" operator. Both return "`True`" if a sequence with the specified value is present in the object.

Table 19: Membership Operator in Scratch & Python

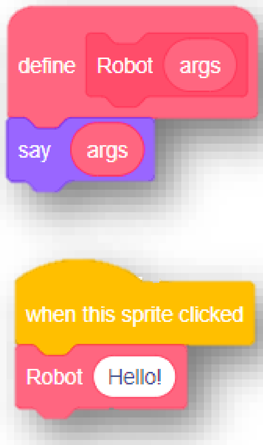
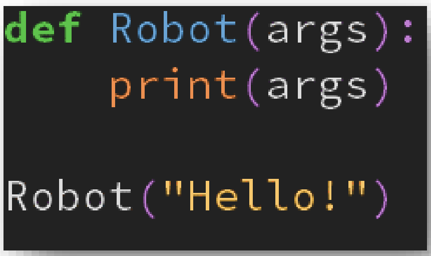
SCRATCH	PYTHON
	 `if "a" in "apple":` `    print ("YES")`

4.2.8 Procedures

A procedure is a separate code module that executes some tasks and is referenced within the main body of the source code. This code item can also be called a function, subroutine, routine, method, or subprogram. Procedures can be utilised repeatedly throughout a program and triggered by invoking the process.
In Scratch, we can use "My Blocks" to construct a procedure. "My Blocks" creates new blocks, in which we should give a name and "`define`" their operation. We can also add "inputs" to "My blocks" to define parameters. In Python, a function is determined utilising the "`def`". To invoke a function, we use the function name followed by brackets. Moreover, the parameters are determined after the function name is inside the brackets.

Table 20: Procedures in Scratch & Python

SCRATCH	PYTHON

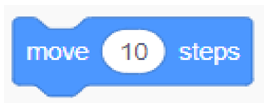
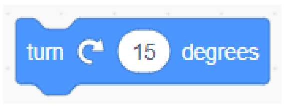
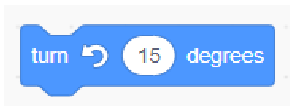
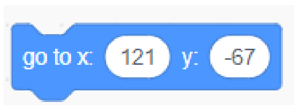
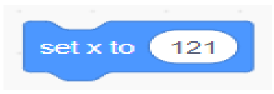
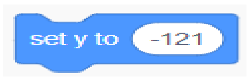
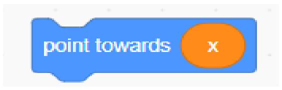
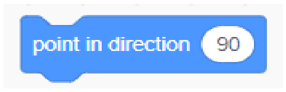
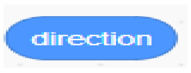
 define
Robot
args
say
args
when this sprite clicked
Robot
Hello!
	
	```
	def Robot(args):
	    print(args)

	Robot("Hello!")
	```

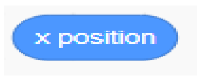
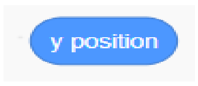
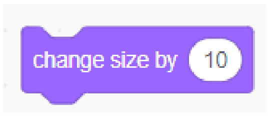
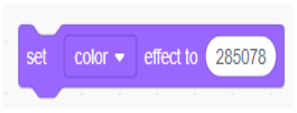
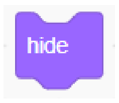
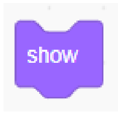
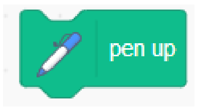
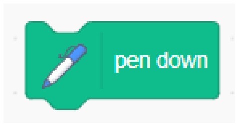
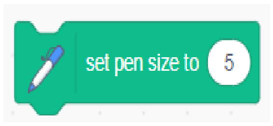
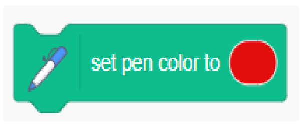
4.3 Turtle Graphics

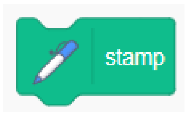
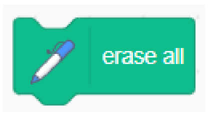
“Turtle graphics” is a famous way of teaching programming to children. It was part of the initial Logo programming language developed by Wally Feurzeig, Seymour Papert, and Cynthia Solomon in 1967 (Python, n.d.). “Logo” language permitted the control of a “turtle” device, a kind of robot associated with the computer (Rossum, Foundation, Foundation, Python, & Groovy, n.d.). The turtle may move back and forward on an even surface and alter its direction and orientation, taking off a mark shaping graphics. In Scratch, each “sprite” has the choice to act just like the “Logo turtle”, deciding their position, direction, and movement orientation, and they can take off a mark when they move. The magnitude, form, and colour of the “sprite” do not influence the “mark” they take off since the sprites are drawing with a “pen”. The “sprite” can also be invisible or composed of, as it were, one “dot”, not including its drawing. In Python, a module “`import turtle`” is imported, and the code is given suitable commands. So, the robotic turtle (→) moves on the screen, drawing a line, starting at (`0,0`) in the (`x,y`) plane. The turtle module is an “extended reimplementation” from Python. Below, itis mentioned and explained the standard commands of turtle graphics, transitioning them from Scratch to Python:

Table 21: Turtle Graphics Commands in Scratch & Python

SCRATCH	PYTHON	EXPLANATION

 move 10 steps	`turtle.forward(10)`	Move the object (turtle/sprite) forward by the specified distance in the direction it is headed.
 turn ↻ 15 degrees	`turtle.right(15)`	Turn the object's (turtle/sprite) direction to the specified degrees clockwise.
 turn ↺ 15 degrees	`turtle.left(15)`	Turn the object's (turtle/sprite) direction to the specified degrees counterclockwise.
 go to x: 121 y: -67	`turtle.goto(x=121, y=-67)`	Set the object's (turtle/sprite) X and Y coordinates to the specified direction.
 set x to 121	`turtle.setx(121)`	Change the X position to a specified value.
 set y to -121	`turtle.sety(-121)`	Change the Y position to a specified value.
 point towards x	`turtle.towards(x)`	Direct the object in a specified direction.
 point in direction 90	`turtle.setheading(90)`	Point the object in the specified direction, rotating it.
 direction	`turtle.heading()`	Hold the object's direction, measured in degrees.

	`turtle.xcor()`	Report the object's X position.
	`turtle.ycor()`	Report the object's Y position.
	`turtle.shapesize(10)`	Change the object's size by the specified amount.
	`turtle.color("#285078")`	Change the object's colour by a pair of colour specification hexadecimal RGB colours.
	`turtle.hideturtle()`	Disappear the object.
	`turtle.showturtle()`	Appear the object.
	`turtle.penup( )`	Cause the object to start drawing a trail.
	`turtle.pendown( )`	Cause the object to stop drawing a trail.
	`turtle.pensize(5)`	Set the pen's size by a specified amount.
	`turtle.pencolor("red")`	Sets the pen's colour to the specified colour.

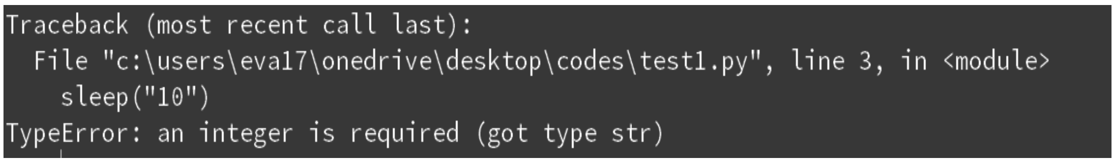
	`turtle.stamp()`	Produce a clone of the object stamped onto the stage, which cannot be programmed.
	`turtle.clear()`	Remove all marks made by the pen or stamping.

4.4 Debugging

“Debugging” is the procedure of detecting and removing existing and potential errors (also called “bugs”) in a code that can cause it to act unexpectedly or crash. Debugging is used to discover and fix bugs or defects to contain the flawed process of a software or system (The Economic Times, n.d.).

Debugging is significant since it permits software engineers and developers to fix errors in a program before launching it to the public. It is a complementary process to testing, which includes learning how an error influences a program by and large. Thinking and working efficient and logical is something nearly everyone can understand. It is an issue of preparation and discipline. So, debugging the proper way will assist the student in preparing logical and analytical skills (Lorenz, 2018), which, in my opinion, are the most remarkable abilities for an engineer.

In Scratch, the programs do not crash. They might not do anything or do what is expected, but they will never be shown error messages. When utilising a text-based language like Python, the programs crash pretty frequently. A single misplaced quotation mark, a missing colon, or a lower case letter where an upper case letter should be, and not only will the program refuse to run, but it will be found that the IDE suddenly fills with red text that might be barely comprehensible and therefore intimidating:



```
Traceback (most recent call last):
  File "c:\users\eva17\onedrive\desktop\codes\test1.py", line 3, in <module>
    sleep("10")
TypeError: an integer is required (got type str)
```

Figure 6: Type Error Example in “Mu” Python IDE

Thus, students must understand how to debug a text-based program when bugs are shown. With some coaching and detailed explanations, the students can develop the debugging skills required to resolve the issues in their programs' cases.

4.4.1 Types of Error

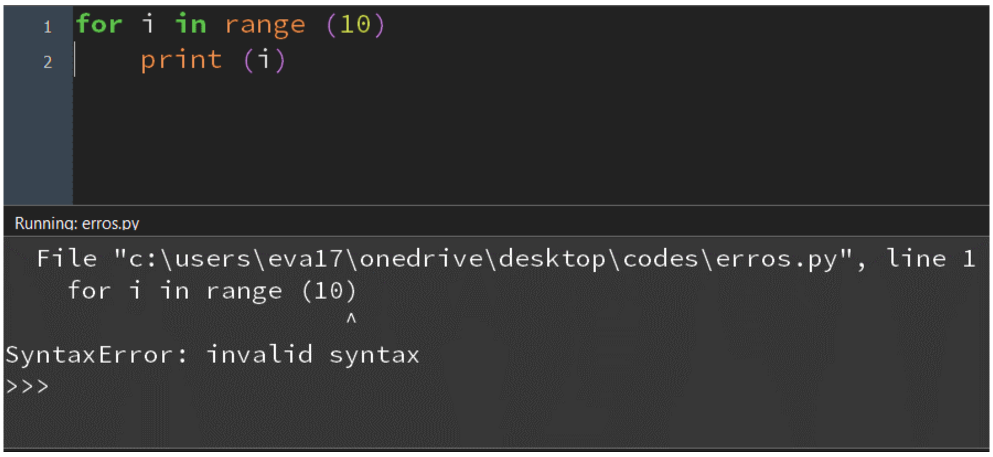
There are three types of errors that can appear in a program: "*syntax*", "*runtime*", and "*semantic*" errors. The first is relatively easy to detect in any standard programming interface. The second may be effortless to spot, but this depends on the exact error. The last might be hard to spot, but accomplishing so gets more effortless with practice.

4.4.1.1 Syntax Errors

Syntax Errors occur when the programmer does not follow the programming language's rules. Sometimes the programming environment will refuse even to try and run the program. Some standard cases are below:

❖ Missing colon

Figure 7 below shows a relatively simple syntax error in Python. The programming environment, in this case, "Mu" Python IDE, caught the error and declined to execute the program:



```
for i in range (10)
    print (i)
```

Running: erros.py

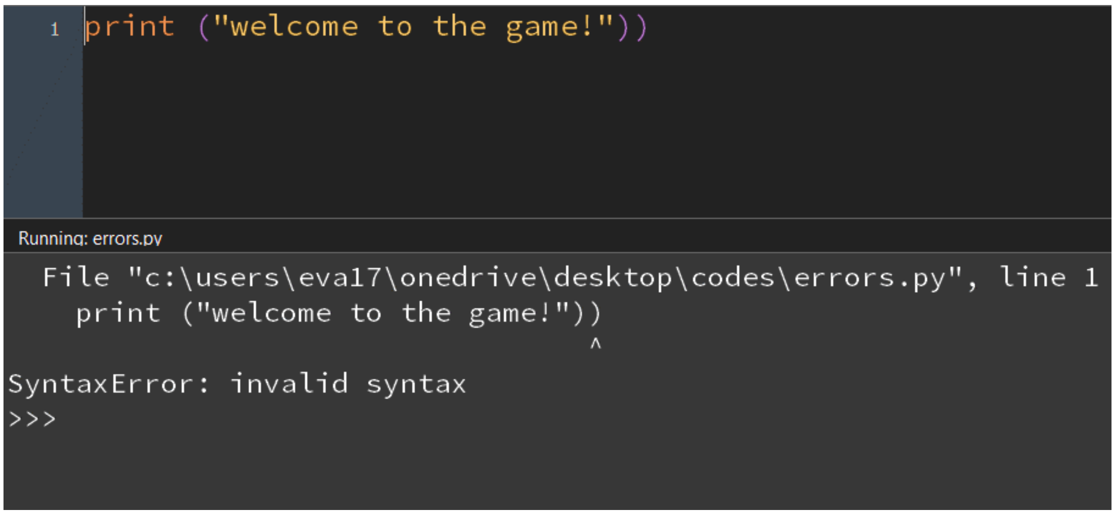
```
  File "c:\users\eva17\onedrive\desktop\codes\erros.py", line 1
    for i in range (10)
                      ^
SyntaxError: invalid syntax
>>>
```

Figure 7: Syntax Error (Missing colon) - Example in "Mu" Python IDE

The line where the problem has occurred is given in the error message (figure 7), and a mark (^) is used to designate the position of the error. A colon is missing after the bracket on line 1 in this case.

❖ Additional bracket

Another standard error that is effortless caught:



```
print ("welcome to the game!"))
```

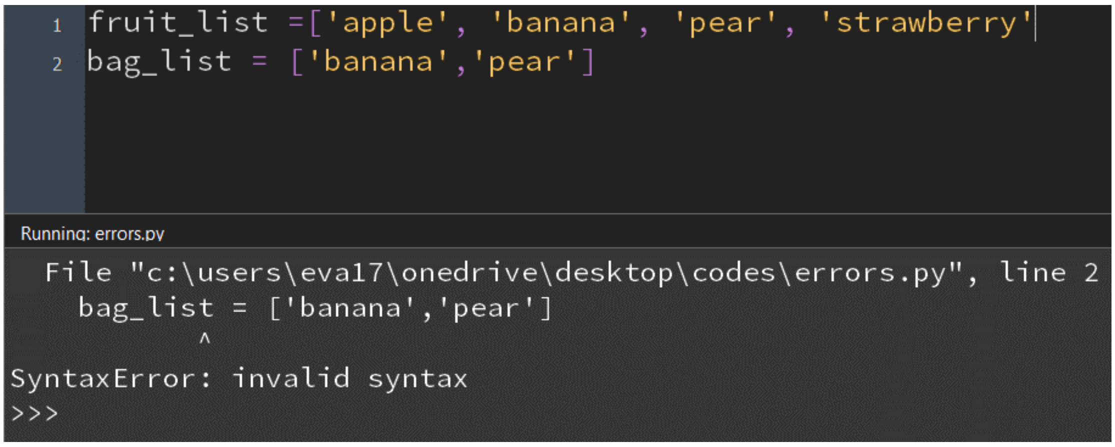
```
Running: errors.py
  File "c:\users\eva17\onedrive\desktop\codes\errors.py", line 1
    print ("welcome to the game!"))
                                  ^
SyntaxError: invalid syntax
>>>
```

Figure 8: Syntax Error (Additional Brackets) - Example in "Mu" Python IDE

This error message (figure 8) occurs when the programmer enters two brackets where there should only be one.

