Introducing Textual and Visual Based Programming to First Time Users

JULIA DANG
KEJSI GJORDENI
Abstract

The objective of this thesis is to introduce textual and visual based programming to first time users and evaluate how they are perceived, by performing literature research and conducting field studies at Grimstaskolan. By dividing a class of students in two test groups, each were introduced to textual language Python and visual language Scratch respectively. The initial response showed visual based programming as a more fun and motivating method of programming for a continued interest in learning. However, it was subsequently noted that visual programming itself was very distracting from the actual assignments. Despite Python being considered less motivating, the actual program was not a source of distraction in the aspect as Scratch was. What can be concluded is that Scratch is a good introductory language to promote systematic reasoning. It may also motivate children more to pursue programming, to then later have the ability to visualize the processes in a text based language such as Python. There are many different methods of practical implementation of computational programming, as of which cross-curricular programming and having it as a separate course are notably the two main possibilities, both of which are arguably preferable. As a recommendation that can be concluded for The National Agency for Education (Skolverket), a combination of multiple methods could be more appropriate, but should be more extensively studied.
# Contents

1 Introduction .......................... 1
   1.1 Objectives ......................... 1
      1.1.1 Research Questions .............. 2
      1.1.2 Limitations .................... 2

2 Background .......................... 3
   2.1 Theoretical Frameworks .............. 5
      2.1.1 Educational theories .......... 5
      2.1.2 Programming languages and environments .... 7

3 Related work ......................... 13

4 Methodology .......................... 14
   4.1 Literature study .................... 14
   4.2 Field study ........................ 14
      4.2.1 User study .................... 14
      4.2.2 Questionnaire ................ 15
      4.2.3 Semi structured interviews .... 15

5 Results ............................. 16
   5.1 Literature study .................... 16
      5.1.1 Programming languages and environments ...... 16
   5.2 Field study ........................ 17
      5.2.1 User study .................... 17
      5.2.2 Questionnaire ................ 18
      5.2.3 Semi structured interviews .... 20

6 Discussion ........................... 22
   6.1 Observation ....................... 22
   6.2 User study ........................ 23
   6.3 Questionnaire .................... 24
   6.4 Semi structured interviews ....... 25

7 Conclusion .......................... 26

References ............................ 27

Appendix ............................... 30
1 Introduction

Simultaneously as technology has developed in a rapid pace, the importance of teaching Computer Science and the requirement of programming skills has increased, in order to fill the gap in the labor market. The need of individuals with knowledge in several programming languages has expanded since numerous traditional jobs have undergone a digital transformation. In order to reach a higher rate of future graduates with programming skills, it is vital to stimulate the students’ interest at an early age. Being introduced to coding as a child in the same natural approach as one is taught reading or the multiplication table would lead to multiple positive outcomes. If the number of children having basic knowledge in programming would increase then so would the possible employees in developer-positions. Concurrently the technical field would become more equal both in terms of gender equality but also in the aspect of diversity. Seeing that only 14% of the software developers with an experience exceeding 20 years were women in 2015[8], efforts have to be made in order to provide our children with a more equal working environment in 20 years from now.

Derived from the importance programming has on today’s technology, this report aims to investigate how different programming languages are received by first time users and how programming can be implemented in a Swedish school based on current resources.

1.1 Objectives

The objective of this study is to compare text based programming languages and visual based programming languages by having parameters as user-friendliness and structure in consideration. More precisely, the languages being compared are Python, which is a text based programming language, and Scratch, which is a visual based language. Additionally, the thesis intends to investigate how Grimstaskolan in Vällingby could implement programming in their curriculum based on their existing conditions and with respect to the theory of learning.

User-friendliness has been examined through a practical research by teaching first time users how to program using the two stated languages and thereafter conduct semi-structured interviews and questionnaires. The structure of the programming languages was analyzed and compared through literature studies. In order to research how Grimstaskolan may implement programming in their curriculum, semi-structured interviews with teachers and principal have been conducted and also literature studies regarding pedagogy.
1.1.1 Research Questions

Based on the objectives of this study, the following questions have been specified:

- What are the differences between the textual programming language Python and the visual programming language Scratch?
- How does the student’s learning experience differ depending on if Python or Scratch is being taught as a first programming language?
- How should the investigated school implement the programming subject in their curriculum?

1.1.2 Limitations

The examination of user-friendliness has only been conducted on one school and in a defined group of 24 students, and the chosen students from 6th grade were aged between 12 and 13. Furthermore the programming languages used were only Python and Scratch, to avoid having the children focus on too many languages and also making the comparison of the results more distinct.
2 Background

In recent years many have acknowledged digital technology as a new fundament of knowledge, and teaching programming skills at early ages is considered a long-term solution to the qualification gap between the number of technology jobs\[4\]. Several first world governments have discussed the importance of mandating computer programming in primary and secondary schools, but only a few countries such as the UK and Finland have as of today implemented a new national curriculum which incorporates coding into the traditional schooling.

