Universiteit Leiden
The Netherlands

Opleiding Informatica & Economie

Programming Misconceptions of Children from Ages 8 to 11

Tessa Krabbendam

1674641

Supervisors:
Felienne Hermans & Alaaeddin Swidan

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

03/07/2019

Abstract

This research is about programming misconception of children between the ages 8 and 11. In this research, we specifically look at the programming environment Scratch. We asked fifteen children what their answers and explanations behind those answers were on specific code sequences in Scratch. From their answers, we learn that the basic concepts of programming, such as variables, are not well understood by children. This has partly to do with not reading the blocks carefully. We also learn that children do not seem to see the differences between different blocks in Scratch. To make sure that these things are resolved in the future we advise teachers to start with the basics and make sure that these concepts are well understood before continuing on to more complex concepts. We encourage to use the colours and shapes of the blocks that are provided by Scratch.

Contents

Introduction

1.1 Problem

Currently, programming education is gaining popularity. In The Netherlands, we see that the primary schools are interested in teaching their students programming skills. One of the tools used for programming education in primary school is Scratch. Scratch is a visual programming environment using blocks to make scripts that let sprites do things.
The teaching of programming comes with difficulties. One of these difficulties is the children having programming misconceptions. This means that they understand a concept in the wrong way. Having programming misconceptions can be very frustrating while programming. For example, not being able to figure out why the program is not giving the expected output can be time-consuming and frustrating. By looking at the reasoning behind some programming misconceptions that children of ages 8 to 11 have, we want to find a way to resolve these misconceptions using learning materials and also investigate what is causing these misconceptions.

1.1.1 Research Question

Therefore the research question of this thesis is. "What is the origin of programming misconceptions of children from ages 8 to 11 and how can these be resolved?".
From the research question multiple sub-questions can be defined:

RQ1 What is the origin of programming misconceptions?

RQ2 How can the programming misconceptions be resolved?

RQ3 How can the programming misconceptions be prevented?

1.2 Thesis Overview

This thesis starts with background information and related work on programming education, programming misconceptions and Scratch. Then the thesis goes into the approach that is taken to do the research. Then the thesis will finish with the results and draw conclusions. The thesis is about programming misconception in Scratch of children between the ages 8 and 11. To get a good idea of the programming misconceptions they have we interview them. Based on the answers that the children give we write a trajectory which contains tips and tricks for teachers so that the misconceptions we found can be prevented in the future. This research will be supervised by Felienne Hermans and Alaaeddin Swidan from PERL-group at LIACS.

Background and Related Work

2.1 Programming Education

Programming is a relatively new subject when it comes to education. There are more and more examples of countries that are adding programming education to their primary school curriculum [1, 2]. One of the ways used to teaching children how to program is Scratch [3]. Teaching programming is not only about programming, but also about learning children that failing is okay and that we can learn from the mistakes [4]. In programming you will not have everything correct the first try, you need to keep trying to find the right or best solution to your problem. It also teaches the idea of computational thinking, which means being able to think about a problem and solving in such a way that is understandable for a computer and a human.

2.1.1 Scratch

Scratch [5] is a visual programming environment using a block-based programming language. This means that every block in the program represents a variable, an operator or a function. The developers came up with three design principles for Scratch. They wanted to develop something that was "more tinkerable, more meaningful and more social than other programming environments" [6].

In Figure 2.1, the user interface of Scratch is presented. This is the screen that shows when starting the program. All the available blocks can be found in the column on the left. This column is called 'the palette'. These blocks can be combined into code sequences, also known as scripts, an example is Figure 4.1. With these code sequences, you can program sprites or stages. Stages in Scratch are the background behind the sprites. These stages are able to change during the program. Sprites in Scratch can be compared with objects in textual programming languages.

Scratch also has a social environment that allows users to share their projects and get comments and feedback. Other users can also copy projects and alter them. This approach helps users learn from each other. Since its launch in 2007 Scratch has grown in the amount of users [7]. At the time of this research, Scratch has more than 41 million unique registered users. Most users are children and young adults with the biggest group being 12 years old.