❖ <u>Missing square bracket</u>

This error can be a bit difficult:



```
fruit_list =['apple', 'banana', 'pear', 'strawberry'
bag_list = ['banana','pear']
```

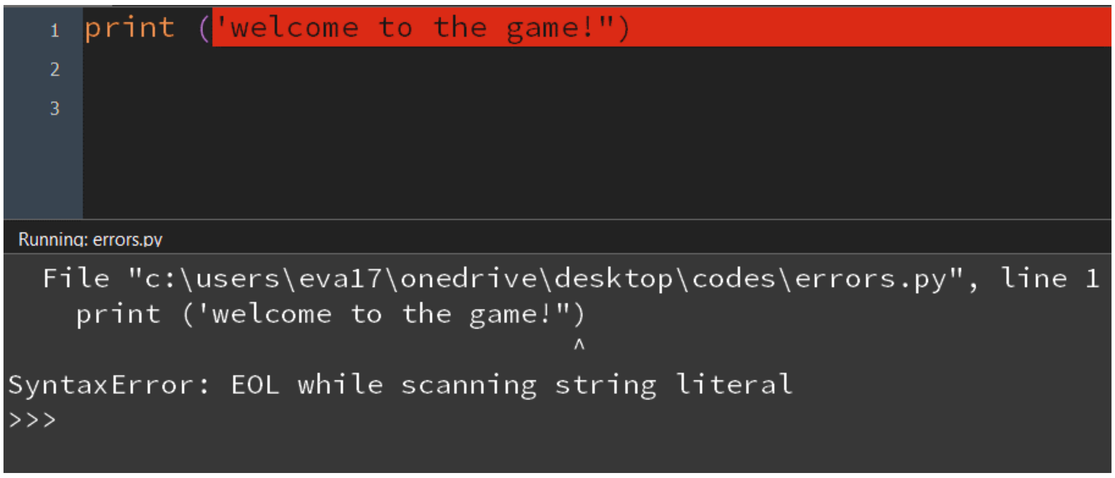
```
Running: errors.py
  File "c:\users\eva17\onedrive\desktop\codes\errors.py", line 2
    bag_list = ['banana','pear']
           ^
SyntaxError: invalid syntax
>>>
```

Figure 9: Syntax Error (Missing square bracket) – Example in "Mu" Python IDE

IDE seems to suggest that the error is on line 2. However, the problem is the failure to close the square brackets on line 1.

❖ <u>Incorrect quotation mark</u>

In the instance below (figure 10), the programmer has used a double quotation mark matched with a single quotation mark:



```
print ('welcome to the game!")
```

Running: errors.py

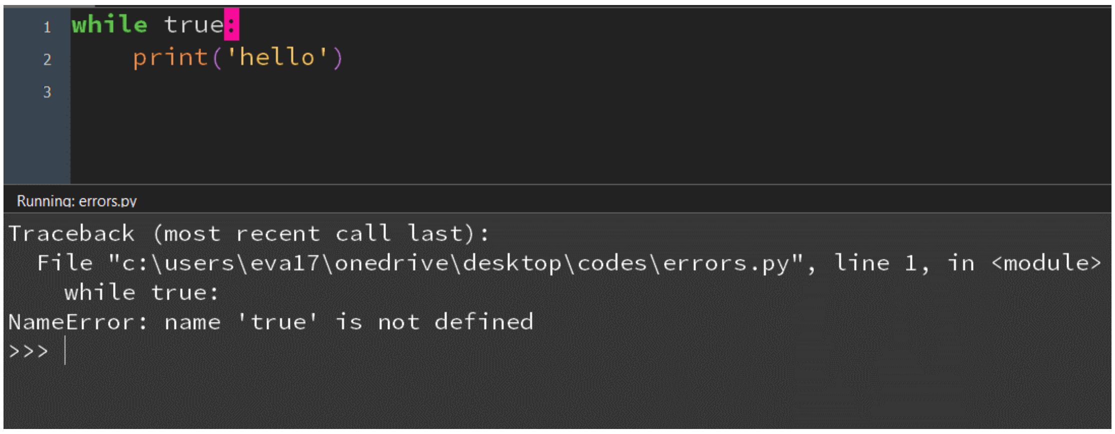
```
  File "c:\users\eva17\onedrive\desktop\codes\errors.py", line 1
    print ('welcome to the game!")
                                 ^
SyntaxError: EOL while scanning string literal
>>>
```

Figure 10: Syntax Error (Not same Quotation Mark) - Example in "Mu" Python IDE

Quotation marks should be the same around a single string. They are not interchangeable.

4.4.1.2 Runtime Errors

When Python provides a traceback error, things can get a bit difficult. These are runtime errors. They are not caught until the program executes, at which point a line of code that the IDE cannot perform will lead to a traceback error getting printed. The traceback provides information about what is incorrect in the code. Traceback errors may look wearisome, but they can be beneficial once you break them down. In the instance below, the error is that "`true`" has been used with a lowercase "`t`", whereas it should be "`True`".



```
while true:
    print('hello')
```

Running: errors.py

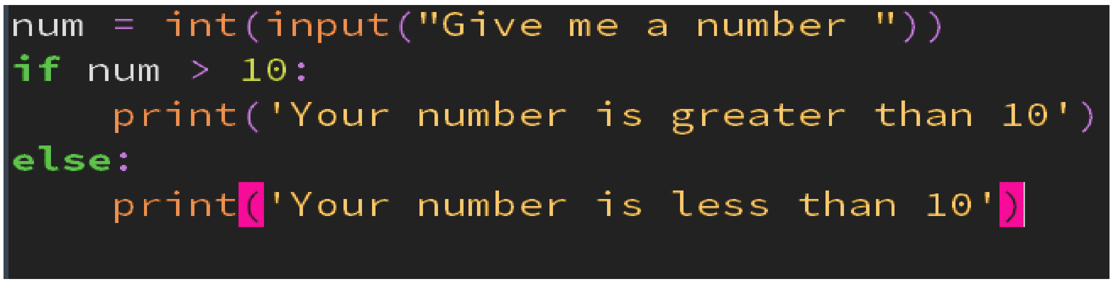
```
Traceback (most recent call last):
  File "c:\users\eva17\onedrive\desktop\codes\errors.py", line 1, in <module>
    while true:
NameError: name 'true' is not defined
>>>
```

Figure 11: Runtime Error (name bug) - Example in " Mu" Python IDE

The IDE did not recognise "`true`" as a keyword in this error and assumed it was a variable. Then when it tried to find the value of "`true`", it discovered that it had no value and kicked back a traceback error. The code can be fixed by replacing “`true`” with “`True`”.

4.4.1.3 Semantics Errors

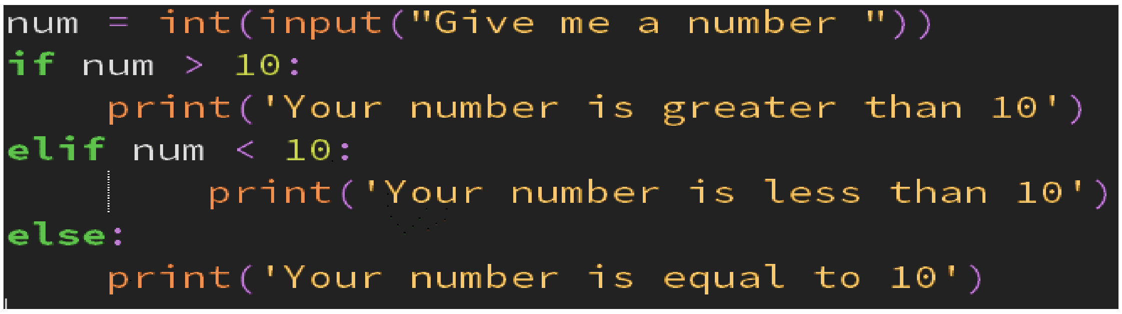
Semantic errors are the hardest to spot. The program will execute without problems, but it will not produce the expected result. Utilising a debugging tool, an option in most installed IDEs, to run the code step by step to find the exact point where the program made a mistake. Another helpful trick for debugging is the add lots of “`print()`” statements to the program to show the values of variables and the outputs of functions at specific points in the program. This is useful for checking that the values of certain variables are what you expect them to be. Also, another trick is the add “`print()`” statements to show when the program is reached a particular part, to work out what code is running. However, learning to debug the plan is going to take practice. Below (figure 12) is a program with a semantic error:



```
num = int(input("Give me a number "))
if num > 10:
    print('Your number is greater than 10')
else:
    print('Your number is less than 10')
```

Figure 12: Semantic Error Example in Python

The error is the program does not cover all possible cases. To fix this error should be added a point, which is executed in the case that “num is equal to 10” (figure 13):



```
num = int(input("Give me a number "))
if num > 10:
    print('Your number is greater than 10')
elif num < 10:
        print('Your number is less than 10')
else:
    print('Your number is equal to 10')
```

Figure 13: Fixed Semantic Error Example in Python

4.5 Basic Differences between Scratch & Python

In Scratch, each sprite provides its code. Sprites are connected through shared resources such as variables and messages. Typically, frequently confounding and leads to ineffectively viable code. Moreover, each block is perused in a consecutive arrangement. The code begins with an occasion or “`Hat`” block. Putting blocks inside external squares is utilised to group blocks together. Each sprite has a set of blocks that may or may not be associated.

In Python, each line of code is perused by a Python “Interpreter”, and the computer executes the information directly and in successive arrange. In addition, each line is examined from top to bottom, perusing each line from left to right. In case necessary, it is assigned its return value to a variable. Also, indentation groups statements together.

Chapter 5 – STEAM Implementation

Having understood the coding process, students can create more dynamic programs. It can become with physical computing hardware, such as LEGO WeDo 2.0. It will assist the students in noticing the abstract concepts of Scratch and Python programming in practice.

5.1 LEGO WeDo 2.0

LEGO Education launched WeDo 2.0 in January 2016. Mainly, it establishes the WeDo kits at elementary schools for children matured 7 to 11. But to my knowledge, the models are immensely advanced for this target group. Numerous children will discover it challenging to drive forward with a model and require a bit of offer assistance and support to get to the end of a project. Moreover, WeDo 2.0 is a "physical computing" robot kit for elementary students that develops programs on a computer or smart device based on a connected robot kit utilising Bluetooth Low Energy (BLE) technology (Ruzzenente et al., 2012; Chalmers, 2018; Olabe et al., 2011).

5.1.1 WeDo 2.0 Robot Kit

WeDo 2.0 Robot kit includes 280 LEGO blocks and electronics such as a "*Smarthub*", a "*Motor*", a "*Tilt Sensor*", and a "*Distance Sensor*":

5.1.1.1 Smarthub

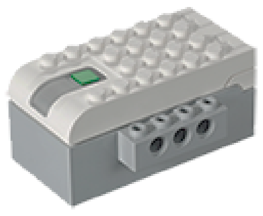


Figure 14: Lego WeDo 2.0 Smarthub

The WeDo 2.0 "Smarthub", or "brain of the robot", shown in Figure 14, is an electronic LEGO development block with two ports for input-output devices (including "motor", "distance sensor", and "tilt sensor"), a "green button" that can be pushed to combine the robot with another device and an "RGB LED". Moreover, the Smarthub can play sounds at foreordained frequencies. It utilises two "AA batteries" or a rechargeable battery as a power source. It has coordinates BLE, which permits it to put through and communicate with computers, tablets, and mobiles (LegoEducation, n.d.-b).

5.1.1.2 Motor

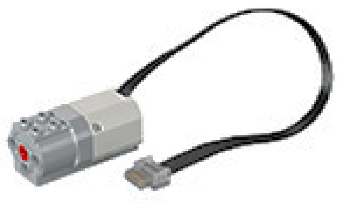


Figure 15: Lego WeDo 2.0 Motor

The WeDo 2.0 "Motor", shown in Figure 15, can be turned "clockwise" and "counterclockwise". It is a "simple-textured" motor that does not provide the robot's Smarthub feedback on how much it has turned, so a certain tilt cannot turn it, but the engine tries to turn at the power that the robot's Smarthub has set for it (LegoEducation, n.d.-b).

5.1.1.3 Tilt Sensor

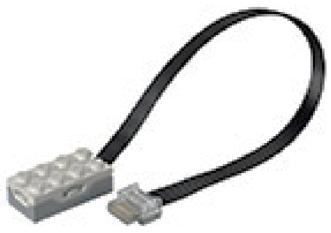


Figure 16: Lego WeDo 2.0 Tilt Sensor

The WeDo 2.0 "Tilt sensor", shown in Figure 16, sends the robot's Smarthub report about its recent tilt. It can detect seven positions (or orientations): tilt this way, tilt that way, tilt-up, tilt-down, no tilt, any tilt, and shake. The information is shipped to the Smarthub based on the angle of the two axes of the tilt sensor within 90 degrees. (LegoEducation, n.d.-b).

5.1.1.4 Distance (Motion) Sensor

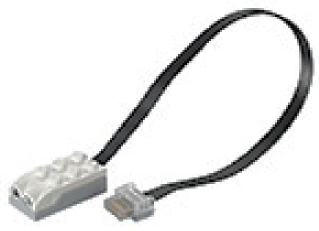


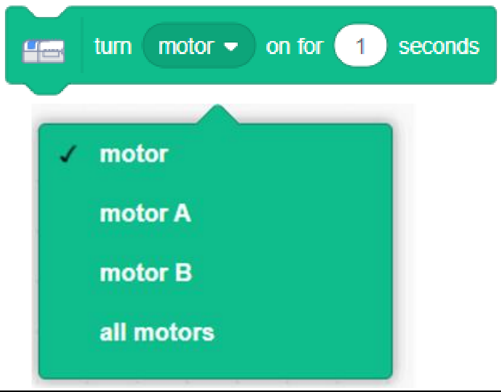
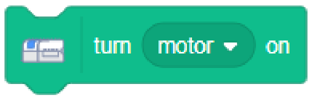
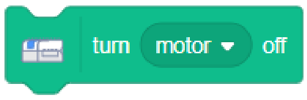
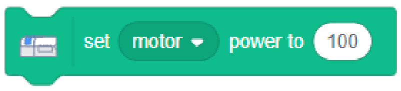
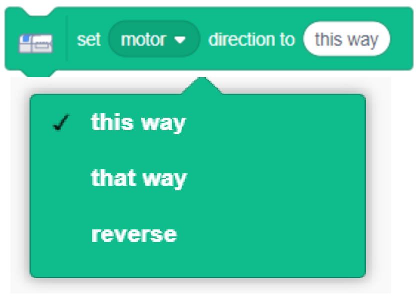
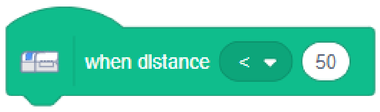
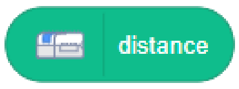
Figure 17: Lego WeDo 2.0 Distance (Motion) Sensor

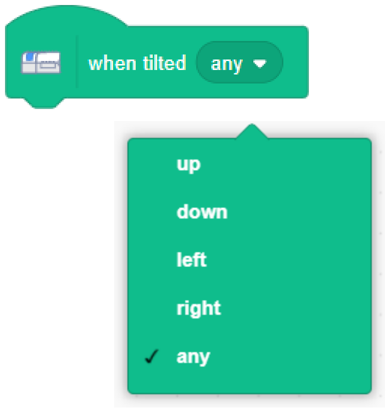
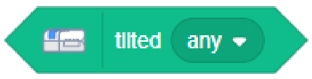
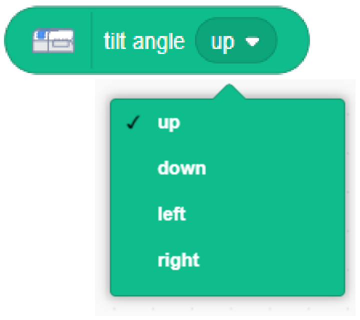
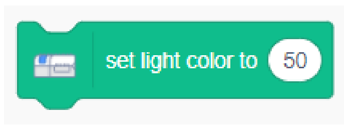
The WeDo 2.0 "Distance (Motion) sensor", which can be seen in Figure 17, provides the robot's "Smarthub" data about if an object is currently in front of it and, if so, how distant it is. The distance (motion) sensor can detect objects from a length of up to 15 cm. The robot's Smarthub can also measure and register how many motions it has seen in front of it (LegoEducation, n.d.-b).