The United Kingdom

A new national curriculum was introduced in the United Kingdom in September 2013, making its largest upheaval in 14 years. Merely a year later in 2014, the UK replaced the subject Information and Communications Technology (ICT) with Computing as a mandatory subject which includes coding lessons for children starting from age five\[3\]. The only prominent challenge with such a drastic curriculum change is to make sure teachers are adequately prepared. In certain instances, tech-savvy pupils have taught the coding lessons such as at St Thomas the Apostle College. A YouGov study done following the new programmes of study in the UK measured that out of 1009 polled teacher, only 40% stated they had sufficient knowledge in order to teach computer technology. This study reflected criticism that there had not been sufficient training for teachers unfamiliar with this technology. However, there have been in fact large activities\[4\] trying to prepare teachers for this, consisting of fundings of £1.1m by the government to develop a programme for new computing teachers, and BCS’s two Computing in the National Curriculum guides\[11\]. Investments from Google and Microsoft and initiatives from Codeacademy to develop its educational site have also been made\[2\].

Finland

The national curriculum in Finland is renewed every decade for primary and secondary education. A new curriculum widening the competence areas of ICT became effective in the school year 2016-2017, but had been finalized in 2014 in order for municipalities and schools to have over two years to prepare. Finland have interpreted programming as a method to achieve computational and logical thinking and to develop problem solving. Therefore it has been a mandatory implementation in the curriculum but rather as a cross curricular theme instead of an independent subject, starting from first grade. The Finnish approach to learning coding is to position it as a new learning skill to complement reading, writing, drawing and calculating\[13\].
Sweden

Recent studies have proved Swedish schools to be falling behind on IT development, despite being one of the leading countries in digitization today. Programming in Sweden is only mandatory if students apply for the technical programme in secondary education. After the government constituted a secondary school reform in 2011, which entailed prioritizing subjects depending on programme field, programming was not considered a necessary subject for the majority. By 2015, the National Agency for Education (Skolverket) has been assigned to establish an entirely new digitization strategy for the Swedish curriculum, where programming is prioritized. With inspiration from neighbouring countries, the objective is to raise awareness and understanding of the technological fundamentals of the digitized society of today, and also view programming as a method of problem solving. Propositions have motivated to directly implement programming into other subjects to be utilized as a tool for learning. The vision is to realize the proposition in year 2022, by improving competence development, acquiring digital methods and tools for learning and contributions to employ the potential of digitization in education and administration.
2.1 Theoretical Frameworks

2.1.1 Educational theories

Educational theories are conceptual frameworks that examine how knowledge is absorbed, processed and retained during learning. How knowledge is retained is mainly influenced by prior experiences and cognitive-, emotional- and environmental impacts\cite{15}. However there is not a universally accepted definition of learning that is shared by theorists, researchers and practitioners\cite{26}. Three established criteria of learning are:

- Learning involves change
- Learning endures over time
- Learning occurs through experience

A general definition that captures the central criteria of learning is:

“Learning is an enduring change in behavior, or in the capacity to behave in a given fashion, which results from practice or other forms of experience”

Learning is inferential, which implies that it can not be observed directly but the only way to know if students have learned is by assessing learning’s outcomes and products. The different methods of assessing learning are e.g; direct observations, written responses, oral responses and self-reports\cite{23}. The three main educational theories are behaviourism, cognitive information processing (cognitivism) and constructivism. By considering the theories when conducting this study, a broader perspective of how knowledge is optimally acquired will be attained. The theories will underlie for the analytic discussion succeeding the field study.

Behaviorism

Behaviorists believe that changes in behaviors or new behaviours are achieved through associations between stimuli and responses. It was believed that only observable, recordable and measurable behaviour was of any real value. The aim with behaviorist pedagogy is to modify and promote the observed behaviours. Learning is therefore a behavior that displays acquisition of knowledge. If the behavior is not desirable it should be discouraged and otherwise promoted. This can be done by contracts, consequences and behaviour modification\cite{29}.

Cognitivism

During the 1950’s educational theory started to shift from behaviourism to more cognitive sciences. Observable behaviour became less important and the focus was on more complex cognitive processes, such as thinking, information processing and problem solving\cite{5}. Learning in cognitivism is associated with small changes between states of knowledge rather than with changes in the student’s
response. It is more important to study the learning process by examining what the learner knows and how the learner acquires it, and not with what the learner does[9].

**Constructivism**

Both behaviorism and cognitivism have underlying philosophical assumptions that are objectivistic, this means that the goal with instructions is to map the structure of the world onto the learner. This concept has been discussed and theorists are now adopting a more constructivist approach to learning[18]. The idea of constructivism can be stated as “knowledge is a function of how the individual creates meaning from his or her own experiences”. Constructive psychologists believe that the mind filters the world’s input in order to produce its own reality. Learning is acquired by creating meaning from experience[1].

**Multimedia Learning**

Also known as e-learning theory, multimedia learning entails text-based and illustrative learning, animated and narrated computer lessons, and actual real life verbal and graphical presentation slides. The main principle of multimedia learning relies in cognitive theory which affirms that deep learning is improved by a combination of images and words rather than from words alone. It is additionally required for an effective multimedia learning, that images and messages are designed in alignment with how the human mind works. Simply attaching an image to words is not a productive method to achieve multimedia learning. Richard Mayer discusses the three main assumptions for cognitive theory of multimedia learning:

- Information is processed in two separate channels, auditory and visual, which is sometimes referred to as Dual-Coding theory
- Both channels have limited capacity
- Learning is a continual and active process of filtering, selecting, organizing, and integrating information with prior knowledge as a basis

This implies that humans are only capable of processing a finite amount of information in each channel at a time, and produces mental images of the incoming information. It is therefore fundamental with multimedia learning to reflect on the choice of words presented in text or narration organized together with coherent images to promote and foster learning[14].
2.1.2 Programming languages and environments

Textual Programming

The conventional form of programming languages has purely textual syntax. The first programmers using electronic computers programmed in machine language using sequences of bits. As the programs written became larger and larger it was necessary for the notation to develop, and assembly languages were invented. However as competing design started developing programmers required a machine-independent language, which led to the first high-level programming language being invented. There are numerous high-level programming languages today and new ones are being developed continuously[24]. Every single programming language has been developed for a different purpose and its basic design is influenced by several factors, such as; readability, writability, reliability and cost[25].