The program Scratch can be accessed via two different ways. The program can be downloaded onto the computer and used offline. Scratch can also be used accessed via the internet, then there is no installation required. Both ways allow the user to upload their projects to their Scratch account.

Scratch has a seperate archive that can be used by teachers and educators to get content for their lessons. This archive is called 'The ScratchED Online Community Archive' [8]. It was developed in 2009 to offer teachers and educators a platform to support and help each other. Since 2019 it is no longer possible to join the platform but teachers and educators are able to use everything that has been put on the platform.

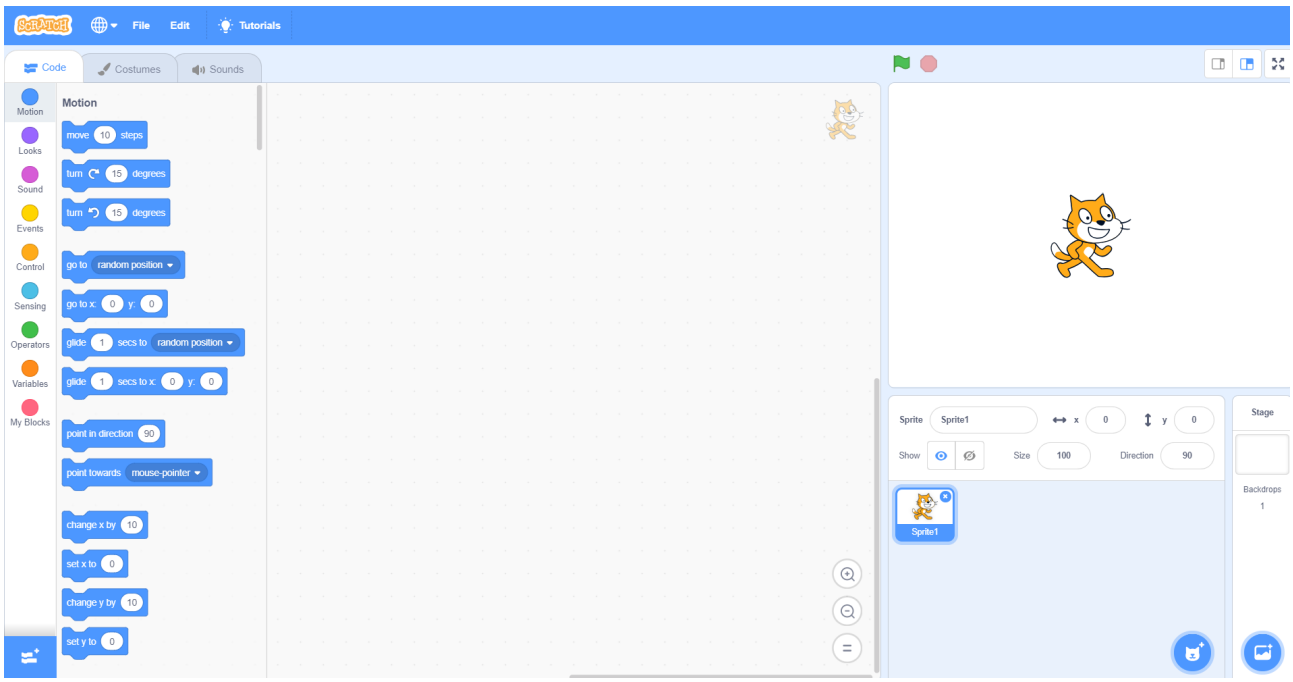

Figure 2.1: User Interface of Scratch [5]. On the left are all the blocks that can be dragged into the canvas in the middle. This is called the 'Code Area'. Sequences of blocks form things that the sprite on the right can do. In the right corner are all the sprites that belong to this program.

Blocks in Scratch

Scratch has a lot of blocks which are divided into multiple categories. Below an explanation of the categories of blocks that are available in Scratch. Every category has a unique colour that can be used to easily recognise a block.

Motion *Colour*: Dark blue.
The blocks that are in this category can be used to move and rotate sprites. The sprite can be moved to a specific coordinate or just X steps in a certain direction. The sprite can be rotated using degrees. These motion blocks can not be used on stages.