5.1.3 WeDo 2.0 & Scratch

To develop the Python WeDo 2.0 commands, we will initially study Scratch Language's WeDo 2.0 block commands. The LEGO WeDo extension can be added within Scratch's "More Blocks" category. We should use the choice "Add an Extension" and choose "LEGO Education WeDo 2.0". We present and explain WeDo 2.0 blocks in Table 22 below:

Table 22: Commands Lego WeDo 2.0 in Scratch

 turn motor on for 1 seconds ✓ motor motor A motor B all motors	It powers up a specific motor for a particular amount of time. There are four options for the block, recorded as “motor”, “motor A”, “motor B”, and “all motors” (Scratch MIT, n.d.).
	It powers up a motor depending on the drop-down input.
	Depending on the drop-down input, it powers off a motor.
	It sets the power of a particular motor, controlling the engine’s rate at which it turns (Scratch MIT, n.d.).
 set motor direction to this way ✓ this way that way reverse	It sets the direction in which a particular motor should spin. There are three options for the “direction”, recorded as “this way”, “that way”, and “reverse”. The first two correspond to clockwise and counterclockwise. “Reverse” inverts the direction (Scratch MIT, n.d.).
	This “cap” executes a script when the distance gets to be less (or greater) than a particular value (Scratch MIT, n.d.).
	It mentions the distance sensor value (Scratch MIT, n.d.).

	This “cap” block executes a script when the kit is tilted in the specified direction. The drop-down options are “up”, “down”, “left”, “right”, and “any” (Scratch MIT, n.d.).
	It mentions whether or not the kit is tilted in the specified direction (Scratch MIT, n.d.).
	It mentions the angle at which the kit has been tilted in the specified direction. In contrast with other tilt blocks, “any” cannot be set in this block (Scratch MIT, n.d.).
	It sets the light colour on the kit LED to the specified value (Scratch MIT, n.d.).

5.1.4 WeDo 2.0 & Python

Interacting with Lego WeDo 2.0 with Python requires a system with Bluetooth Low Energy (BLE) technology (such as android, raspberry pi, etc.). We utilised Raspberry Pi and Python GATT Library to interact with WeDo's BLE system in this research. The connection to the Raspberry Pi Platform is realised through Headless Mode with Direct Ethernet Connection.

5.1.4.1 Raspberry Pi

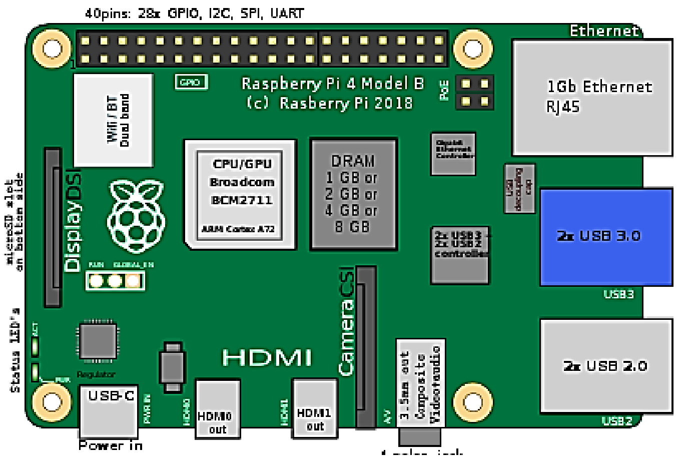

40pins: 28x GPIO, I2C, SPI, UART
Ethernet
1Gb Ethernet RJ45
Raspberry Pi 4 Model B (c) Rasberry Pi 2018
GPIO
PoE
Wifi / BT Dual band
CPU/GPU Broadcom BCM2711 ARM Cortex A72
DRAM 1 GB or 2 GB or 4 GB or 8 GB
2x USB 3.0
USB3
2x USB 2.0
USB2
2x USB3 2x USB2 controller
microSD slot on bottom side
Display DSI
Status LED's
HDMI
Camera CSI
USB-C
Power in
HDMI0 out
HDMI1 out
3.5mm out Composite Video+audio
Regulator

Figure 18: Raspberry Pi 4 Model B Platform

The Raspberry Pi (figure 18) is a low-cost, credit-card-sized computer plugged into a computer screen or TV and uses a standard keyboard and mouse. It may be a small, capable gadget that empowers individuals of all ages to investigate computing and learn how to program in languages like Python. It is a popular platform because of its low price and high embeddedness (Raspberry Pi, n.d.; Vega, 2018).

Headless Mode with Direct Ethernet Connection

Figure 19: Raspberry Pi Desktop

"Headless Mode" is a mode that can directly access the terminal or the Desktop of Raspberry Pi (figure 19) through VNC (Virtual Network Computing) over the network (Wi-Fi or Ethernet) without needing a monitor, keyboard or mouse (Dexter, n.d.). To get an Internet connection in Raspberry Pi from the PC over the Ethernet port, we need to do the following in Windows 10 (Piltch, 2020.):

1. Explore the Network Connections menu, a portion of the old-school Control Board. We can get to this screen by going to *Settings* → *Network & Internet* → *Wi-Fi*, and then "Change Adapter Settings" on the proper side of the screen. This works when we share an internet connection that comes to a PC from Wi-Fi or Ethernet.
2. Right-click on the adapter connected to the Internet and select "Properties".
3. Enable "Allow other network users to connect" on the "Sharing" tab.

4. Select the Ethernet port connected to the Raspberry Pi from the "Home networking connection" menu, and click "OK".

Subsequently, we need to establish an SSH connection:

1. Download and install Putty, the leading Windows SSH client.
2. Enter "*raspberrypi*" or "*raspberrypi. local*" or the direct IP Address of the device as the address we wish to connect to in Putty and click Open. We usually need to add the "*.local"* if the Pi is directly connected to the PC through an Ethernet cable.
3. Click "Yes" if we get a security warning alert. It is not a problem.
4. Enter "*pi*" as username and "*raspberry*" as password.

Now, we are connected at the command prompt, but if we want to access the GUI, complete with a desktop and floating windows, we will need to enable VNC:

1. Enter "*sudo raspi-config*" at the command prompt.
2. Select "*Interfacing Options*".
3. Select "*VNC*".
4. Select "*Yes*".
5. Hit Enter to acknowledge the VNC server is enabled.
6. Select "Finish".
7. On PC, Download, install and launch VNC Viewer.
8. Select "*New connection*" from the File menu.
9. Enter "*raspberry. local*" or the IP Address of Raspberry Pi direct in the "VNC Server" field.
10. Click "*OK*".
11. Double-click on the connection icon to connect.
12. Enter the Pi's username and password when prompted.
13. Click "*OK*".

Standard Utilities

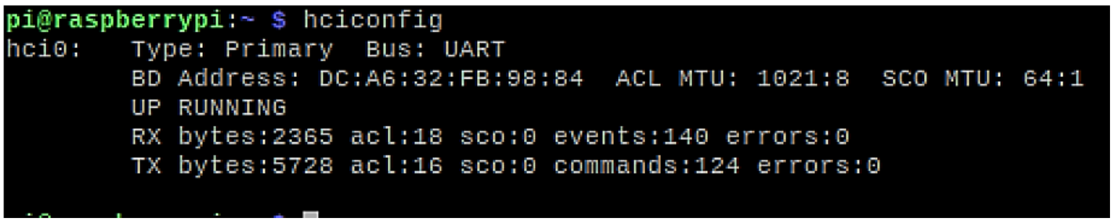
Besides Wi-Fi and Ethernet, Raspberry pi has incorporated Bluetooth, supporting BLE. It supports Bluetooth Low Energy (BLE) by integrating a combo Wi-Fi and Bluetooth chipset. It also uses Linux, leveraging the OpenSource BlueZ Bluetooth stack. This stack supports both Bluetooth Classic and BLE. BlueZ contains a collection of utilities we will use to associate and control devices (Vega, 2018).

Below (Table 23) is a list of the significant standard utilities (Raspberry Pi, n.d.):

Table 23: Standard Utilities of Raspberry Pi

Utility	Function
`hciconfig`	Configures the HCI interface to the radio
`hcitool`	Configures Bluetooth connections
`gattool`	Connect to devices and control GATT

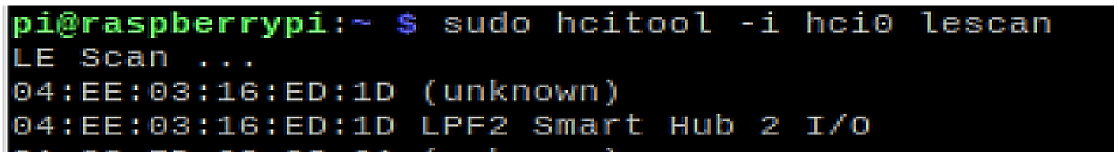
The BT adapter provides Bluetooth capability through an interface and the "`hciconfig`" utility. This utility shows (figure 20) that our interface (WeDo 2.0) is the "hci0":



```
pi@raspberrypi:~ $ hciconfig
hci0:   Type: Primary  Bus: UART
        BD Address: DC:A6:32:FB:98:84  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING
        RX bytes:2365 acl:18 sco:0 events:140 errors:0
        TX bytes:5728 acl:16 sco:0 commands:124 errors:0
```

Figure 20: Manipulation of Raspberry Pi Utility "hciconfig"

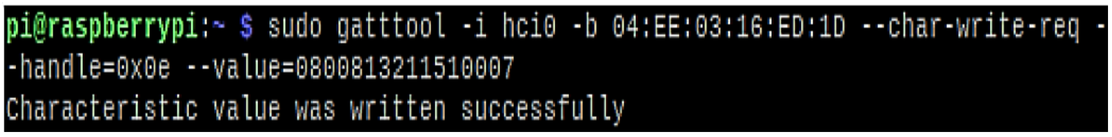
To connect to BLE devices, we need to scan using the "`hcitool`" utility (figure 21) with "hci0" as the scan and request an LE Scan. So, we find the Bluetooth MAC address of our WeDo 2.0 hub :



```
pi@raspberrypi:~ $ sudo hcitool -i hci0 lescan
LE Scan ...
04:EE:03:16:ED:1D (unknown)
04:EE:03:16:ED:1D LPF2 Smart Hub 2 I/O
```

Figure 21: Manipulation of Raspberry Pi Utility "hcitool"

Then, we use the "`gatttool`" utility (Figure 22) to manipulate the GATT of the WeDo 2.0 hub. One of the most important tasks of "`gattool`" is to read and write to characteristics:



```
pi@raspberrypi:~ $ sudo gatttool -i hci0 -b 04:EE:03:16:ED:1D --char-write-req -
-handle=0x0e --value=0800813211510007
Characteristic value was written successfully
```

Figure 22: Manipulation of Raspberry Pi "gattool"

5.1.4.2 Bluetooth Low Energy (BLE) Technology

Bluetooth Low Energy (BLE) was included in the Bluetooth standard in 2010 as a portion of the Bluetooth Core Specification version 4.0. BLE was made to empower items that would support information exchange compared to conventional Bluetooth but with lower energy utilisation, complexity and price. The lower energy utilisation is due to the littler1 information packets sent in BLE and less habitually than in conventional Bluetooth, making BLE most reasonable for devices that do not require large-scale or persistent information communications (Bhargava, 2017; Developers, n.d.).

The Bluetooth LE architecture, shown in Figure 23, is separated into three major parts: the controller, host, and application. A controller is a physical device that can transmit and get radio signals and deciphers those signals as information packets. The controller incorporates lower layers of a Bluetooth LE protocol stack. The host contains the upper layers of the BLE protocol stack. It oversees the communication between the devices (counting communication security) and decides how to get to the information transmitted by the device. The application is the top layer of the architecture and incorporates the client interface, logic, and information administration made, concurring with the procedure of the application (Bhargava, 2017; Argenox, n.d.).

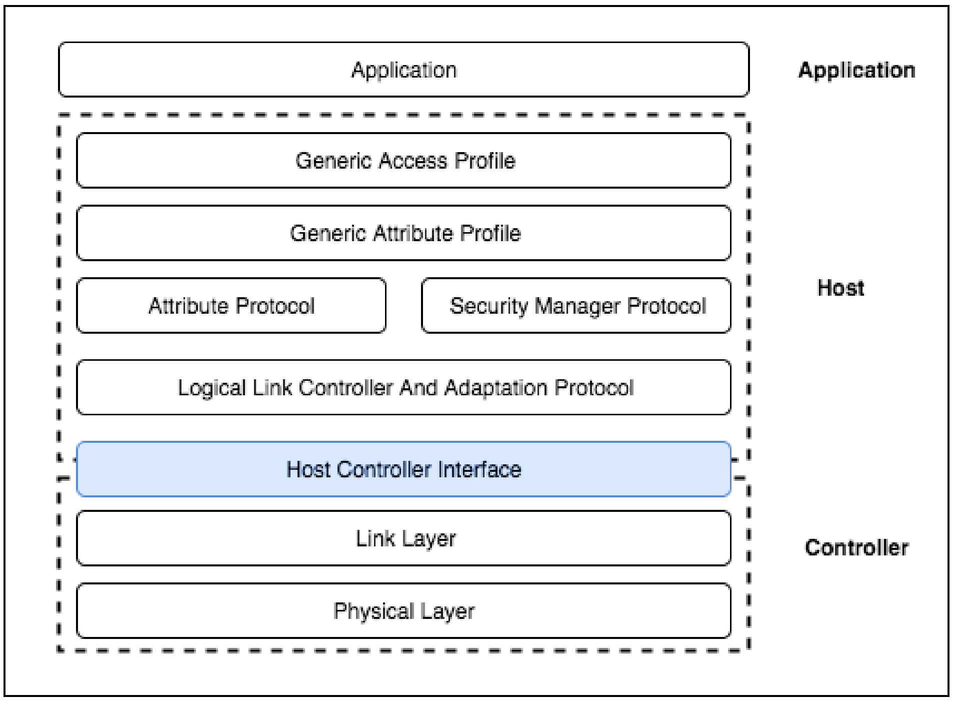

Application
Generic Access Profile
Generic Attribute Profile
Attribute Protocol
Security Manager Protocol
Logical Link Controller And Adaptation Protocol
Host Controller Interface
Link Layer
Physical Layer
Application
Host
Controller

Figure 23: Bluetooth Low Energy (BLE) Architecture (source: Bhargava, 2017)

Additionally, there is an institutionalised HCI (Host Controller Interface) protocol for communication between the host and the controller. HCI is designed to permit controllers and hosts made by diverse companies to communicate without any issues (Argenox, n.d.; Bhargava, 2017).