One of the challenges of textual programming is describing the language in a manner that is precise and making it comprehensible for all of its users. In order to accomplish this, every language consists of syntax and semantics, two features that are closely related and should be specified without ambiguity. Its syntax is the form of its expressions, program units and statements. The semantics of a programming language is the meaning of the expressions, program units and statements. In programming languages that are well-designed, semantics follow directly from syntax[25].

Python

Python is an open source, object-oriented, high-level programming language, making it widely distributable and free to use. It was developed by Guido van Rossum in 1991, and is commercially used by large companies such as Google. It is an interpreted language, meaning most programs are executed immediately without compiling into machine-language. The program is instead translated immediately to a bytecode form of a process virtual machine language. Python is designed with an interface that underlines code readability and simplicity. This is notable in the use of whitespace indentation to demarcate code blocks, and also in an easy to learn syntax often using English keywords[6]. Python is a dynamically typed programming language. This is distinguished by having the majority of its language type checking during run-time performance rather than at compile-time, also called duck typing. In addition to being dynamically typed, Python is also strongly typed, which prevents operations that are not well-defined such as adding a number to a string[20].

Object-oriented programming design computer programs using objects, which are data structures consisting of datafields and methods. Python implements classes to user-define a framework for an object, with a set of attributes that characterizes any object of the class. An object is associated with its class through class and instance variables, and methods, which are retrieved through
Python programming requires data which must be specifically defined, which is referred to as data types. Since it is a dynamically typed language the compiler will understand what data type a variable has during run-time and therefore does not require typed variables, but will raise a run-time type error during a violation of the Python syntax rules. Python has a large variety of data types described in its library[19]. Some of the most common are str (string characters) and numerical data types, for example int (integer) and float (for reals). More data types are presented below, as explained by Professor David Joyner[10].

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Syntax example</th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>An immutable sequence of Unicode characters</td>
<td>&quot;String&quot;, &quot;hello!&quot;, '2017'</td>
</tr>
<tr>
<td>bytes</td>
<td>An immutable sequence of bytes</td>
<td>b' Some ASCII'</td>
</tr>
<tr>
<td>list</td>
<td>Mutable, can contain mixed types</td>
<td>[1.0, 'list', True]</td>
</tr>
<tr>
<td>tuple</td>
<td>Immutable, can contain mixed types</td>
<td>(-1.0, 'tuple', False)</td>
</tr>
<tr>
<td>set, frozendset</td>
<td>Unordered, contains no duplicates</td>
<td>set([1.2, 'xyz', True]), frozendset([4.0, 'abc', True])</td>
</tr>
<tr>
<td>dict</td>
<td>A mutable group of key and value pairs</td>
<td>{'key1': 1.0, 'key2': False}</td>
</tr>
<tr>
<td>int</td>
<td>An immutable fixed precision number of unlimited magnitude</td>
<td>21</td>
</tr>
<tr>
<td>float</td>
<td>An immutable floating point number (system-defined precision)</td>
<td>3.1415</td>
</tr>
<tr>
<td>complex</td>
<td>An immutable complex number with real and imaginary parts</td>
<td>-3 + 1.4j</td>
</tr>
<tr>
<td>bool</td>
<td>An immutable Boolean value</td>
<td>True, False</td>
</tr>
</tbody>
</table>
Some basic Python statements are:

- **Assignments** - this statement uses the equals sign `=` as its token. The assignment `x = 3` would imply giving the name `x` a reference to a distinct object of the numerical `int` type value 3.

- **Conditionals** - conditional statements are used to execute certain computations in a block of code to be evaluated as true or false. Keywords used are `if`, `elif` and `else`.

- **Iterations** - an iteration is a type of control flow using `for` or `while` loops to execute multiple iterations until or as long as a condition is satisfied. If not satisfied, the loop will be infinite.

Below follows an example of a program written in Python. The program takes as input a number and will then evaluate if it is an even or odd number and print out the answer.

```python
number = int(input("Please type a number: "))
if (number%2) == 0 :
    print("It is an even number.")
else :
    print("It is an odd number.")
```
Visual Programming

Programming can be considered difficult for beginners since it requires unfamiliar expressions of solutions. Previous studies have observed how some of its difficulties are attributed to the user interface of a programming system[16]. Through combining usability, human-computer interaction (HCI) knowledge and user studies of natural solutions of non-programmers solving programming problems, several visual programming systems have been developed. Visual programming languages allow users to develop software programs through series of visual graphics elements serving as inputs, functions or outputs, and thus eliminating text based software code. These have proven to perform significantly better for first-time users and especially for teaching children[7]. Visual programming usually provides operators that match the methods a non-programmer would use to solve algorithms. By implementing interactive animations and simulations, it is feasible to graphically illustrate the events of an algorithm to improve understanding.