Looks *Colour*: Dark purple.
The blocks in this category can be used to modify the sprite and stage. They can change colour or change to a different version of themselves. The 'say'-block is also in this category. This block can make the sprite say something. This can be plain text or the contents from some variable.

Sound *Colour*: Light purple.
This category contains the blocks that play, stop and change sounds. The page with sounds can be found in the upper left corner as can be seen in Figure 2.1. On this page, the user can alter the speed of a sound or add an effect. This category also contains the blocks that can alter the volume of the program.

Events *Colour*: Yellow.
This one of the most important categories in Scratch. In this category contains all the blocks that trigger a code sequence. In Figure 2.2 can be seen that there are multiple things that can trigger a code sequence. The most simple blocks are the block where the green flag is clicked and the block where a key is pressed. The default key for this block is the space bar.

Figure 2.2: Different blocks in Scratch that trigger the start a code sequence

This category also contains blocks that are able to send out a message to other sprites or code sequences. As can be seen in Figure 2.2 there is a trigger block that activates the code sequence when a message is received. This blocks will only activate when another code sequence has sent out that specific message.

Control *Colour*: Light orange.
This category contains blocks that 'control' the code. In this category, you can find the for-loop, the if-statement, if-then-else-statement and the repeat-until. These are all blocks that look very similar to statements in textual language. This makes the switch from Scratch to a textual language easier since some concepts will already be clear.

Sensing *Colour*: Light blue.
In this category, you can find blocks that get activated only in a certain scenario. For example when the mouse pointer is touching a specific colour. Here you also find the blocks that need user input. An example is the block that saves your input in a variable called 'answer'. This variable has a different colour than the standard variable. Therefore it is easy to see the difference between a variable made by the programmer and a variable that gets filled when the user gives an input.

Operators *Colour*: Green
The blocks that are in this category can be used to apply mathematical operations such as the addition of two things. Those two things can be two plain texts, two variables or a combination of both. In this category, you also find the blocks that compare two things with each other. Here those two things can also be two plain texts, two variables or a combination of both. This category also contains blocks that are operators that are used on operators. There are the and-block, the or-block and the not-block.

Figure 2.3: These operator blocks can compare two things such as variables or plain text with each other.

Figure 2.4: These operator blocks combine two operators or give the opposite outcome of one operator.

Variables *Colour*: Dark orange.
This category contains all the blocks that can set, make and change variables. When a variable is being set, it means that you give the variable a specific value. When changing a variable you can only use numbers and either add or subtract a value from the value inside in the variable. Variables can have any name.

My Blocks *Colour*: Red
In this category, the user can design their own block. This way the user can let the blocks do anything they want. These blocks can, for example, shorten the code sequence when combining multiple blocks into one self-designed block.

Add an Extension
Scratch has the option to add an extension. These extensions are provided by Scratch and are there to lift the program to a different level or add another dimension to the program. One example of an extension is the Music extension with which the user can play specific instruments. There are also extensions for which an extra piece of software is needed. One example is the collaboration with micro:bit.

As we can see in Figure 2.5 the blocks in Scratch have specific shapes. These shapes show exactly which blocks can be placed in a code sequence and which blocks have to be dragged into another block. The blocks that are dragged into other blocks only fit there when the left and right side match with the left and right side of the spot that the user wants them.

Figure 2.5: The variable on the left will fit into the operator in the middle but not in the operator on the right. The operator in the middle will fit into the operator on the right.

2.2 Misconceptions

A misconception is a concept that someone has not understood the right way. Smith [9] says "Misconceptions arise from students' prior learning, either in the classroom (especially for mathematics) or from their interaction with the physical and social world.". Therefore it is important that learning new concepts is combined with resolving misconceptions around it.
The word misconceptions has many synonyms such as 'alternative conceptions', 'alternative frameworks' and 'preconceptions' [10].
Research into misconceptions has been done in many scientific fields such as electrochemistry [11] and physics [12].