5.1.4.3 Generic Attribute (GATT) Profile

GATT (Generic Attribute Profile) is a component of the host. This system determines the BLE-based information model and the related methods by which devices can find, read, and adjust each other's information. GATT is built on the ATT protocol. ATT task is based on “addressable” information within the shape of attributes. The attribute is an information entity that holds data around itself and any value available through a one-of-a-kind “handle”, depending on the get-to rights of the given point. GATT makes a hierarchy that composes the subtle elements and creates a solid system for building GATT-based profiles (Adafruit, n.d.; Leonardi et al., 2018).
The arrangement appears in Figure 24 below:

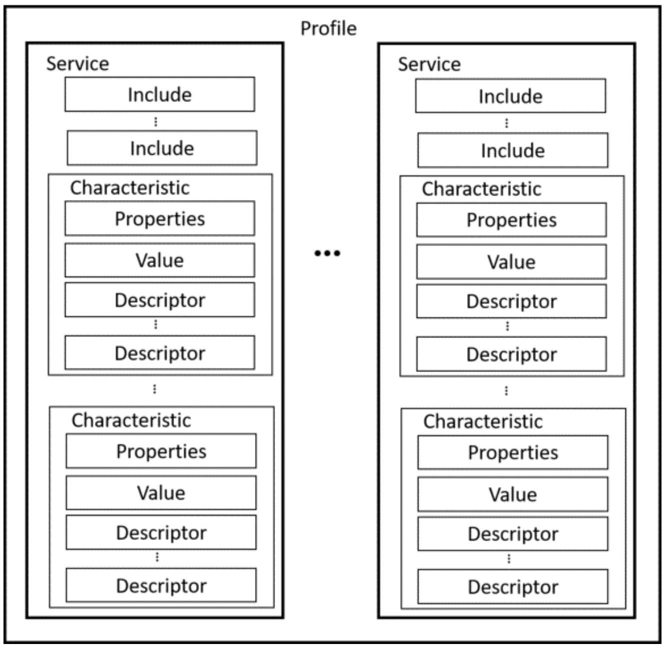


Figure 24: Generic Attribute (GATT) profile services (Source: Leonardi et al., 2018)

The attributes are gathered under GATT services. Each Service has characteristics that have descriptors. Services are gathered into characteristics that are related to a particular substantive objective. Moreover, the Service references other services utilising the attribute, whose value refers to another service. The characteristics have two attributes: an attribute that holds data almost the properties of the information in that characteristic and an attribute that stores the value of this characteristic. Also, a characteristic contains descriptors that, if accessible, give extra data around the characteristic and its information (Adafruit, n.d.; Leonardi et al., 2018).

GATT operations are entirely determined methods under which GATT-based information can be transmitted. Its operations are four: discovering services and characteristics, reading the values of "attributes" and "descriptors", writing the values of characteristics and descriptors, and updates sent by the server. There are two ways to find services; we can ask the server for a list of all services, and search for a service by the UUID (Universal Unique Identifier) of that service. There are also two ways to discover the characteristics; we ask the server for a list of all the service characteristics found or explore the feature by UUID. It is also conceivable to inquire about the server for all descriptors of the given characteristic within the case of a found characteristic (Adafruit, n.d.; Leonardi et al., 2018).

The characteristic and descriptor values can be read from their handles, and the characteristic's value can also be read from its UUID. Assume the get-to rights for a given characteristic or descriptor are such that writing values are permitted. In that case, it is conceivable to write the value of the characteristic or descriptor by sending a packet to the server containing the attribute of the value to be changed and the new value (Adafruit, n.d.; Leonardi et al., 2018).

5.1.4.4 Gattlib Python Library

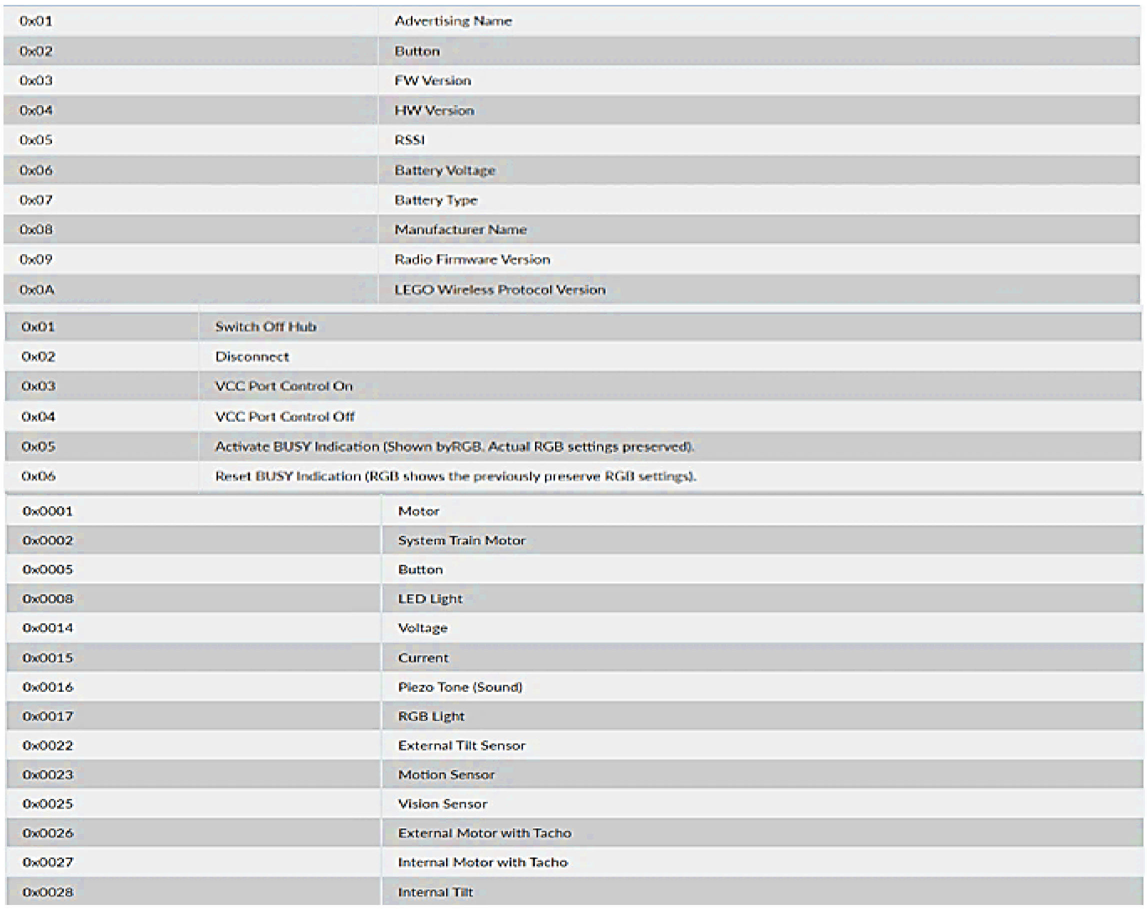
"Gattlib" is a python library (Linux only) to access Bluetooth LE devices. It utilises the command "gatttool" within the "BlueZ" package. BlueZ is a Linux operating system software that actualises the Bluetooth protocol stack. So, the gatttool in "Gattlib" can only be utilised on Linux computers. Also, this library gives two ways of work: sync and async (LabapartGitHub, n.d.; Oscaracena, n.d.).

The Bluetooth LE GATT protocol is asynchronous, so when we have to read a few values, we make a request and hold up for a response. From the programmer's viewpoint, we must pass a callback object and return it promptly when we call a read method. The response will be "injected" into that callback object. This Python library permits to call utilising a callback object (async) or without it (sync). If we do not give a callback (working sync.), the library inside will make one and hold up until a response arrives or a timeout terminates. The call will return with the received information (LabapartGitHub, n.d.; Oscaracena, n.d.).

We can utilise the "DiscoveryService" provided to discover BLE devices, which shows the Bluetooth device we need to use. Then we call the method "discover" with a timeout. It will return a "dictionary" with all devices that responded to the "discovery" and the address and name. To read data, we first need to make a "GATTRequester", passing the device's address through connecting. Then, we can read a value determined by its "handle" or its "UUID". Writing data is the same as "read". We need to make a "GATTRequest" object and utilise the method "write_by_handle" to send the data. This method will issue a written request (LabapartGitHub, n.d.; Oscaracena, n.d.).

5.1.4.5 LEGO Wireless Protocol (LWP)

The LEGO Bluetooth Hub Profile contains a single BLE Generic Attribute Profile (GATT) service. The service lets users get data regarding the LEGO Hub (name, battery level, etc.) and interact with connected sensors and motors (LegoGitHub, n.d.). Some tables from the website "LEGO GitHub" are in figure 25:



0x01	Advertising Name
0x02	Button
0x03	FW Version
0x04	HW Version
0x05	RSSI
0x06	Battery Voltage
0x07	Battery Type
0x08	Manufacturer Name
0x09	Radio Firmware Version
0x0A	LEGO Wireless Protocol Version

0x01	Switch Off Hub
0x02	Disconnect
0x03	VCC Port Control On
0x04	VCC Port Control Off
0x05	Activate BUSY Indication (Shown byRGB. Actual RGB settings preserved).
0x06	Reset BUSY Indication (RGB shows the previously preserve RGB settings).

0x0001	Motor
0x0002	System Train Motor
0x0005	Button
0x0008	LED Light
0x0014	Voltage
0x0015	Current
0x0016	Piezo Tone (Sound)
0x0017	RGB Light
0x0022	External Tilt Sensor
0x0023	Motion Sensor
0x0025	Vision Sensor
0x0026	External Motor with Tacho
0x0027	Internal Motor with Tacho
0x0028	Internal Tilt

Figure 25: LEGO Wireless Protocol (LWP) GATT services

5.2 Python Commands Development for WeDo 2.0

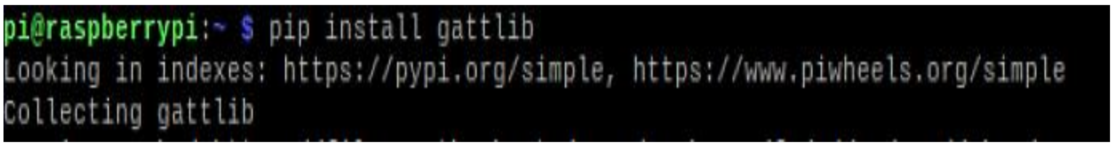
We constructed experimental commands in Python's programming language, permitting a suitable connection to the WeDo 2.0 robot using functions (Motor Activation, RGB LED Color Change, Sound Playing, Distance Sensor Activation, Tilt Sensor Activation). This would allow STEAM educational robotics to be taught with Python utilising WeDo 2.0.

Therefore, in developing Smarthub (or "brain" of the robot) commands, it was necessary to know more precisely how the commands are formed in the form of byte sequences. LEGO's GitHub profile provided a lot of information to interact with LEGO Hub through the "LEGO Wireless Protocol (LWP) GATT services" (LegoGitHub, n.d.). Also, a blog of Portuguese origin called "O Falcão" was a great help, where the blog's author had

made various attempts through the Command Prompt (CMD) to give commands to WeDo 2.0 and to read values from the sensors (Falcão, 2016).
However, it was vital to discover a Python library that could organise Bluetooth LE communication to begin to construct functions in Python Language. This library is called "Gattlib", which requires a framework with Bluetooth Low Energy (BLE) technology. So, we utilised the Raspberry Pi Platform with the created Python commands and read characteristic values from the sensors to communicate with WeDo 2.0.

5.2.1 Python Library "Gattlib" Installation

We should use the Python module "`pip`" to install the Python GATT library (gattlib), using the Python module "`pip`". So, we should use to enter "`pip install gattlib`" at the command prompt of Raspberry Pi (Figure 26):

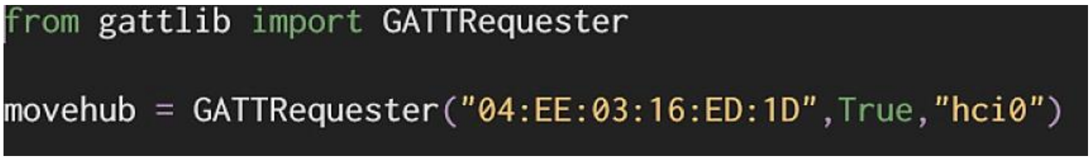


```
pi@raspberrypi:~ $ pip install gattlib
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting gattlib
```

Figure 26: Installing Python Gattlib in command prompt

5.2.2 Smarthub Connection

For the Smarthub connection, we need to create a "`GATTRequester`" object, passing the device address (each WeDo 2.0 device has a unique address) and interface "`hci0`" of WeDo 2.0 to connect (figure 27).



```
from gattlib import GATTRequester

movehub = GATTRequester("04:EE:03:16:ED:1D",True,"hci0")
```

Figure 27: Connecting WeDo 2.0 Smarthub with GATTRequester object

This process assumes that the WeDo 2.0 smarthub is kept close to the computer for maximum signal strength during the connection. Then we should press the button on Smarthub to connect.

5.2.3 WeDo 2.0 Python Functions

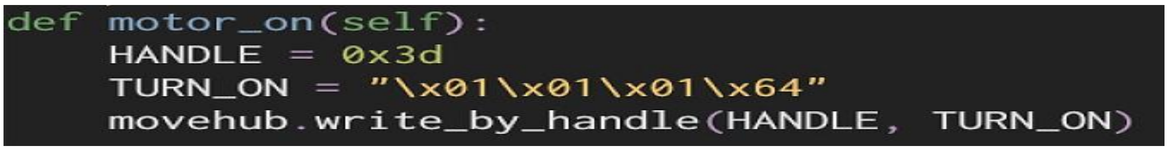
For the development of WeDo 2.0 Python Functions, we used the method "`write_by_handle`" to send the "handle" features of the device.

5.2.3.1 *Motor Activation Functions*

We used to send the data with a sequence of 4 bytes with handle features. The first byte represents the port. This means the value is 01 for Port A or 02 for Port B. The last byte activates a specific function (e.g. TURN_ON etc.) to construct motor activation functions (Figures 28 - 31).

❖ **Turn on / off**

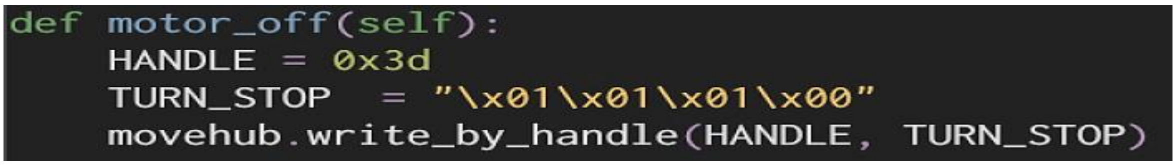
We need to call the function “ `motor_on()` ” to turn on the motor in our main program (figure 28):



```
def motor_on(self):
    HANDLE = 0x3d
    TURN_ON = "\x01\x01\x01\x64"
    movehub.write_by_handle(HANDLE, TURN_ON)
```

Figure 28: Function to turn on the motor

We need to call the function “ `motor_off()` ” to turn off the motor in our main program (figure 29):

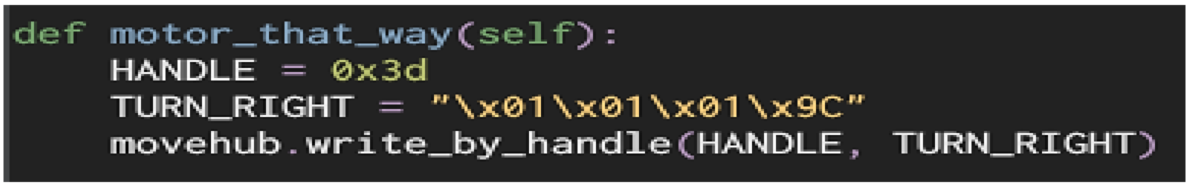


```
def motor_off(self):
    HANDLE = 0x3d
    TURN_STOP  = "\x01\x01\x01\x00"
    movehub.write_by_handle(HANDLE, TURN_STOP)
```

Figure 29: Function to turn off the motor

❖ **Turn right / left**

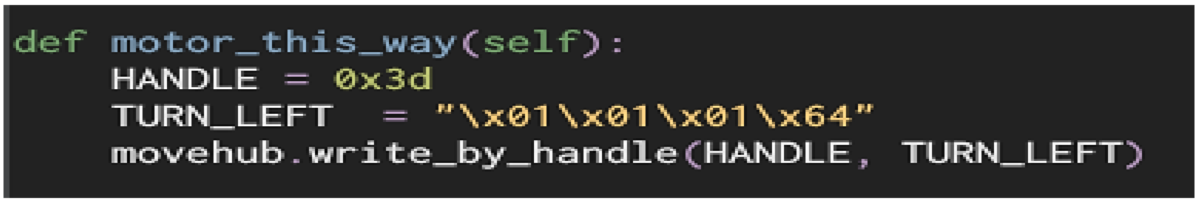
We need to call the function “ `motor_that_way()` ” to turn right the motor in our main program (figure 30):



```
def motor_that_way(self):
    HANDLE = 0x3d
    TURN_RIGHT = "\x01\x01\x01\x9C"
    movehub.write_by_handle(HANDLE, TURN_RIGHT)
```

Figure 30: Function to turn right the motor

We need to call the function “ `motor_this_way()` ” to turn left the motor in our main program (figure 31):



```
def motor_this_way(self):
    HANDLE = 0x3d
    TURN_LEFT   = "\x01\x01\x01\x64"
    movehub.write_by_handle(HANDLE, TURN_LEFT)
```

Figure 31: Function to turn left the motor

5.2.3.2 LED RGB Color Change Function

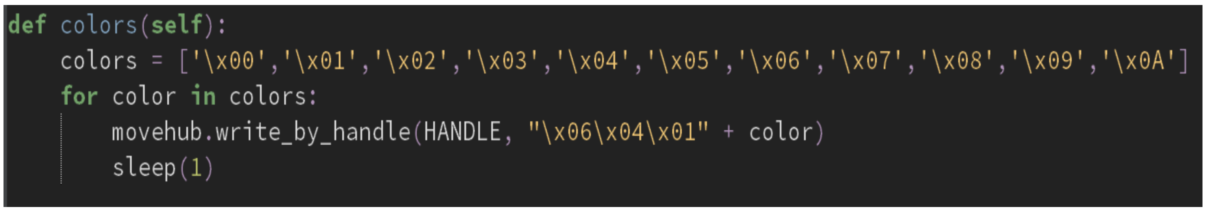
We used the handle features and 11 bytes for each colour to construct the LED RGB Colour Change Function (Figure 34). The first byte, “06”, is the LED port, the second byte is the command sent to the port, so “04” indicates “change colour”, and the third byte is the length of the arguments of the command, so “01” means that the colour is just 1-byte length. For the colours of LED, each is a different colour. They exist 11 values for each colour (Table 24):

Table 24: The 11 values of LED Colors

00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0A**
Off	Pink	Purple	Blue	Cyan	Light Green	Green	Yellow	Orange	Red	Light Blue

For instance, the handle “06040106” indicates that the Smarthub’s LED changes to Green.