Scratch

Scratch is a programming language developed by the Lifelong Kindergarten Group at the MIT Media Lab and is currently used in more than 150 countries and available in more than 40 different languages. The aim with the project is to help mainly children between the ages 8 to 16 to reach a more creative way of thinking, have a systematic reasoning and work collaboratively[31]. Easily put Scratch can be compared to Lego. The programming language is based on graphical code blocks where the blocks represent different programming commands. By putting together the blocks the user is able to create a program.

The Scratch programming language consists of four different kinds of blocks and Scratch does not allow the blocks to be attached in meaningless ways:

- **Command blocks** can be compared to statements in a text based programming language, when multiple command blocks are merged they create a sequence of commands called a stack
- **Function blocks** are as opposed to command blocks not joined in linear sequences, they are used as arguments to commands and build expressions by being nested together
- **Trigger blocks** have the purpose to connect events to the stacks that handle those events
- **Control structure blocks** are similar to command blocks, but with one or more nested command sequences[12]
Table 2: Scratch Block Types

| Command block - multiple blocks can be combined to create a stack |
| Function block - returns a value |
| Trigger block - when the triggering events occur it runs the commands below it |
| Control structure block - holds nested command sequences |

Just as any other programming language Scratch consists of data types, more precisely three different types. String, number and boolean are the only data types stored in variables, used in expressions and returned by built-in functions. The variables in Scratch can contain either strings or numbers and are therefore untyped and a variable can hold values of both data types. The reason is to simplify the process by not requiring the user to specify the type when the variable is created. Depending on the context Scratch automatically converts between numbers and strings.12

Table 3: Scratch Data Types

| Command block with a round parameter slot for a number parameter |
| Command block with a rectangular parameter slot for a string parameter |
| Control structure block with a hexagon parameter slot for a boolean parameter |

Wegner stated in 1987 that a language must support inheritance to be called object-oriented.34 Since Scratch neither has inheritance nor classes it is not an object-oriented language, however it is an object-based language. It consists of sprites, which are objects that encapsulate state and behaviour. Commands appear in different sprites and can only operate on the sprite they appear in, and every sprite has its own independent set of scripts. Since it is not possible for sprites to call each other’s scripts directly, Scratch uses a broadcast mechanism. This mechanism is one-to-many, loosely-coupled, asynchronous and supports synchronization and communication between sprites.

11
Another technique provided by the visual programming language Scratch is concurrency. This can be found in the sprites ability to execute several things simultaneously. Despite this, Scratch lacks the explicit concurrency control mechanisms that can be found in other languages. Instead of providing locks or monitors, Scratch intends to avoid race conditions by building concurrency control in its threading model by constraining where thread switches can occur. This does however not eliminate all concurrency problems[22].

Below follows an example of a program written in Scratch. The program takes as input a number and will then evaluate if it is an even or odd number and print out the answer.
3 Related work

Implementing computational programming in primary and secondary education is a relatively new initiative, therefore not many studies have been conducted on the specific matter, however they have produced qualitative and positive results. One particular study performed by Saéz-López et al. in 2015 in Spain analyzed the outcomes and approach of visual programming using Scratch with 107 primary school students between 5th to 6th grade in five different schools. Saéz-López integrated coding and visual blocks programming through a cross-curricular implementation in sciences and arts. The learning processes were studied over two years and the experimental group was analyzed through questionnaires and structured observation. The research showed positive results in favor of visual programming and a recommendation to implement a visual programming language in elementary schools was concluded\[30\].

Another relevant research published by Carnegie Mellon University studied what solutions non-programmers would have to programming problems in terms of language used and solving structure. The reason for examining this problem was the acknowledgement of the difficulties of programming since the required solutions are expressed in unfamiliar or unnatural ways for a beginner. One of the methodologies was to examine children’s solutions to tasks necessary for a computer game. This information was later intended to be used as a design technique for future languages that could match how a beginner would solve algorithms. By employing the methodology of the related work as a framework and inspiration, a better understanding of how beginner’s understanding and natural tendencies for problem solving can be applied to the following user-study\[17\].

A research paper published in American Society for Engineering Education by Yoder and Black in 2006 presents the results of using both MATLAB (a text-based language) and LabVIEW (a visual programming language) in teaching discrete-time signal processing (DSP). By conducting numerous mini-projects and educating the students in the different languages the authors did not aim to determine whether or not one of the languages is superior to the other, but instead investigate the tradeoffs in choosing between the two types of programming languages to teach DSP. Disregarding the fact that the objective of the research executed by Yoder and Black differs from the objectives of this report, lessons can be drawn from the methodology used by Yoder and Black. The students were given the same tasks both in MATLAB and in LabVIEW so that if there were any differences in the results they would most likely be a consequence of the change in language. Furthermore some of the students had prior knowledge in MATLAB, therefore two additional LabVIEW-labs were imposed. The research concluded that the students preferred using the visual language and the instructors found it easier to grade a program written in a visual language\[35\].
4 Methodology

4.1 Literature study

Information was gathered in order to benchmark how programming had been introduced to primary and secondary education in other countries, specifically Finland and the UK. The study focused on these countries since they are the few countries where programming has been successfully implemented in its school curriculum. The study also collected relevant theories to use for understanding the necessities of learning and pedagogy for children to later apply this for Swedish schools. This could additionally be intended to use as a framework of methods to consider for primary schools in Sweden.