2.2.1 Programming misconceptions

We will be looking specifically at programming misconceptions which Sorva [13] defined as "understandings that are deficient or inadequate for many practical programming contexts".
Programming misconception can be tedious during the process of making a process. Not having the right understanding of a programming concept can lead to frustration and irritation by the programmer. Since we will be working with children who are in the process of learning how to program, it is important that they have as little as possible misconceptions. This so that beginning programmers such as these children will not see programming as something impossible.

2.2.2 Research into programming misconceptions

Research that looks at programming misconception is not new. The current research into programming misconceptions is mostly done on Computer Science students [14], beginning programmers [15]. The research into programming misconceptions that looks at children under 18 is not very much, but some examples are [16–18].
Most of the programming misconceptions that have been written about have been combined in a list compiled by Sorva [19]. This list contains misconceptions that are found in research, up until the year 2012, about beginning programmers who take part in courses that are introductions to programming. The list does not contain misconceptions that are very specific for a programming language.
The misconceptions are grouped based on their nature. One example of these groups is 'VarAssign' which contains misconceptions about assigning variables. Another example of these groups is 'Ctrl' which contains misconceptions around if-statements, for-loops and sequentiality of statements. Every misconception contains the source to their original research. 'Local' means that Sorva or one of his colleagues has observed the misconception themselves.

Approach

To answer the research question, "What is the origin of programming misconceptions of children from ages 8 to 11 and how can these be resolved?", proposed in Subsection 1.1.1 we are doing interviews (See Section 3.2) with children that are from the ages 8 to 11. We are measuring which misconceptions (See Section 3.1) they have and what the reasoning behind that is. The last step in this research is analysing the interviews (See Section 3.3) and drawing conclusions.

3.1 Misconceptions

To measure the programming misconceptions that the children could have, we made sequences of Scratch-blocks that could induce a misconception. We look at the programming misconceptions from the list with misconceptions provided by Sorva [19]. This list is composed of other papers that describe programming misconceptions of mostly adults. Next, we will also test on basic concepts. These concepts will be specifically based on Scratch, this to see if the children have the misconceptions or just do not understand a certain block in Scratch.

3.1.1 Optional Misconceptions

In the list provided by Sorva [19] are misconceptions that can arise when programming in Scratch. Therefore, they are interesting to test on children. A good example is misconception *9: A variable can hold multiple values at a time / 'remembers' old values*. This represents one of the basic concepts of programming and should, therefore, be one of the first things to be learned when programming.
Since we are using Scratch there are several misconceptions on this list that are not applicable because they are based on a textual programming language. A good example is misconception *126: 'The dot operator can only be applied to methods'*.

3.1.2 Selected Misconceptions

We have chosen the following misconceptions from the list provided by Sorva [19]. The numbers refer to the numbers in the original list. We chose these misconceptions because they are applicable in Scratch and therefore it is possible for the children to have these misconceptions. They also represent the basic concepts of programming which are important when you start to learn how to program.
9: A variable can hold multiple values at a time / 'remembers' old values.
11: Primitive assignment works in opposite direction.
17: The natural-language semantics of variable names affects which value gets assigned to which variable.
23: Difficulties in understanding the sequentiality of statements.
24: Code after if statement is not executed if the then clause is.
26: A false condition ends program if no else branch.
27: Both then and else branches are executed.
28: The then branch is always executed.
160: Confusing textual representations with each other.
In Appendix C all code sequences described above are visually presented.

In addition to the misconceptions from the list, we also added some code sequences that have concepts in them that are specifically targeted on Scratch. These concepts represent basic concepts in Scratch that are not fully covered by the misconceptions described above but are interesting to test on. This because these blocks are widely used in code sequences in Scratch. For example, variables are used as scorekeepers in games made in Scratch. What these blocks do has been explained in Section 2.1.1.

- **Control Block: Repeat X times**
- **Control Block: If-then / If-then-else**
- **Operator Block: X + X**
  X could be a variable or a value.
- **Operator Block: X $>$ X / X $<$ X / X = X**
  X could be a variable or a value.
- **Variable Block: Set variable to X**
- **Variable Block: Change variable by X**

By having code sequences that contain these blocks we try to find out whether the children understand exactly how such blocks work. This is important because a lot of these blocks are the basics when it comes to any Scratch-project and any programming language in general.