So, we need to call the function “`colors()`” to change the colours of LED in our main program (figure 32):



```
def colors(self):
    colors = ['\x00','\x01','\x02','\x03','\x04','\x05','\x06','\x07','\x08','\x09','\x0A']
    for color in colors:
        movehub.write_by_handle(HANDLE, "\x06\x04\x01" + color)
        sleep(1)
```

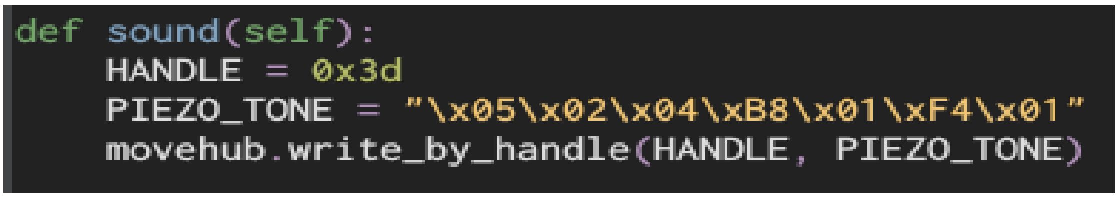
Figure 32: Function to change the colours of LED

5.2.3.4 Sound Playing Function

We controlled the sound by the same handle operated to control the motor and the RGB LED (0x003d) to construct the sound playing function (Figure 35). The “port” is “05”, and the “command” to trigger the sound is “02”, pursued by a payload of “04” bytes covering:

- the Frequency in Hz (2 bytes)
- the Duration in Ms (2 bytes)

Calling the sound function produces a piezo tone. The piezo tone alerts us when a certain threshold is met. Therefore, we need to call the function “`sound()`” to play a piezo tone in our main program (figure 33):



```
def sound(self):
    HANDLE = 0x3d
    PIEZO_TONE = "\x05\x02\x04\xB8\x01\xF4\x01"
    movehub.write_by_handle(HANDLE, PIEZO_TONE)
```

Figure 33: Function to play a piezo tone

5.2.3.5 Motion (Distance) Sensor Activation Function

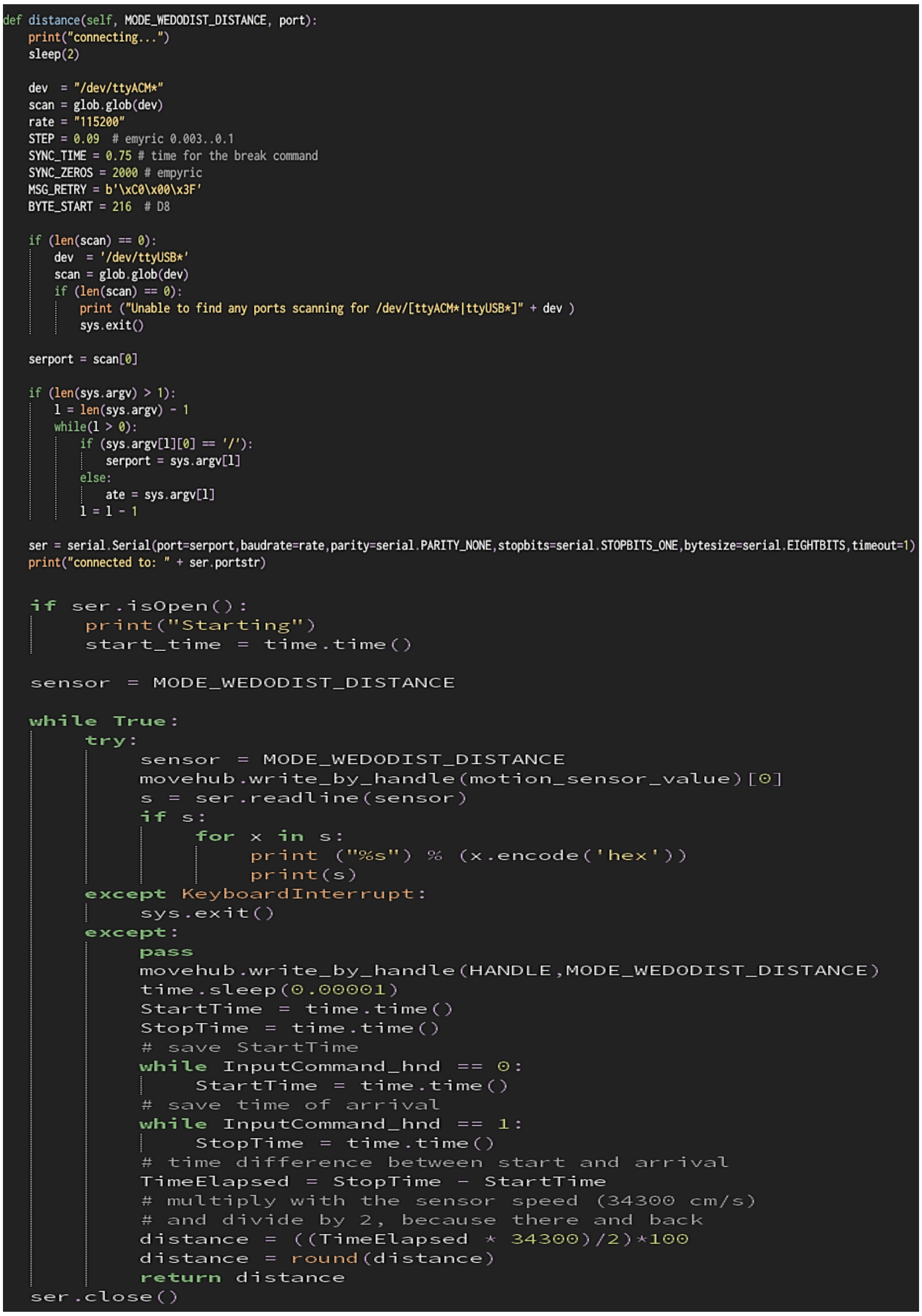
We controlled the distance sensor by the same handle operated to control the motor, the RGB LED, and the Piezo Tone, and by its operating mode value to construct the motion (distance) sensor activation function (Figure 34). Python Function based on this formula:

$$\boxed{\text{Distance} = \frac{\text{Time x Speed}}{2}}$$

Furthermore, we utilise the python library “Pyserial” to connect to WeDo 2.0 port with raspberry pi. From this library, we use three functions:

- **open ( )** → open the serial port
- **close ( )** → close the serial port
- **readline ( )** → read a string from the serial port

Then, we need to call the function “`distance()`” to measure the distance between an object in our main program:



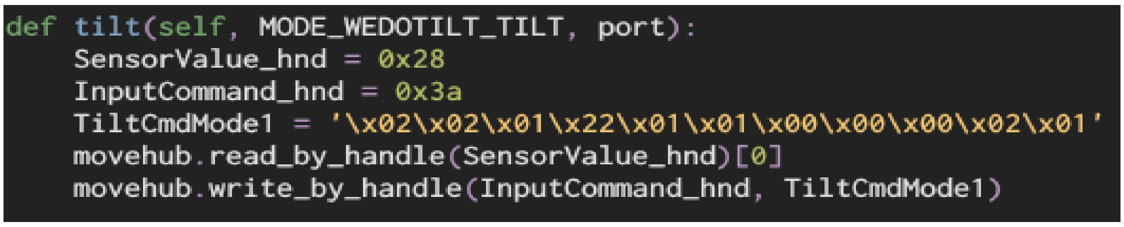
```
def distance(self, MODE_WEDODIST_DISTANCE, port):
    print("connecting...")
    sleep(2)

    dev  = "/dev/ttyACM*"
    scan = glob.glob(dev)
    rate = "115200"
    STEP = 0.09  # emyric 0.003..0.1
    SYNC_TIME = 0.75 # time for the break command
    SYNC_ZEROS = 2000 # empyric
    MSG_RETRY = b'\xC0\x00\x3F'
    BYTE_START = 216  # D8

    if (len(scan) == 0):
        dev  = '/dev/ttyUSB*'
        scan = glob.glob(dev)
        if (len(scan) == 0):
            print ("Unable to find any ports scanning for /dev/[ttyACM*|ttyUSB*]" + dev )
            sys.exit()

    serport = scan[0]

    if (len(sys.argv) > 1):
        l = len(sys.argv) - 1
        while(l > 0):
            if (sys.argv[l][0] == '/'):
                serport = sys.argv[l]
            else:
                ate = sys.argv[l]
            l = l - 1

    ser = serial.Serial(port=serport,baudrate=rate,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,bytesize=serial.EIGHTBITS,timeout=1)
    print("connected to: " + ser.portstr)

    if ser.isOpen():
        print("Starting")
        start_time = time.time()

    sensor = MODE_WEDODIST_DISTANCE

    while True:
        try:
            sensor = MODE_WEDODIST_DISTANCE
            movehub.write_by_handle(motion_sensor_value)[0]
            s = ser.readline(sensor)
            if s:
                for x in s:
                    print ("%s") % (x.encode('hex'))
                    print(s)
        except KeyboardInterrupt:
            sys.exit()
        except:
            pass
            movehub.write_by_handle(HANDLE,MODE_WEDODIST_DISTANCE)
            time.sleep(0.00001)
            StartTime = time.time()
            StopTime = time.time()
            # save StartTime
            while InputCommand_hnd == 0:
                StartTime = time.time()
            # save time of arrival
            while InputCommand_hnd == 1:
                StopTime = time.time()
            # time difference between start and arrival
            TimeElapsed = StopTime - StartTime
            # multiply with the sensor speed (34300 cm/s)
            # and divide by 2, because there and back
            distance = ((TimeElapsed * 34300)/2)*100
            distance = round(distance)
            return distance
    ser.close()
```

Figure 34: Function to measure the distance between an object

5.2.3.6 Tilt Sensor Activation Function

We controlled the tilt sensor by its handle value and operating mode value to construct the tilt sensor activation function (Figure 35).

Now, we need to call the function "`tilt()`" to measure the tilt of an object in our main program:



```
def tilt(self, MODE_WEDOTILT_TILT, port):
    SensorValue_hnd = 0x28
    InputCommand_hnd = 0x3a
    TiltCmdMode1 = '\x02\x02\x01\x22\x01\x01\x00\x00\x00\x02\x01'
    movehub.read_by_handle(SensorValue_hnd)[0]
    movehub.write_by_handle(InputCommand_hnd, TiltCmdMode1)
```

Figure 35: Function to calculate the tilt of an object

5.3 Use of the Python Functions vs Scratch with Lego WeDo 2.0

This section presents some projects using the constructed Python functions we developed versus the same programs in Scratch. Based on the lesson plan "Speed" on Lego Education Website (LegoEducation, n.d.-a), we created the programs in Scratch and Python below as examples for activities in the STEAM advanced classrooms. In the examples below, for activation of the robot car, we used the Lego WeDo 2.0 Robot Kit, the Scratch Programming Environment with the WeDo Scratch block commands, and the Raspberry Pi Platform with the WeDo Python functions (Figure 36):

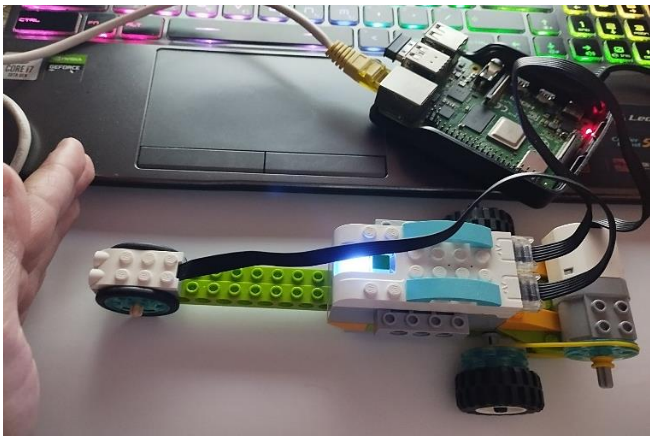


Figure 36: Using LEGO WeDo 2.0 Blocks for robot car construction in the "Speed" lesson plan framework of Lego with the Raspberry Pi Platform and the Python Functions.

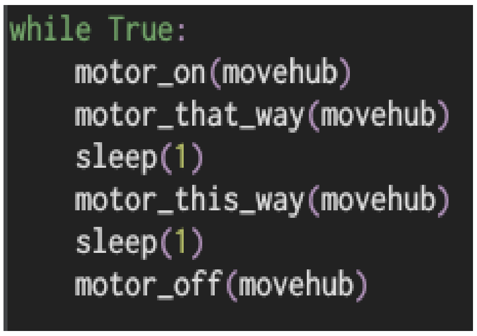
Moreover, for development, we used an infinite loop (or endless loop), a loop that is a sequence of instructions continually repeated until a specific condition is reached. Scratch has a block type for making infinite loops, called a "forever block". There is no equivalent, no clear forever block in Python, but a conditional loop is used, constantly evaluating “True”. So, while “True”, running this code is equivalent to forever. In addition, we used the “sleep” function in Python code to get the program's delay. The “sleep” function is imported from the “time” module. In Scratch code, we utilised the block “wait”. This block is used to wait for an “n” number of seconds, where “n” is any integer or float. This is known as concurrency.

Project A: Using Motor Activation Functions

In this project, we used the motor activation functions in Python (Figure 37 [A]) and Scratch (Figure 37 [B]).