Literature and previous research was primarily retrieved from acknowledged databases such as KTH Primo and Google Scholar. The pilot study sourced its information from known news publishers and articles from national organisations. The selection of articles, books and reports were reviewed through relevance for our research and also its credibility by verifying if its source was reliable.

4.2 Field study

On site field studies were conducted for collecting data about first time programmers by executing a user study, observations and interviews. The investigation took place at Grimstaskolan in Vällingby, where 21 users aged 12-13 years old were introduced to programming, and teachers and principals were interviewed. During the whole field study observations were made and noted.

4.2.1 User study

Teachers were given the task to randomly choose 24 students, with no respect to gender or other attributes. Because three of them were sick the given day, only 21 users participated in the study. Twelve laptops were provided by the IT-section of the school and students were split into two groups; twelve learned to code using Scratch and nine of them using Python. Two separate two-hour long classes were designed similarly, with the aim to only differ in programming language being used. An introduction to the language was given where the syntax and contexture was briefly explained. Then the students were given four tasks with different degree of difficulty, starting with the easier ones. The four tasks conducted in Python (Appendix 1) and the four ones in Scratch (Appendix 2) were the same. Help was provided during the study and it was made clear to the students that their skills were not being evaluated and that the aim was to introduce them to programming. Afterwards, a questionnaire was provided to all of the participants.
4.2.2 Questionnaire

The questionnaire included eight questions and were answered by all 21 participants of the user study. The intention was to retrieve a more structured and quantitative data gathering of the user study. Questions were designed to determine the students' prior knowledge of programming, their perception of the level of difficulty of the tasks and if they found an interest in programming.

4.2.3 Semi structured interviews

To understand and analyze the whole chain of the education system, semi-structured interviews were conducted at Grimstaskolan. The objective was to interview policy makers in order to gather a more qualitative understanding of their impression of programming, and what opportunities and difficulties it may have to implement. At Grimstaskolan, two teachers and a principal were semi-structurally interviewed and recorded. The interviews inquired about the past experiences and resources needed for introducing programming at their school. They were also asked about their personal thoughts and confidence of teaching programming and the optimal way to help teachers prepare for it, and how they would suggest it to be implemented in practice.
5 Results

5.1 Literature study

5.1.1 Programming languages and environments

This report aimed inter alia to investigate how visual- and textual programming, and in particular the programming languages Scratch and Python, differ and relate to each other. With respect to the fact that every type of programming and every programming language has been developed with a certain aspiration or to solve a specific problem, both of the reviewed languages serve their own purpose. Textual programming tends to be challenging to easily explain and to make accessible for all of its users. Visual programming has on the other hand limited syntax and semantics, which tends to be more apparent. Additionally, visual programming has been developed with respect to usability and human-computer interaction. The aim with Scratch was to create a language and platform that develops the creative thinking of children by using blocks that resemble Lego, and simultaneously let them work collaboratively by being able to share projects. Along these lines the creator of Python, Guido van Rossum, had the ambition to create a language that could provide code readability and simplicity by using whitespace indentation and an easy syntax.

Moreover Python is dynamically typed and strongly typed, which implies that numbers can be added to a string. This is a property that is also shared by Scratch and can be explained by both languages’ ambition to simplify the process by not requiring the user to specify the type when the variable is created. Depending on the context, Scratch is able to automatically convert between numbers and strings. Both languages consist of data types and some of the types can be encountered in both Scratch and Python. Scratch consists of three different data types; strings, numbers and booleans. These are also data types in Python, although numbers can be of different types, such as int or float. The textual language Python has in addition, a numerous of other data types, e.g; list, dict and tuple. Although the two languages do not share syntax or structure, similarities can be found and parallels can be drawn. For instance Scratch consists of four different kind of blocks (command-, function-, trigger and control blocks). These blocks can be compared to the Python statements such as; assignments, conditionals and iterations. Python is defined as an object-oriented language, since it supports encapsulation, inheritance and polymorphism. Scratch on the other hand does not support inheritance and does therefore not satisfy the criterion to be classed as an object-oriented language. Scratch is however classified as an object-based language.
Table 4: Similarities and differences between Python and Scratch

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Python</th>
<th>Scratch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Easy and comprehensive programming language</td>
<td>Easy and comprehensive programming language</td>
</tr>
<tr>
<td>Data types</td>
<td>At least ten different data types</td>
<td>Three different data types</td>
</tr>
<tr>
<td>Programming language model</td>
<td>Object-oriented</td>
<td>Object-based</td>
</tr>
</tbody>
</table>

5.2 Field study

5.2.1 User study

21 students regardless of gender participated in the study, whereas ten of them were 13 years old and eleven of them twelve years old. More precisely two of the students programming in the language Python were 13 and seven of them were twelve years old. Of the ones programming in the language Scratch, eight of them were 13 and the other four ones were 12 years old. When asked about their previous experiences regarding programming, the 81% answered that they had none or very little experience. The rest, 19%, answered that their experience corresponded to a three on a scale from zero to five. When analyzing the result of the two languages separately it can be noted that 92% of the ones using Scratch had none or very little experience and 8% chose a three on the grading scale. 78% of the students programming in Python had none or very little experience and 22% chose a three. Some of the programming languages and programs mentioned when asked to specify their experience were; Hour of Code, Minecraft, Scratch and Swedish. The two distinct test groups were not aware of the respective group’s assigned programming language.