3.1.3 Measuring Programming Misconceptions

We will test whether the misconception is present with the child by showing the child a sequence of Scratch-blocks. By asking specific questions about these sequences we try to find the origin of the answer that they give. The Scratch-blocks are made in such a way that specific misconceptions can be tested. For example in Figure 4.1, This sequence represents misconception *9: A variable can hold multiple values at a time / 'remembers' old values*. This way we can clearly see which misconception are present with the children. Because we have so many different code sequences, we can test many different misconceptions.

3.2 Interview

To measure which conceptions the children do and do not have we conduct an interview. The interview takes place with one child at the time. Otherwise, the children could influence each other's answers. The interview takes place in Dutch since the children come from Dutch primary schools.

3.2.1 Consent Forms

Before the children can participate in this research their parent(s) need to sign a consent form. This consent form contains information about the research and information about the rights that the children have during the interviews. By signing the consent form the parents state that they understand what the research is about, why it is done and that their child is allowed to participate. The consent forms are handed out to the children with the question to get it signed by their parents. Once the form is signed the children can participate in our research. See the consent form in Appendix B

3.2.2 Interview Protocol

The interview starts with a questionnaire that the children need to fill out themselves. This questionnaire contains questions to identify the about of knowledge the children have about Scratch. The questions are about their age, their previous experience with Scratch and other languages they have used. These other languages might influence how they understand programming concepts. See Appendix A for the questionnaire.
Subsequently, we conduct the main part of the interview which is measuring the misconceptions. First, we explain that we are doing this interview as if we were on the opening user interface of a new Scratch-project 2.1. This opening user interface always has the cat that can be seen in Figure 3.1 as the starting sprite. This user interface and this sprite will be very recognisable for the children. Every one of our code sequences starts off with the block 'when *green flag* clicked' which is also a very recognisable block for the children.
Then we explain that we will show them a sequence of Scratch-blocks, such as Figure 4.1. Based on this sequence they need to tell us what happens when the green flag is clicked and why. When it is relevant we will also ask why some other branch in this sequence is not happening?. This way we really make the children think about why they think their answer is the right answer.

Figure 3.1: Cat used as default sprite in Scratch [5]

The code sequences have different difficulties. This so that we can test the "simpler" code sequences on the children that are fairly new to Scratch and "harder" code sequences on children that say that they have done more with Scratch outside of school. Based on the answers they give we decide which code sequences we present them with. For example, if we notice that the child does not seem to understand the basics, we continue with the simpler sequences.

3.3 Analysing Interviews

Once all the interviews have been conducted and transcribed we analyse them. During this analysis, we look at all the interviews and see which misconceptions are present with the children. By looking at the explanations behind their answer we try to find the reason behind the misconception. During this process, we also look at which of the basic concepts of Programming/Scratch are not clear with the children. From there we can build a trajectory which contains concepts that should be clear before continuing onto more complicated things.

Results

4.1 Interviews

We have interviewed fifteen children between ages 8 and 11. Of the fifteen children, three are girls and twelve are boys. All of these children participated in extracurricular activities for gifted children. They have all used Scratch as one of these extracurricular activities. Some of them made clear that they have also programmed with other languages such as Python and Javascript. The interviews took about 10 minutes in which we tried to get an idea of the Scratch knowledge they had and at the same time create a challenge for them with the code sequences. We did this by listing to the answers they gave and deciding which code sequence was best fitting to ask next. During the interviews, we asked about seven to ten different code sequences.

Gender/Age	**8**	**9**	**10**	**11**
Boy	2	6	3	1
Girl	1	-	2	-

Table 4.1: Distribution of the gender versus the age of the children that were interviewed.