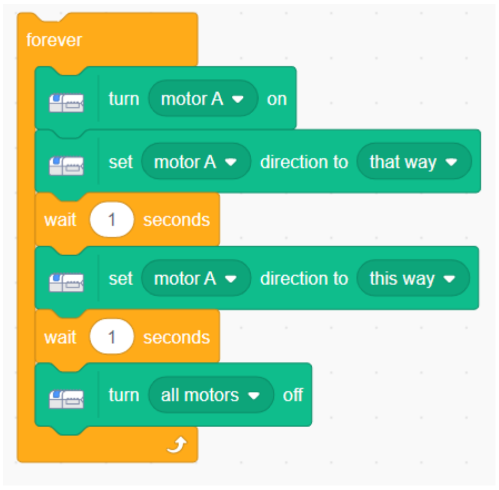
To turn on/off the Smarthub, in Python (Figure 37 [A]), we utilised the “motor_on / motor_off” functions, whereas, in Scratch (Figure 37 [B]), we used the “turn on/off” block command, in which we had to set the port with a choice: “Motor A”, “Motor B”, or “All Motors”.

Also, to turn left/right the motor, we used the “motor_this_way / motor_that_way” functions with the parameter “movehub” (which contains the connection with Smarthub of WeDo 2.0) in Python. In contrast, in Scratch, we utilised the “set [Motor A/B/All] direction to that / this way” block command. Clearly, in Scratch, we should set the motor's port and then the motor's direction left or right, contrasting with Python code.



```
while True:
    motor_on(movehub)
    motor_that_way(movehub)
    sleep(1)
    motor_this_way(movehub)
    sleep(1)
    motor_off(movehub)
```

(A)



```
forever
    turn motor A on
    set motor A direction to that way
    wait 1 seconds
    set motor A direction to this way
    wait 1 seconds
    turn all motors off
```

(B)

Figure 37: Motor Activation Functions: (A) Python (B) Scratch

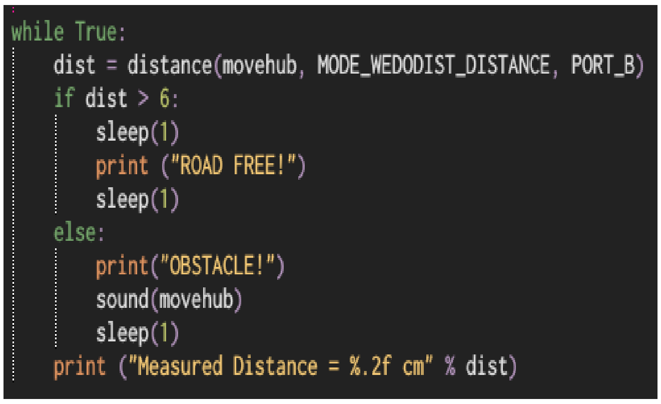
Project B: Using Distance Sensor Activation Function

In this project, we used the distance sensor activation function in Python (Figure 38 [A]) and Scratch (Figure 38 [B]).

To measure the distance between an object, in Python code (Figure 38 [A]), we used the "distance" function with parameters the connection of Smarthub (movehub), the mode of distance sensor (handle) and the port (A or B). Similarly, we utilised the "distance" block command in Scratch code (Figure 38 [B]).

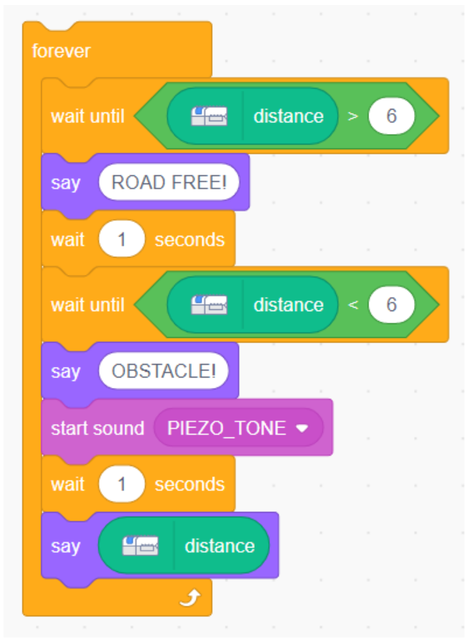
Also, in Python code, we used a condition with an "if" statement to control the distance measurement so that if the "distance of the object is bigger than 6" from the robot, output to the screen the message "ROAD FREE!", that is, the road has not some obstacle, differently if the distance of the object is smaller than 6" from the robot, output to the screen the message "OBSTACLE!", that is, it detected some obstacle. In contrast, in Scratch, we utilised the "wait until" block command, which is a control block. This block pauses its script until the specific "Boolean" condition is "true". In Scratch, the control of distance measurement becomes with a similar way to Python code. However, in Scratch, to output to the screen should create a sprite talk by utilising the "say" block command.

Moreover, to produce a piezo tone alerting when an obstacle is detected, in Python, we used the function "sound". In Scratch, we used the "start sound" block command, a Scratch command but not a specific WeDo 2.0 command, such as in Python code.



```
while True:
    dist = distance(movehub, MODE_WEDODIST_DISTANCE, PORT_B)
    if dist > 6:
        sleep(1)
        print ("ROAD FREE!")
        sleep(1)
    else:
        print("OBSTACLE!")
        sound(movehub)
        sleep(1)
    print ("Measured Distance = %.2f cm" % dist)
```

(A)

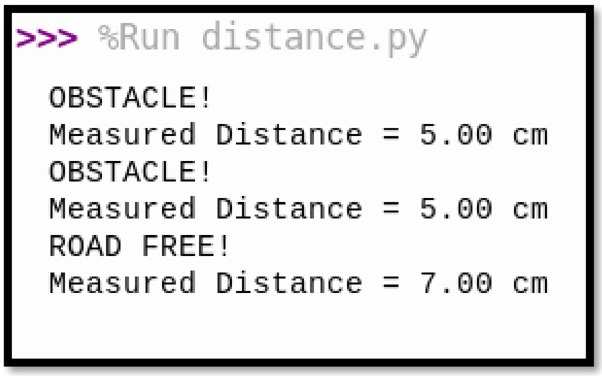


```
forever
    wait until < (distance) > 6 >
    say ROAD FREE!
    wait 1 seconds
    wait until < (distance) < 6 >
    say OBSTACLE!
    start sound PIEZO_TONE
    wait 1 seconds
    say (distance)
```

(B)

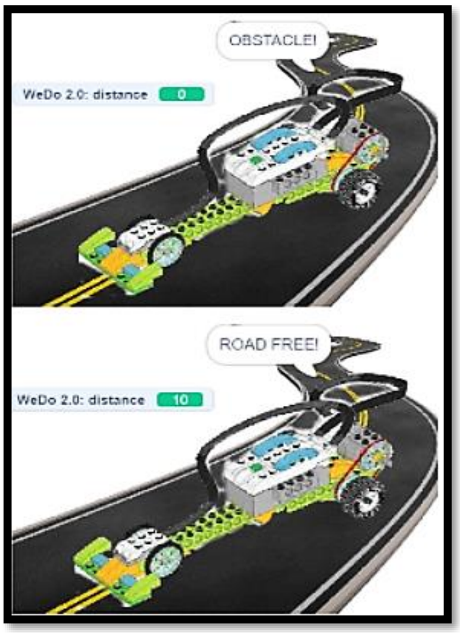
Figure 38: Distance Sensor Activation Function : (A) Python (B) Scratch

In addition, at the end of the code in both languages, output to the screen the distance measurement, showing up an “n” number in centimetre (cm). In Python, the measurement is float numbers (Figure 39 [A]), whereas, in Scratch, the numbers are integers (Figure 39 [B]).



```
>>> %Run distance.py

  OBSTACLE!
  Measured Distance = 5.00 cm
  OBSTACLE!
  Measured Distance = 5.00 cm
  ROAD FREE!
  Measured Distance = 7.00 cm
```

(A)



(B)

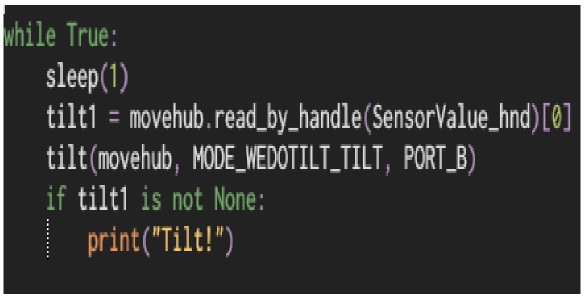
Figure 39: Output to the screen: (A) Python (B) Scratch

<u>Project C: Using Tilt Sensor Activation Function</u>

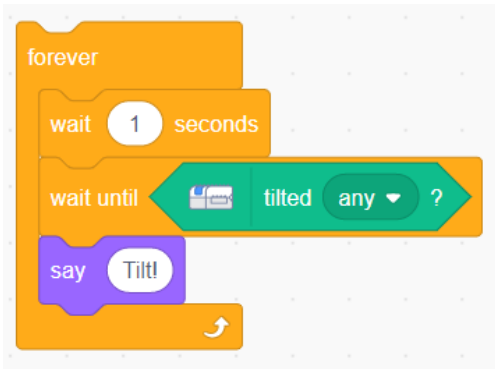
In this project, we used the tilt sensor activation function in Python (Figure 40 [A]) and Scratch (Figure 40 [B]).

To measure the tilt of the robot, in Python code (Figure 40 [A]), we used the “tilt” function with parameters the connection of Smarthub (movehub), the mode of tilt sensor (handle) and the port (A or B). In Scratch code (Figure 40 [B]), we utilised the “wait until” block command to activate the “tilt” block command only when the sensor detects some slope.

In addition, when the sensor detects any tilt in both codes, it outputs the message “Tilt!” to the screen.



```
while True:
    sleep(1)
    tilt1 = movehub.read_by_handle(SensorValue_hnd)[0]
    tilt(movehub, MODE_WEDOTILT_TILT, PORT_B)
    if tilt1 is not None:
        print("Tilt!")
```



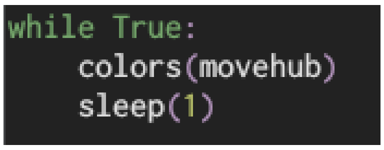
```
forever
    wait 1 seconds
    wait until tilted any ?
    say Tilt!
```

Figure 40: Tilt Sensor Activation Function : (A) Python (B) Scratch

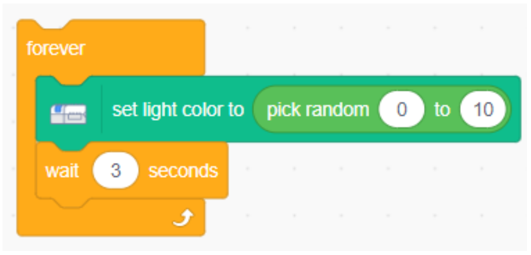
Project D: Using LED RGB Color Change Function

In this project, we used the LED RGB colour change function in Python (Figure 41 [A]) and Scratch (Figure 41 [B]).

To change the LED colours, in Python code (Figure 41 [A]), we used the "colors" function to change the LED colours with the parameter of the connection of Smarthub (movehub). In Scratch code, we utilised the "set light color to" block command.



```
while True:
    colors(movehub)
    sleep(1)
```



```
forever
    set light color to pick random 0 to 10
    wait 3 seconds
```

Figure 41: LED RGB Colour Change Function: (A) Python (B) Scratch

Additionally, in Scratch code (Figure 41 [B]), we used a further block command, the "pick random 0 to 10", which changes randomly to 11 colours of the LED, contrasting Python code, in which the colours change automated (Figure 42):

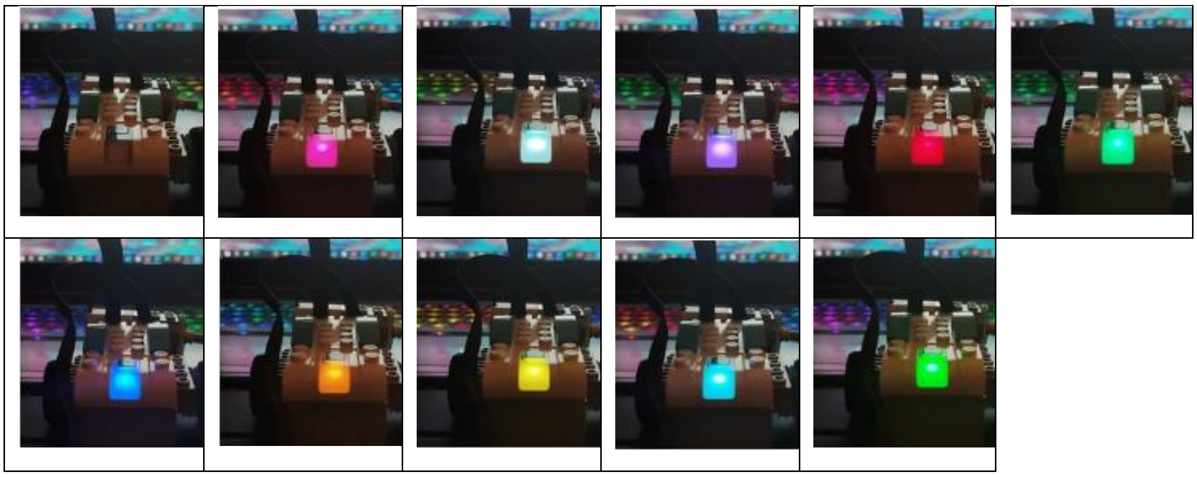


Figure 42: The Output of the 11 LED Colors of WeDo 2.0 with Python Code

Chapter 6 - Conclusion

Through investigations and various experimentations, the lessons with robotic assignments with Lego WeDo 2.0 motivated students, and they also effectively comprehended the fundamental principles of STEAM (Üşengül & Bahçeci, 2020). They are typically done using motorised LEGO models and straightforward programming. WeDo 2.0 bolsters a hands-on, "minds-on" knowledge solution that offers students the confidence to make questions and instruments to discover the answers and solve real-life issues.

As described in the previous chapters, the dissertation's essential subject was moving from Scratch to Python programming. For this reason, experimental commands/functions for Lego WeDo 2.0 in Python were developed for STEAM robotics learning in advanced classes.

The first argument of the thesis was comparing and contrasting differences in the language rules of Scratch and Python. This argument was significant for smoothly shifting from one programming language to another and assisting students effectively. With Scratch coding, doing and learning from mistakes is not as predominant, as students work inside the boundaries of predefined squares of code and cannot exceed those boundaries. Students using novice programming, such as Scratch, gain an incorrect impression of coding because this language is more for beginners and not advanced students (Moors et al., 2018). Over time, students desire to create more complex projects, so Scratch does not cover them (Papadakis & Orfanakis, 2018). In contrast, Python coding presents numerous real-world challenges that assist students in learning how to solve issues. By learning real-world programming, such as Python, they can apply it in their future careers.

The second argument of the thesis was the construction of experimental functions in Python through a Raspberry Pi Platform based on Scratch for operating WeDo 2.0. With educational robots, like WeDo 2.0, students illustrate improvement, show "positive attitudes" toward STEAM, select engineering as a principal, and join an "iterative design method" (Majgaard & Mc-Kinney, 2010; Papadakis, Vaiopoulou, Sifaki, Stamovlasis, & Kalogiannakis, 2021). These constructed functions will assist the students in noticing the abstract concepts of Scratch and Python programming in practice and allow STEAM educational robotics to be taught with Python utilising WeDo 2.0. This will give in students a “feedback-oriented learning environment”, a “collaborative working environment”, and openings to work and investigate arrangements for real-world issues.

6.1 Future Work

Future work should focus on testing the constructed Python functions in STEAM classrooms and extracting conclusions about the effectiveness of the learning procedure and the problem-solving methods students utilised on the different assignments or the kinds of blunders that students made on the programming assignments. This could permit educators to construct suitable activities for advanced students (Papadakis & Kalogiannakis, 2022). In addition, experts could evaluate the implementation of the commands/functions to STEAM Education, Programming and Pedagogy.

Moreover, many parts of this work can be upgraded and reconfigured. It could upgrade the tilt function to offer more than one orientation and calculate the angle of these. Another upgrade that can be done is to construct more functions for the motor to control the low/high intensity and to reverse direction.

References

Ackermann, E. (2001). Piaget's Constructivism, Papert's Constructionism: What's the difference?