Observations were made during the whole field study, both about specific users and regarding group dynamics. Despite the fact that the students were chosen randomly according to their teacher, it was noticeable that they were divided into different groups of friends. The different groups chose to sit together, which resulted in various dynamics in the room. Some conveyed the impression of being focused, eager to learn and interested in the subject being taught, others seemed easily distracted and unenthusiastic. When given the different tasks a couple of the students gave the impression of being very active by asking several questions when needing help. Others were less active and hesitated to ask for help when they found a given task difficult. A few key points that had not been considered beforehand were the facts that the provided computers had access to Internet and that Scratch provides other features, such as designing your own mascot. The students did on several occasions deviate from the given tasks and spent the time on the two above mentioned activities.
5.2.2 Questionnaire

One of the questions asked in the questionnaire was to estimate the difficulty level of the given tasks by choosing a number from one to five, where one equals very easy and five very difficult. The result is presented in Figure 1. None of the students programming in Python perceived the difficulty level of the tasks to be easy or very easy, however 8.3% of the ones programming in Scratch did. 25% of the Scratch-users thought the tasks were difficult or very difficult and 44.4% of the Python-users thought similarly. Furthermore the majority of both user-groups chose to classify the tasks on an average level regarding difficulty.

When being asked to specify which parts of the lesson or the tasks were especially difficult or easy the answers differed between the two separate groups. A couple of the students programming in Python answered that everything was difficult without specifying which parts were perceived that way. Some answered that it was difficult to begin solving the tasks. Unlike the Python-users, the students programming in Scratch chose to answer more thoroughly. Several of them thought that the first two tasks were quite easy, however the difficulty level increased a lot on the last two tasks. Some thought that the instructions were difficult to comprehend and that it was not easy to know which of the many blocks provided by Scratch was the right one to use for specific tasks.

Question 1

![Figure 1: Difficulty level of the given tasks (1 = very easy, 5 = very difficult)](image-url)
The questionnaire also inquired if the test subjects would consider continue pro-
gramming in Scratch and Python respectively. As demonstrated in Figure 2, 83.3% would program in Scratch again, compared to the 44.4% that would con-
tinue programming with Python. None of the Scratch users responded that they
did not want to attempt using it again, whilst 22.2% of the Python testers re-
marked that they would not. Additionally, 33.3% of the students trying Python
had no opinion regarding the matter, whereas 16.7% of the Scratch users an-
swered the same.

![Question 2](image)

The overall response towards both programming languages considered Scratch
slightly more fun because of the combined allotment 58.4% (score 4 and 5). Scratch had a wider distribution in responses. Less than half, 44.4%, of the
Python users considered it fun after the day’s task. Score 3 is evaluated as
neither fun nor not fun, in which Python retrieved 44.4% and Scratch 16.7%. Both programming languages received one person each that did not consider
programming fun at all.
5.2.3 Semi structured interviews

Two teachers and one principal were interviewed at Grimstaskolan. One of the teachers had had experience in primary school programming using Scratch, and the others did not have any personal experience with programming. They were all very positive towards introducing computational programming at primary and secondary schools. The principal had not retrieved any information regarding a new curriculum, only that programming may be included in mathematics. Despite the lack of instruction, it was not considered a major difficulty to implement considering the current resources of the school and also the many possibilities to acquire more beneficial resources. The school already had a few teachers with either interest or training in programming, enabling them to institute a “student’s choice” course where programming was optional as an ungraded class once a week. 7th graders had also programmed in Scratch during a period of 20 weeks, where the teacher responsible for teaching had not had any knowledge prior to the course, but used the Scratch tutorials simultaneously as teaching the students interactively. That teacher still felt confident in being able to teach, and was positive with having the students with even more knowledge being able to help their classmates in the class. The teacher with no prior knowledge in programming suggested that internal training for teachers would be helpful to prepare the teachers and also for example, having the Swedish National Agency for Education provide teachers a training course in computational programming. As for materialistic concerns for the future, the school had plans on investing in Chromebooks for the students and replace the current iPads they had that were not very compatible for certain softwares and external applications not approved by the municipality of Stockholm.
The interviewees were also briefed about the current benchmarks of programming implementation in the UK and Finland, where they were asked about their opinion of how programming should be realized in schools. The principal and the teacher with prior knowledge of programming were in favour of having programming as a cross-curricular introduction, where they could be taught together with technological sciences and mathematics. They reckoned it would be potentially easier to implement a new and foreign subject together with a familiar one, with having the student’s interest and understanding of a subject in consideration. Meanwhile, the other teacher argued that having it as an individual course would give the learning more substance and a wider focus to the concept of computational programming. By teaching it cross-curricular, he believed it could probably be neglected in a course and a developed understanding would take longer time.
6 Discussion

6.1 Observation

The given results show that the students perceived the given tasks to be more difficult when programming in Python, moreover a lower rate of students would like to continue programming in Python compared to Scratch. Seeing that the ones programming in Scratch found programming more fun compared to the ones programming in Python, one could argue that first time programmers comprehend the given tasks easier when programming in Scratch and this leads to them perceiving programming as something more enjoyable than if they would have started programming in Python. However the students trying to program in Scratch were not the same students who programmed in Python and every individual perceives given problem in their unique way. The study could have been made in such manner that every student had to try both Scratch and Python, but then one programming language would have to be introduced before the other one and the experience would not be of a first time user.