4.2 Misconceptions

We have at least one answer per code sequence that we made. All code sequences are presented in this Section and in Appendix C. All the correct answers to the code sequences in Appendix C are presented in Appendix D. Below we discuss the concepts that had an interesting outcome. The numbers of the misconceptions refer to the original list by Sorva [19]

Variable

9: *A variable can hold multiple values at a time / 'remembers' old values* [19].
With the code sequences in Figure 4.1 and Figure 4.2 we tested if the children understand what happens when you assign a value to a variable. We tested this with two different code sequences since assigning a number to a variable could have a different thought process than assigning a word to a variable.
From the thirteen children that we tested on these code sequences, five of them gave the right answer of both code sequences and said that the last value added to the variable is the value that is currently in that variable and therefore is the value that the cat says.
One of the other children understood that the cat says what the variable contains. This child did not understand that a variable can only contain one value. With code sequence 4.1 his answer was 30 meaning that the 20 gets added to the 10 that was already in A. With code sequence 4.2 the cat would say both words added to the variable back to back. So this child has misconception 9.

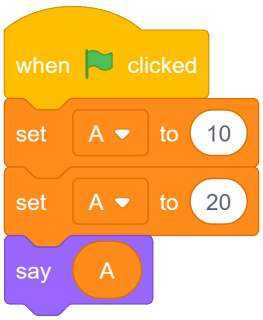

Figure 4.1: Sequence of blocks which represent misconception 9 [19]. This is the English version of the code sequence in Figure C.1.

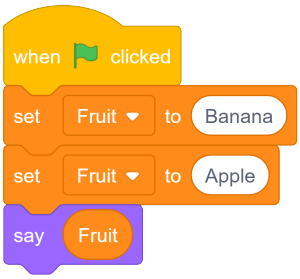

Figure 4.2: Sequence of blocks which represent misconception 9 [19]. This is the English version of the code sequence in Figure C.2.

Six of the other children said that the cat would literally say 'A' and 'Fruit'. Here we noticed that these children do understand seem to understand what the 'say-block' does but they do not understand the difference between a say-block with a variable and a say-block with plain text, see Figure 4.3.
Two of these six children said that in code sequence 4.1 there are 30 A's. This they explained by saying that 10 + 20 = 30 so there are 30 A's, but the cat only says one A.

Figure 4.3: The left block has as output 'A' and the right block has as output the value that is inside variable A.

One of the children had their own explanation. With code sequence 4.1 he understood that variable A contains 20. But the cat would say 'Hello!' which is the default value when you drag the say-block onto your canvas. With code sequence 4.2 he said that instead of text we would see pictures of the fruits that are in the variable. Where in sequence 4.1 he understood that the variable only contains one value, in this sequence, the variable still contains both values assigned to the variable.

Variable Assignment

11: *Primitive assignment works in opposite direction* [19].
17: *The natural-language semantics of variable names affects which value gets assigned to which variable* [19].
With the code sequences C.3 and C.4 we tested whether the children understand what happens when you assign one variable to another. We did this with two different sequences, one with numbers and one with words. This so we could see if the children understand the concept.
Out of the six children that we tested on code sequence C.3 only one gave the correct answer.
Three of the children said that both variables would contain the same value at the end of the sequence. This indicates that they have misconception 23: *Difficulties in understanding the sequentiality of statements*.
One of the other children said that the variable 'Fruit' contains 2 values and variable 'Dier' only contains its original value. The child seemed to think that a 'Fruit' can not be assigned to a 'Dier'. This indicates misconception 17.
Out of the six children that we tested on code sequence C.4 four gave the correct answer. One of the other children said that variable A still contains value 1 and value B contains 3 because 2+1=3. This indicates that this child has misconception 11.
The one remaining child said that variable A would contain the letter B and the end of the sequence.

Textual Representation

160: *Confusing textual representations witch each other* [19].
With the code sequences in Figure C.6 and Figure C.7 we tested if the children understand that there is a difference between the number 2 and the word two. We did this by making a variable A that contains the number 2 and then using an operator block to 'test' what is in A. In the code sequence C.6 we asked whether the number 2 is in A. In code sequence C.7 we asked whether the word two is in A. In Figure 4.4 we see how this looks different.