Adafruit, L. S. (n.d.). GATT | Introduction to Bluetooth Low Energy | Adafruit Learning System. Retrieved 17 March 2022, from https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

Afari, E., & Khine, M. S. (2017). Robotics as an Educational Tool: Impact of Lego Mindstorms. *International Journal of Information and Education Technology*, 7(6), 437–442. Retrieved from https://doi.org/10.18178/IJIET.2017.7.6.908

Almeida, T. O., De Netto, J. F. M., & Rios, M. L. (2017). Remote robotics laboratory as support to teaching programming. *Proceedings - Frontiers in Education Conference, FIE*, 2017-Octob, 1–6. Retrieved from https://doi.org/10.1109/FIE.2017.8190472

Ampartzaki, M., Kalogiannakis, M., Papadakis, S., & Giannakou, V. (2022). Perceptions About STEM and the Arts: Teachers', Parents' Professionals' and Artists' Understandings About the Role of Arts in STEM Education. *Lecture Notes in Educational Technology*, 601–624. Retrieved 10 July 2022 from https://doi.org/10.1007/978-981-19-0568-1_25/COVER/

An, S. (2020). The impact of STEAM integration on preservice teachers' disposition and knowledge. *Journal of Research in Innovative Teaching & Learning*, 13(1), 27–42. Retrieved from https://doi.org/10.1108/jrit-01-2020-0005

Argenox. (n.d.). Using BLE Devices with a Raspberry Pi. Retrieved 23 January 2022, from https://www.argenox.com/library/bluetooth-low-energy/using-raspberry-pi-ble/

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to 'Real' programming. *ACM Transactions on Computinig Education*, 14(4). Retrieved from https://doi.org/10.1145/2677087

Astuti, Nurul Heni; Rusilowati, Ani; Subali, B. (2021). View of STEM-Based Learning Analysis to Improve Students' Problem Solving Abilities in Science Subject: a Literature Review. Retrieved 23 June 2022, from https://journal.unnes.ac.id/sju/index.php/jise/article/view/38505/16063

Bart, A. C., Tibau, J., Tilevich, E., Shaffer, C. A., & Kafura, D. (2017). BlockPy: An Open Access Data-Science Environment for Introductory Programmers. *Computer*, 50(5), 18–26. Retrieved from https://doi.org/10.1109/MC.2017.132

Benotti, L., Gómez, M. J., & Martínez, C. (2017). UNC++Duino: A kit for learning to program robots in python and C++ starting from blocks. *Advances in Intelligent Systems and Computing*, 457, 181–192. Retrieved from https://doi.org/10.1007/978-3-319-42975-5_17

Bhargava, M. (2017). Architecture of Bluetooth Low Energy | IoT Projects with Bluetooth Low Energy. Retrieved 19 March 2022, from https://subscription.packtpub.com/book/hardware_and_creative/9781788399449/1/01lvl1sec11/architecture-of-bluetooth-low-energy

Blanchard, J., Gardner-Mccune, C., & Anthony, L. (2019). Effects of Code Representation on Student Perceptions and Attitudes Toward Programming. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 2019-Octob, 127–131. Retrieved from https://doi.org/10.1109/VLHCC.2019.8818762

Blank, D., Meeden, L., & Kumar, D. (2003). Python robotics, (January), 317. Retrieved from https://doi.org/10.1145/611993.611996

Catlin, D., & Woollard, J. (2014). Educational Robots and Computational Thinking. In *Proceedings of 4th International workshop teaching robotics, teaching with robotics & 5th International conference robotics in education* (pp. 144–151).

Chalmers, C. (2018). Robotics and computational thinking in primary school. *International Journal of Child-Computer Interaction*, 17, 93–100. Retrieved from https://doi.org/10.1016/j.ijcci.2018.06.005

Dell'Erba, M. (2019). Preparing Students for Learning, Work and Life through STEAM Education. Policy Brief, Education Commission of the States, 2019-Sep. *Education Commission of the States*. Retrieved 13 July 2022 from https://eric.ed.gov/?id=ED598088

Developers, A. (n.d.). Bluetooth Low Energy | Android Developers. Retrieved 17 January 2022, from https://developer.android.com/guide/topics/connectivity/bluetooth/ble-overview

Dexter, I. (n.d.). Connecting to Raspberry Pi without a monitor for Beginners. Retrieved 23 June 2022, from https://www.dexterindustries.com/howto/connecting-raspberry-pi-without-monitor-beginners/

Dorling, M., & White, D. (2015). Scratch: A way to logo and python. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 191–196. Retrieved from https://doi.org/10.1145/2676723.2677256

Elkin, M., Sullivan, A., & Umashi Bers, M. (2014). Implementing a Robotics Curriculum in an Early Childhood Montessori Classroom. *Journal of Information Technology Education: Innovations in Practice*, 13, 153–169. Retrieved from https://doi.org/10.28945/2094

Erba, M. D. (2019). Policy Considerations for STEAM Education - Education Commission of the States. Retrieved 13 March 2022, from https://www.ecs.org/policy-considerations-for-steam-education/

Evripidou, S., Georgiou, K., Doitsidis, L., Amanatiadis, A. A., Zinonos, Z., & Chatzichristofis, S. A. (2020). Educational Robotics: Platforms, Competitions and Expected Learning Outcomes. *IEEE Access*, 8. Retrieved from https://doi.org/10.1109/ACCESS.2020.3042555

Falcão, O. (2016). WeDo 2.0 – reverse engineering – O Falcão. Retrieved 15 May 2022, from https://ofalcao.pt/blog/series/wedo-2-0-reverse-engineering

Herro, D., Quigley, C., & Cian, H. (2018). The Challenges of STEAM Instruction: Lessons from the Field. *Https://Doi.Org/10.1080/01626620.2018.1551159*, 41(2), 172–190. Retrieved 13 July 2022 from https://doi.org/10.1080/01626620.2018.1551159

Hodges, S., Sentance, S., Finney, J., & Ball, T. (2020). Physical Computing: A Key Element of Modern Computer Science Education. *Computer*, 53(4), 20–30. Retrieved from https://doi.org/10.1109/MC.2019.2935058

Huang, K.H., Yang, T.M., & Cheng, C. ., Huang, K. H., Yang, T. M., & Cheng, C. C. (2013). Engineering to see and move: Teaching computer programming with flowcharts vs. LEGO robots. *International Journal of Emerging Technologies in Learning*, 8(4), 23–26. Retrieved 3 January 2022 from https://doi.org/10.3991/ijet.v8i4.2943

Huei, Y. C. (2015). Benefits and introduction to python programming for freshmore students using inexpensive robots. *Proceedings of IEEE International Conference on Teaching, Assessment and Learning for Engineering: Learning for the Future Now, TALE 2014*, (December), 12–17. Retrieved from https://doi.org/10.1109/TALE.2014.7062611

Juškevičienė, A., Stupurienė, G., & Jevsikova, T. (2020). Computational thinking development through physical computing activities in STEAM education. Retrieved from https://doi.org/10.1002/cae.22365

Kalovrektis, K., Papageorgiou, T., Psycharis, S., Theodorou Telecommunications, P., Vasiliki Ntourou, G., & Xenakis, A. (2021). The Impact of Physical Computing and Computational Pedagogy on Girl’s Self- Efficacy and Computational Thinking Practice. *IEEE Global Engineering Education Conference (EDUCON)*. Retrieved 15 March 2022 from https://doi.org/10.1109/EDUCON46332.2021.9454003

Khamphroo, M., Kwankeo, N., Kaemarungsi, K., & Fukawa, K. (2017). MicroPython-based educational mobile robot for computer coding learning. *8th International Conference on Information and Communication Technology for Embedded Systems, IC-ICTES 2017 - Proceedings*. Retrieved from https://doi.org/10.1109/ICTEmSys.2017.7958781

Khanlari, A. (2014). Effects of educational robots on learning STEM and on students’ attitude toward STEM. *2013 IEEE 5th International Conference on Engineering Education: Aligning Engineering Education with Industrial Needs for Nation Development, ICEED 2013*, 62–66. Retrieved from https://doi.org/10.1109/ICEED.2013.6908304

Kim, T., & Han, S. (2018). Development of Python Education Program for Block Coding Learners. *Journal of The Korean Association of Information Education*, 22(1), 53–60. Retrieved from https://doi.org/10.14352/jkaie.2018.22.1.53

Korkmaz, Ö. (2016). The Effect of Scratch- and Lego Mindstorms Ev3-Based Programming Activities on Academic Achievement, Problem-Solving Skills and Logical-Mathematical Thinking Skills of Students. *Malaysian Online Journal of Educational Sciences*, 4(3), 73–88.

LabapartGitHub. (n.d.). labapart/gattlib: Library to access GATT information from BLE (Bluetooth Low Energy) devices. Retrieved 17 March 2022, from https://github.com/labapart/gattlib

Lamb, A., & Johnson, L. (2011). Scratch: Computer Programming for 21st Century Learners. *Teacher Librarian*, 38(4), 64–68. Retrieved 13 March 2022 from https://scholarworks.iupui.edu/bitstream/handle/1805/8622/38-4.pdf?sequence=1

Lawhead, P. B., Duncan, M. E., Bland, C. G., Goldweber, M., Schep, M., Barnes, D. J., & Hollingsworth, R. G. (2003). A road map for teaching introductory programming using LEGO© mindstorms robots. *ACM SIGCSE Bulletin*, 35(2), 191–201. Retrieved from https://doi.org/10.1145/782941.783002

LegoEducation. (n.d.-a). Speed | WeDo 2.0 Lesson Plan | LEGO® Education. Retrieved 5 July 2022, from https://education.lego.com/en-us/lessons/wedo-2-science/speed/student-worksheet

LegoEducation. (n.d.-b). WeDo 2.0 Support | Everything You Need | LEGO® Education. Retrieved 8 January 2022, from https://education.lego.com/en-us/product-resources/wedo-2/teacher-resources/teacher-guides

LegoGitHub. (n.d.). LEGO Wireless Protocol 3.0.00 Doc v3.0.00 r17 documentation. Retrieved 17 March 2022, from https://lego.github.io/lego-ble-wireless-protocol-docs/index.html#document-1-Introduction

Leonardi, L., Patti, G., & Lo Bello, L. (2018). Multi-Hop Real-Time Communications over Bluetooth Low Energy Industrial Wireless Mesh Networks. *IEEE Access*, 6(September), 26505–26519. Retrieved from https://doi.org/10.1109/ACCESS.2018.2834479

Lorenz, T. (2018). The Importance of Debugging. Retrieved 15 March 2022, from https://blog.bitlabstudio.com/the-importance-of-debugging-886df73427ea

Majgaard, G., & Mc-Kinney, M. (2010). Design-based Action research in the World of Robot Technology and Learning. In *The Third IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning: DIGITAL 2010. IEEE Press* (pp. 85–92). IEEE Press. Retrieved 14 August 2022 from www.playalive.dk

Mayerové, K., & Veselovská, M. (2017). How to teach with LEGO WeDo at primary school. *Advances in Intelligent Systems and Computing*, 457, 55–62. Retrieved from https://doi.org/10.1007/978-3-319-42975-5_5

McGregor, D. (2007). Developing Thinking; Developing Learning. *British-Journal-of-Educational-Studies*, 466–468. Retrieved from https://doi.org/http://dx.doi.org/10.1111/j.1467-8527.2007.00388_2.x

Moors, L., Luxton-Reilly, A., & Denny, P. (2018). Transitioning from Block-Based to Text-Based Programming Languages. *Proceedings - 2018 6th International Conference on Learning and Teaching in Computing and Engineering, LaTiCE 2018*, (April 2018), 57–64. Retrieved from https://doi.org/10.1109/LaTICE.2018.000-5

Murcia, K., Pepper, C., Joubert, M., Cross, E., & Wilson, S. (2020). A framework for identifying and developing children's creative thinking while coding with digital technologies. *Issues in Educational Research*, 30(4), 1395–1417.

Noviani, S., & Wangid, M. N. (2018). Developing Inquiry-Based Lectora Multimedia in Order to Increase the Logical Ability and the Creative Thinking. *Jurnal Prima Edukasia*, 6(1), 89–101. Retrieved from https://doi.org/10.21831/jpe.v6i1.9653

Olabe, J. C., Olabe, M. A., Basogain, X., Maiz, I., & Castaño, C. (2011). Programming and Robotics with Scratch in Primary Education. *Education in a Technological World: Communicating Current and Emerging Research and Technological Efforts*, (July 2016), 356–363.

Oscaracena, G. (n.d.). oscaracena/pygattlib. Retrieved 17 March 2022, from https://github.com/oscaracena/pygattlib

Papadakis, S., & Kalogiannakis, M. (2022). Learning Computational Thinking Development in Young Children With Bee-Bot Educational Robotics. *Research Anthology on Computational Thinking, Programming, and Robotics in the Classroom*, 2, 926–947. Retrieved from https://doi.org/10.4018/978-1-6684-2411-7.CH040

Papadakis, S., Kalogiannakis, M., Zaranis, N., & Orfanakis, V. (2016). Using Scratch and App Inventor for teaching introductory programming in secondary education. A case study. *International Journal of Technology Enhanced Learning*, 8(3–4), 217–233. Retrieved from https://doi.org/10.1504/IJTEL.2016.082317

Papadakis, S., & Orfanakis, V. (2018). Comparing novice programing environments for use in secondary education: App Inventor for Android vs. Alice. *International Journal of Technology Enhanced Learning*, 10(1–2), 44–72. Retrieved from https://doi.org/10.1504/IJTEL.2018.088333

Papadakis, S., Vaiopoulou, J., Sifaki, E., Stamovlasis, D., & Kalogiannakis, M. (2021). Attitudes towards the use of educational robotics: Exploring pre-service and in-service early childhood teacher profiles. *Education Sciences*, 11(5). Retrieved from https://doi.org/10.3390/educsci11050204

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, New York, USA: Basic Books.

Pham, Q. T., Misra, S., Huynh, L. N. H., & Ahuja, R. (2019). *Investigating Enterprise Resource Planning (ERP) Effect on Work Environment BT - Computational Science and Its Applications – ICCSA 2019* (Vol. 1). Springer International Publishing. Retrieved from https://doi.org/10.1007/978-3-030-24308-1

Piltch, A. (2020). How to Set Up a Headless Raspberry Pi, No Monitor Needed | Tom's Hardware. Retrieved 23 June 2022, from https://www.tomshardware.com/reviews/raspberry-pi-headless-setup-how-to,6028.html

Plaza, P., Castro, M., Merino, J., Restivo, T., Peixoto, A., Gonzalez, C., … Strachan, R. (2020). Educational Robotics for All: Gender, Diversity, and Inclusion in STEAM. *Proceedings of 2020 IEEE Learning With MOOCS, LWMOOCS 2020*, 19–24. Retrieved from https://doi.org/10.1109/LWMOOCS50143.2020.9234372

Plaza, P., Sancristobal, E., Carro, G., Castro, M., & Blazquez, M. (2018). Scratch day to introduce robotics. *IEEE Global Engineering Education Conference, EDUCON*, 2018-April, 208–216. Retrieved from https://doi.org/10.1109/EDUCON.2018.8363230

Przybylla, M., & Romeike, R. (2014). Key Competences with Physical Computing. In Brinda, T., Reynolds, N., Romeike, R. & Schwill, A. (Eds.) KEYCIT 2014 Key competences in informatics and ICT. In *University of Potsdam, Germany*. Retrieved 3 January 2022 from https://publishup.uni-potsdam.de/opus4-ubp/frontdoor/deliver/index/docId/7032/file/cid07.pdf

Przybylla, M., & Romeike, R. (2014). Physical computing and its scope - towards a constructionist computer science curriculum with physical computing. *Informatics in Education*, 13(2), 241–254. Retrieved from https://doi.org/10.15388/INFEDU.2014.05

Psycharis, S. (2018). Computational Thinking, Engineering Epistemology and STEM Epistemology: A Primary Approach to Computational Pedagogy. *Advances in Intelligent Systems and Computing*, 917, 689–698. Retrieved 14 March 2022 from https://doi.org/10.1007/978-3-030-11935-5_65

Psycharis, S., Kalovrektis, K., & Xenakis, A. (2020). A Conceptual Framework for Computational Pedagogy in STEAM education: Determinants and perspectives. *Hellenic Journal of STEM Education*, 1(1), 17–32. Retrieved 14 March 2022 from https://doi.org/10.51724/HJSTEMED.V1I1.4

Python, D. (n.d.). turtle — Turtle graphics — Python 3.10.5 documentation. Retrieved 13 July 2022, from https://docs.python.org/3/library/turtle.html

Raspberry Pi, F. (n.d.). Teach, Learn, and Make with Raspberry Pi. Retrieved 17 January 2022, from https://www.raspberrypi.org/

Rossum, G. Van, Foundation, P. S., Foundation, P. S., Python, S., & Groovy, A. (n.d.). Python (programming language).