The teachers were asked to randomly pick 24 students without respect to anything but the lack of programming experience. Three students being sick led to the two groups being unevenly divided and only nine students programmed in Python, while twelve programmed in Scratch. At the same time the majority of the students participating in the Scratch-lesson were 13 years old and the majority of the ones in the Python-lesson were twelve years old. Even if all of the students attend the same class and one year is not a noticeable difference these sources of error has to be taken into consideration when examining the results. Additional factors that did affect the conducted study was that the chosen students were different groups of friends and they sat together. This caused them to be distracted and get involved in discussion that did not concern the given tasks. The fact that the computers had access to Internet caused another source of distraction and if the study would be conducted at another time the students would be positioned at given places and the computers would not be connected to the Internet.

Only having two hours each with the two groups made it difficult to give full attention to every student and for everyone to learn and perform their very best. In addition to the limited time, technical difficulties such as not having login credentials to every student made it time constraining. Knowing that learning is inferential and can not be observed directly by the teacher, it is difficult to say by this study if Scratch or Python gives the best result when teaching first time programmers. To know if the students have learned anything one has to assess the outcomes and products of learning. This could not be done within two hours. Despite this, the result did show how the first impression of programming differed depending if the student programmed in Python or in Scratch.
An overall problematic matter was having the test subjects not focusing on doing the actual tasks. It would have been preferable to have a familiar supervisor for the students in the classroom during the test, in order to take the study more seriously. Grimstaskolan is in fact known to be a problematic school in an exposed neighbourhood, it would therefore have been appropriate to conduct the same study in a less unruly school to test the same study in other circumstances. To further improve the study, the assignment descriptions should have implemented the cognitive theory of multimedia learning. It was difficult to know prior to the field study if the explanations made were sufficiently pedagogical. Also, due to technical difficulties no images of the presentation lecture could have been made to visually portray the computational programming methods, which automatically obstructs the optimal cognitive learning according to theory.

As a behaviourist, the focus should have been on the students’ behaviour and if it showed signs of assessing knowledge it should be rewarded and if not the student should be disfavoured in some way. The choice was made to not create a gap between the students and create an environment where no one was being judged. Instead the focus was on a more cognitive way of pedagogy where information processing and problem solving is more important. A lot of time was spent on explaining the tasks and letting the students process the information in their own way, thereafter the students did think-alouds where they discussed how they would try to solve the problem.

6.2 User study

It was notable that Scratch seemed more interesting with the graphical images and colorful blocks. However, that was also a cause of distraction for the test subjects. The children deviated from the actual assignment tasks when they were unsupervised and started to customize their Scratch characters and backgrounds instead. A difficulty for first time users was locating the blocks, which only needed some time and supervision to realize. Most probably due to the time restriction, the students did not fully understand the different types of blocks (command, function, trigger and control structure blocks) and only used the ones they had been introduced to. Since it was possible to reuse the building blocks for new functions, the solutions were facilitated for the following assignments.

Python required much more help to solve the designated tasks step by step. One of the main problems for the students was the strict syntax Python requires. It was apparent that the children had difficulties with spelling and also with English, as Python uses many built in keywords from the English vocabulary. This issue therefore caused run-time errors and problems with solving the tasks, when for example the variables were spelled inconsistently or because they could not understand the conditional statements. Since each task required a new program file, the students did not think of reusing their previous meth-
ods of solution for other assignments which also made it more complicated and tedious to start blank with a new solution. The majority of the test subjects were also not accustomed to using a computer, meaning they had issues finding basic keys such as ‘+’ or ‘%’.

Following this lack of knowledge, they did not know how to save a program file to their computers either. The school where the field study was conducted on has mainly focused on using iPads for their education, which could potentially account for the technical unskillfulness.

The students utilized IDLE to write, save and run their Python code. IDLE provides several features that simplifies the process, such as automatic indentation, text colour for text with special meaning and calltips. This was however not sufficient considering the knowledge of the students, so problems and difficulties occurred despite the use of IDLE. For further studies a consideration could be to instead of IDLE use help tools such as Python Tutor, which is a live programming mode that continually runs and visualizes one’s code as one types. That would evade and/or help with issues occurring while writing, saving and running the code and optimize the time during the learning sessions.

The user study was conducted during two sessions, two hours each, and comparing this to the amount of training and teaching the students get in other subjects it can be stated that it is not much time. A consequence of this was that the given tasks were constructed to not be complex and quite easy. A different approach could have been to make less tasks, but make them more complicated and time consuming. Working on a bigger task could have made the students more motivated and in the end lead to them learning more from the sessions. More complex tasks could also make the students find Python easier than Scratch, due to the fact that Python has a more logical and shorter solutions to for example function calling or recursive functions. Due to fact that there are many factors affecting the learning experience of the students it is crucial for the ones conducting future studies in this subject to consider alternative choices when it comes to length of sessions, number of tasks and the difficulty of the task.

6.3 Questionnaire

It was notable when examining the responses of the questionnaires that many of the students did not fully understand the questions nor bothered to answer the questions as thoroughly as possible. E.g., one of the students answered “Swedish” when asked in which programming language him/her had programmed before. A different and better approach might have been to ask the students directly about their experience and conduct interviews instead. This would have let them to speak more freely and not think out their answers beforehand. Furthermore some of the students chose to look at their neighbours and answer just like them, others did not want to confess how difficult the tasks actually were.
6.4 Semi structured interviews

The semi structured interviews depicted a very positive attitude towards introducing programming in schools, with several suggestions on how it could practically be implemented. In comparison, the benchmarks performed in the UK and Finland have portrayed a much wider reaction towards programming, and since the interviews conducted are not of a quantitative analysis, it is impossible to generalize as a reaction for all schools in Sweden. Grimstaskolan is also more progressive and on front edge, according to the principal, when it comes to primary school programming which promotes their positivity towards it. The principal of Grimstaskolan also stated that the school already had teachers with knowledge, but since programming is a broad term and have many forms, the principal may not be aware herself of which knowledge is required. Noted during the interviews was that some perspectives of programming could be pressing buttons on a robot, and not actual typed programming such as Python, and therefore misunderstanding may have arisen.

To further improve the interviews qualitatively, sending out the question beforehand would have given the interviewees more time to thoroughly reflect upon the questions. An alternative would also to have conducted the study with the children first, to then later interview the teachers, instead of doing them in parallel. The reason is that questions regarding the students that arose afterwards could have been answered by their supervisors.
7 Conclusion

Textual and visual based languages all serve their own purpose and the visual language Scratch and the textual language Python are both invented with the aim to make programming easier. While Scratch consists of different kind of blocks that put together create a program, Python uses different statements. Both languages are composed of numerous data types and whilst Scratch only provides three different, Python provides at least ten different ones. The languages share the characteristic of being dynamically typed and moreover Scratch is able to automatically convert between strings and numbers. It can be stated that despite the fact that Scratch and Python are different kind of programming languages, one a visual based language and the other a textual one, they do share similarities.

First time users perceived Scratch as more fun and based from the field study somewhat more willing to continue visual based programming. It was on the contrary more distracting having customizable graphical images, which diverted the test subjects from the actual assignment. Python users did not have a majority considering it fun, and the overall preciseness of syntax was difficult for them to comprehend. However, both user studies had a time constraint of two hours, making it impossible to draw a definite conclusion of which programming language is more appropriate than the other. What can be concluded is that Scratch is a good introductory language to promote systematic reasoning. It may also motivate children more to pursue programming, to then later have the ability to visualize the processes in a text based language such as Python.

There are different methods of practical implementation of computational programming, as of which cross-curricular programming and having it as a separate course are notably the two main possibilities, both of which are arguably preferable. To examine which of these methods is better to applicate to the Swedish school curriculum, a solution could be to extensively test both methods by dividing many schools into two separate test groups, and concurrently implement either of the methods to later evaluate the response and results. There are possibilities that a combination of multiple methods are more appropriate, so the study does not require having one concrete decision. The National Agency for Education (Skolverket) should measure the competence of teachers and provide training for current teachers, and also evaluate how practical resources like computers and software should be supplied appropriately.

For future studies and further improvements within the areas of learning and programming, it is reasonable to conduct a more extensive field study in several schools and also over a longer period of time. The field study should be even more considerate to children, and can therefore include a more pedagogical method to the assignments. Generally, the study requires more time to be fully explanatory. However, a perception of how first time users comprehend programming could be attained.
References


Appendix

Appendix 1 - Tasks in Python

1. Write a program that prints the string “Hello World”.

```python
# print("Hello World!")
x = "Hello"
y = "World"
print(x + " " + y)
```

2. Write a program that defines two variables x and y to any optional whole number, and print the sum of each variable.

```python
x = 1
y = 2
func = x + y
print(func)
```

3. Write a program that takes input a number. The program will then evaluate if it is an even or odd number and then prints the answer.

```python
nummer = int(input("\nVänligen skriv ett nummer: "))
if (nummer%2) == 0 :
    print("\nDet är ett jämnt nummer.\n")
else:
    print("\nDet är ett udda nummer.\n")
```
4. Write a program where a variable “number” is equal to zero. As long as the number is less than 100, the value 10 will be added for each function call say_hello(). Define a function say_hello() that prints “Good bye!” if the number is equal to 100, otherwise print “Hello!”.

def say_hello():
    if number == 100:
        print("Hejdå!")
    else:
        print("Hej!")

number = 0
while number < 100:
    number += 10
    say_hello()
Appendix 2 - Tasks in Scratch

1. Write a program that prints the string “Hello World”.

2. Write a program that defines two variables x and y to any optional whole number, and print the sum of each variable.
3. Write a program that takes input a number. The program will then evaluate if it is an even or odd number and then prints the answer.
4. Write a program where a variable “number” is equal to zero. As long as the number is less than 100, the value 10 will be added for each function call say_hello(). Define a function say_hello() that prints “Good bye!” if the number is equal to 100, otherwise print “Hello!”.