Ruzzenente, M., Koo, M., Nielsen, K., Grespan, L., & Fiorini, P. (2012). A Review of Robotics Kits for Tertiary Education. *Proceedings of 3rd International Workshop Teaching Robotics, Teaching with Robotics Integrating Robotics in School Curriculum*, 153–162.

Scratch MIT, W. (n.d.). Scratch Wiki. Retrieved 16 March 2022, from https://en.scratch-wiki.info/

Sisman, B., & Kucuk, S. (2019). An Educational Robotics Course: Examination of Educational Potentials and Pre-service Teachers' Experiences An Educational Robotics Course: Examination of Educational Potentials and Pre-service Teachers' Experiences. *International Journal of Research in Education and Science (IJRES)*, 5(2), 510–531. Retrieved 13 March 2022 from www.ijres.net

Sutradhar, P., & Naraginti, A. R. (2022). Teaching Effectiveness of Science, Technology, Engineering, and Mathematics (STEM) Teachers. *SSRN Electronic Journal*, 1–11. Retrieved from https://doi.org/10.2139/ssrn.4096832

Times, T. E. (n.d.). What is Debugging? Definition of Debugging, Debugging Meaning - The Economic Times. Retrieved 3 January 2022, from https://economictimes.indiatimes.com/definition/debugging

Üşengül, L., & Bahçeci, F. (2020). The Effect of Lego Wedo 2.0 Education on Academic Achievement and Attitudes and Computational Thinking Skills of Learners toward Science. *World Journal of Education*, 10(4), 83. Retrieved from https://doi.org/10.5430/WJE.V10N4P83

Vega, J. (2018). Educational Framework Using Robots with Vision for Constructivist Teaching Robotics to Pre-university Students, 1–208.

Vihavainen, A., Paksula, M., & Luukkainen, M. (2011). Extreme apprenticeship method in teaching programming for beginners. *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 93–98. Retrieved from https://doi.org/10.1145/1953163.1953196

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education*, 18(1), 1–25. Retrieved from https://doi.org/10.1145/3089799

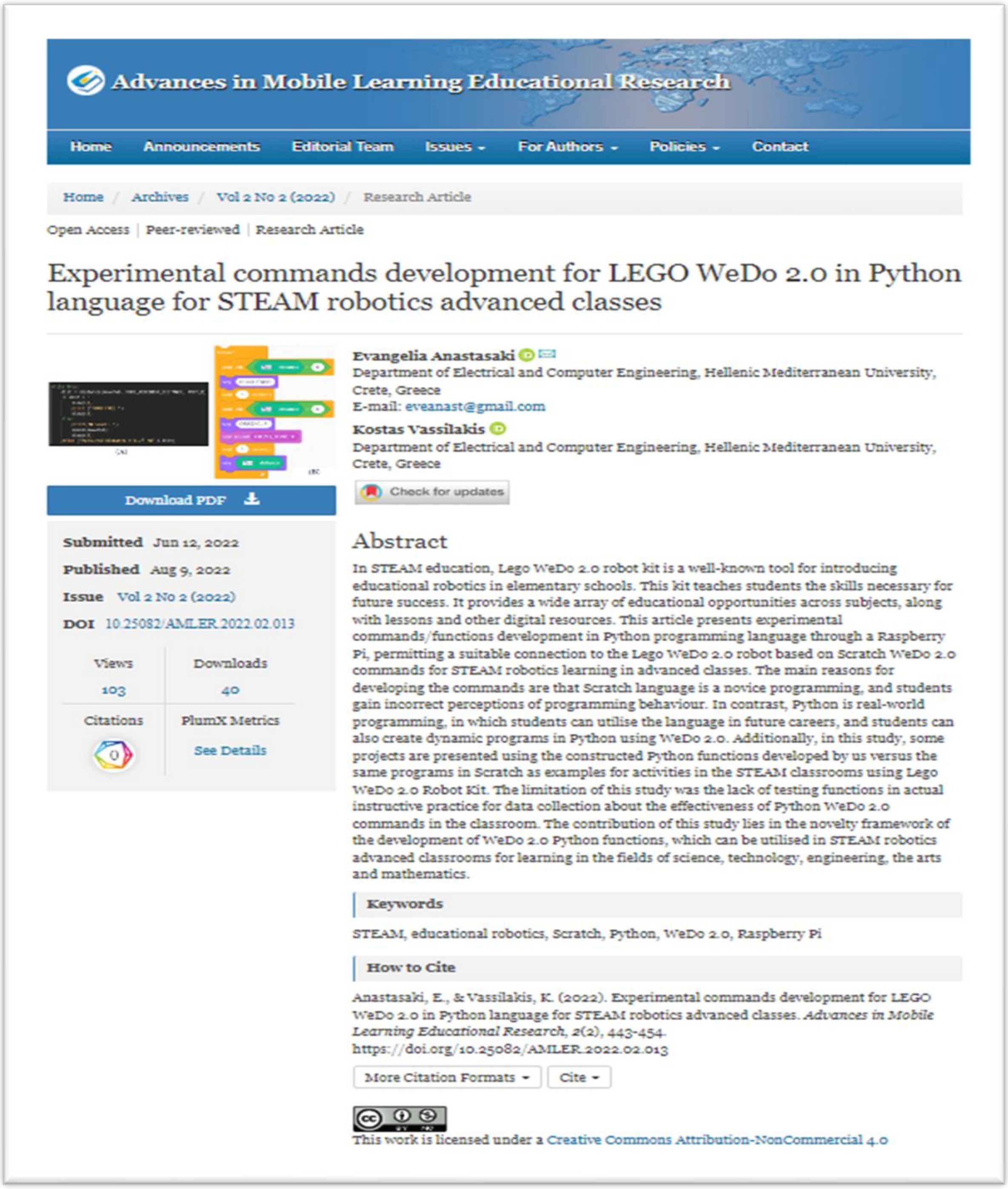
Wikipedia. (n.d.). Python (programming language) - Wikipedia. Retrieved 13 March 2022, from https://en.wikipedia.org/wiki/Python_(programming_language)

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1881), 3717–3725. Retrieved 10 July 2022 from https://doi.org/10.1098/RSTA.2008.0118

Zaher, A. A., & Hussain, G. A. (2020). STEAM-based active learning approach to selected topics in electrical/computer engineering. *IEEE Global Engineering Education Conference, EDUCON*, 2020-April, 1752–1757. Retrieved from https://doi.org/10.1109/EDUCON45650.2020.9125367

Appendix A - Published Journal Article

DOI: *https://doi.org/10.25082/AMLER.2022.02.013*



Open Access | Peer-reviewed | Research Article

Experimental commands development for LEGO WeDo 2.0 in Python language for STEAM robotics advanced classes

Evangelia Anastasaki
Department of Electrical and Computer Engineering, Hellenic Mediterranean University, Crete, Greece
E-mail: eveanast@gmail.com
Kostas Vassilakis
Department of Electrical and Computer Engineering, Hellenic Mediterranean University, Crete, Greece

Submitted Jun 12, 2022
Published Aug 9, 2022
Issue Vol 2 No 2 (2022)
DOI 10.25082/AMLER.2022.02.013

Views	Downloads
103	40
Citations	PlumX Metrics
0	See Details

Abstract

In STEAM education, Lego WeDo 2.0 robot kit is a well-known tool for introducing educational robotics in elementary schools. This kit teaches students the skills necessary for future success. It provides a wide array of educational opportunities across subjects, along with lessons and other digital resources. This article presents experimental commands/functions development in Python programming language through a Raspberry Pi, permitting a suitable connection to the Lego WeDo 2.0 robot based on Scratch WeDo 2.0 commands for STEAM robotics learning in advanced classes. The main reasons for developing the commands are that Scratch language is a novice programming, and students gain incorrect perceptions of programming behaviour. In contrast, Python is real-world programming, in which students can utilise the language in future careers, and students can also create dynamic programs in Python using WeDo 2.0. Additionally, in this study, some projects are presented using the constructed Python functions developed by us versus the same programs in Scratch as examples for activities in the STEAM classrooms using Lego WeDo 2.0 Robot Kit. The limitation of this study was the lack of testing functions in actual instructive practice for data collection about the effectiveness of Python WeDo 2.0 commands in the classroom. The contribution of this study lies in the novelty framework of the development of WeDo 2.0 Python functions, which can be utilised in STEAM robotics advanced classrooms for learning in the fields of science, technology, engineering, the arts and mathematics.

Keywords

STEAM, educational robotics, Scratch, Python, WeDo 2.0, Raspberry Pi

How to Cite

Anastasaki, E., & Vassilakis, K. (2022). Experimental commands development for LEGO WeDo 2.0 in Python language for STEAM robotics advanced classes. *Advances in Mobile Learning Educational Research*, *2*(2), 443-454. https://doi.org/10.25082/AMLER.2022.02.013

Appendix B - Whole Code in Python

```python
from gattlib import GATTRequester, GATTResponse
from time import sleep
import time
from binascii import hexlify
import serial,sys,glob,select
import struct

movehub = GATTRequester("04:EE:03:16:ED:1D",True,"hci0")

service_uuid = '00004f0e-1212-efde-1523-785feabcd123'
characteristic_uuid = '00001565-1212-efde-1523-785feabcd123'


BUTTON_PRESSED = '\x01'

# Ports

PORT_A = 0x37
PORT_B = 0x38

SensorValue_hnd  = 0x28
tilt_data=[0,0,0,0,0,0]

TiltCmdMode1 = '\x02\x02\x01\x22\x01\x01\x00\x00\x00\x02\x01'

# Motors:

MOTOR_A = bytes([0x37])
MOTOR_B = bytes([0x38])

# WeDo 2.0 Tilt Sensor modes:
# 0 = angle
# 1 = tilt
# 2 = crash
# 3 = some variation of angle
# modes 0,1,2 like WeDo 2.0 modes

MODE_WEDODIST_DISTANCE =
b'\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02'

# WeDo 2.0 Distance Sensor modes:
# 0 = distance 00..0A
# 1 = increments when blocked
# 2 = distance and luminosity ?
# all continuous reading
# just using mode 0=distance

MODE_WEDODIST_1 = b'\x01'
MODE_WEDODIST_2 = b'\x02'

MOTION_SENSOR = b'\x02\x01\x01\x23\x00\x00\x00\x10\x00\x00\x00\x10'

# Get a distance reading
motion_sensor_value = 0x23
InputCommand_hnd = 0x3a
OutputCommand_hnd  = 0x3d
device = object

RGBAbsoluteMode_cmd = "\x01\x02\x06\x17\x01\x01\x00\x00\x00\x02\x01"
```

```
RGBAbsoluteOutput_cmd = "\x06\x04\x03"

MODE_WEDODIST_DISTANCE = 
b'\x02\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02'


#sound function

def sound(self):
    HANDLE = 0x3d
    PIEZO_TONE = "\x05\x02\x04\xB8\x01\xF4\x01"
    movehub.write_by_handle(HANDLE, PIEZO_TONE)

#colorRGB Function

def colors(self):
    colors = 
['\x00','\x01','\x02','\x03','\x04','\x05','\x06','\x07','\x08','\x09','\x0A']
    for color in colors:
        movehub.write_by_handle(HANDLE, "\x06\x04\x01" + color)
        sleep(1)

#motor turn right

def motor_that_way(self):
    HANDLE = 0x3d
    TURN_RIGHT = "\x01\x01\x01\x9C"
    movehub.write_by_handle(HANDLE, TURN_RIGHT)

#motor turn left

def motor_this_way(self):
    HANDLE = 0x3d
    TURN_LEFT  = "\x01\x01\x01\x64"
    movehub.write_by_handle(HANDLE, TURN_LEFT)

#motor Off

def motor_off(self):
    HANDLE = 0x3d
    TURN_STOP  = "\x01\x01\x01\x00"
    movehub.write_by_handle(HANDLE, TURN_STOP)

# motor on

def motor_on(self):
    HANDLE = 0x3d
    TURN_ON = "\x01\x01\x01\x9c"
    movehub.write_by_handle(HANDLE, TURN_ON)
    sleep(0.2)

# distance sensor function

def distance(self, MODE_WEDODIST_DISTANCE, port):
    #print("connecting...")
    sleep(2)

    dev  = "/dev/ttyACM*"
    scan = glob.glob(dev)
    rate = "115200"
    STEP = 0.09  # emyric 0.003..0.1
    SYNC_TIME = 0.75 # time for the break command
    SYNC_ZEROS = 2000 # empyric
    MSG_RETRY = b'\xC0\x00\x3F'
    BYTE_START = 216  # D8

    if (len(scan) == 0):
        dev  = '/dev/ttyUSB*'
        scan = glob.glob(dev)
        if (len(scan) == 0):
            print ("Unable to find any ports scanning for /dev/[ttyACM*|ttyUSB*]" + dev )
            sys.exit()

    serport = scan[0]

    if (len(sys.argv) > 1):
        l = len(sys.argv) - 1
        while(l > 0):
            if (sys.argv[l][0] == '/'):
                serport = sys.argv[l]
            else:
                rate = sys.argv[l]
            l = l - 1

    ser = serial.Serial(port=serport,baudrate=rate,parity=serial.PARITY_NONE,stopbits=serial.STOPBITS_ONE,bytesize=serial.EIGHTBITS,timeout=1)
    print("connected to: " + ser.portstr)
    if ser.isOpen():
        #print("Starting")
        start_time = time.time()
    sensor = MODE_WEDODIST_DISTANCE
    while True:
        try:
            sensor = MODE_WEDODIST_DISTANCE
            movehub.write_by_handle(motion_sensor_value)[0]
            s = ser.readline(sensor)
            if s:
                for x in s:
                    print ("%s") % (x.encode('hex'))
                    print(s)
        except KeyboardInterrupt:
            sys.exit()
        except:
            pass
            movehub.write_by_handle(HANDLE,MODE_WEDODIST_DISTANCE)
            time.sleep(0.00001)
            StartTime = time.time()
            StopTime = time.time()
            # save StartTime
            while InputCommand_hnd == 0:
                StartTime = time.time()
            # save time of arrival
            while InputCommand_hnd == 1:
                StopTime = time.time()
            # time difference between start and arrival
            TimeElapsed = StopTime - StartTime
            # multiply with the sensor speed (34300 cm/s)
            # and divide by 2, because there and back
            distance = ((TimeElapsed * 34300)/2)*100
            distance = round(distance)
            return distance
    ser.close()

#tilt sensor function

def tilt(self, MODE_WEDOTILT_TILT, port):
    SensorValue_hnd = 0x28
    InputCommand_hnd = 0x3a
    TiltCmdMode1 = '\x02\x02\x01\x22\x01\x01\x00\x00\x00\x02\x01'
    movehub.read_by_handle(SensorValue_hnd)[0]
    movehub.write_by_handle(InputCommand_hnd, TiltCmdMode1)
```