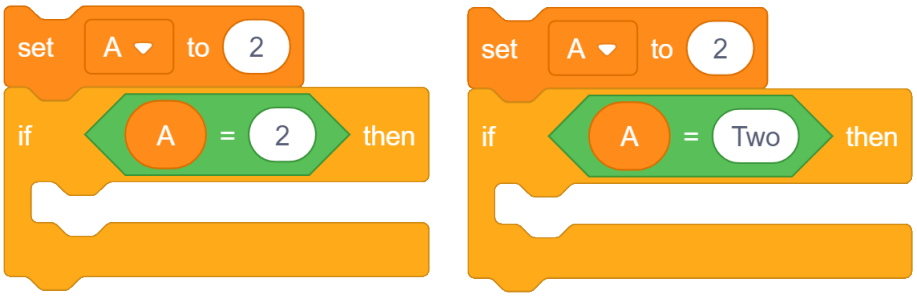

Figure 4.4: In the left code sequence similar to code sequence C.6 the if-statement is true. In the right code sequence similar to code sequence C.7 the if-statement is false.

From the fifteen children that we tested on these code sequences, seven of them do not think there is a difference between the word two and the number 2. They think that in both code sequences the if-statement is true. These children have misconception 160.
Seven of the other children explained to us that in the code sequence C.7 the if-statement is not true because the word two and the number 2 are not the same.
With three of the children that understood the different representations we also noticed misconception 26: *A false condition ends program if no else branch*. They thought that when the if-statement is false, the program does not continue and therefore does not do anything that came after the if-statement.

No Initialisation

With the code sequence in Figure C.14 we tested whether the children understand what happens when we do not initialise a variable. We did this by using an operator block to 'test'
We found that from the eleven children that we tested on this sequence, eight of them said that we do not know what value A has and therefore the if-statement is false. Two of the other children said that if-statement is true because the variable contains this value. The one remaining child said that the variable does not contain the value but since there is a say-block everything inside the if-statement is still said by the cat. This is misconception 28: *The then branch is always executed*.

If-Then-Else-Statements

27: *Both then and else branch are executed* [19].
28: *The then branch is always executed* [19].
With the code sequence in Figure 4.5 we tested whether the children understand what happens with an if-then-else-statement. We did this by using an operator block that will be false so that the if-statement is false and the else-statement will be triggered.

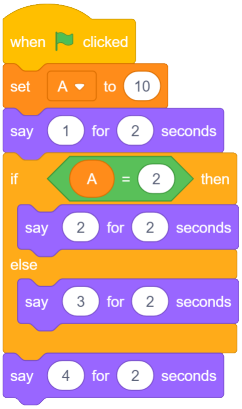

Figure 4.5: Sequence of blocks which represent misconception 27 and 28 [19]. This is the English version of the code sequence in Figure C.8.

Out of the nine children that we tested on this sequence, three gave the right answer. They all understood that the if-statement was false and therefore the else-statement became activated. They also noted that the 'say 1' block and the 'say 4' block are always executed since these are part of the main branch.
One of the other children seemed to understand what the if-then-else-statement does but gave the wrong answer. He said that the if-statement was true and therefore the else-statement will not be activated.
Two of the other children said that the cat says 1, 2, 3 and 4 because that is just how it works. one of the two noted that 'A has to go twice' which would mean that A becomes 20 after two assignments of 10.
The rest of the children do not seem to have a clear understanding of what an if-then-else-statement does. They seemed to have not read the sequence carefully to understand what happens.

Operator Blocks

With code sequence C.5 we originally wanted to test on the sequentiality of statements but during the interviews, we noticed that this sequence and code sequence C.10 were more fitting to test the operator blocks. Specifically, the operator block that adds two things together and the operator block that joins two things together.

Figure 4.6: On the left the operator block that adds two things such as plain text or variables together. On the right the operator block that joins two things such as plain text or variables together.

Out of the five children that we tested on code sequence C.5 one gave the correct answer.
All the other children said the cat would literally say 'X + X'. One of these children said that the cat would say 'A+B' Two of these children said that the cat would say '1+2', so they understood the idea but did not understand that the sum of this value would go into the variable 'Sum'. The one remaining child said that the cat would say '3+4'. Here we see misconception 23: *Difficulties in understanding the sequentiality of statements*.
None of the children that we tested on code sequence C.10 were able to give the right answer. They all said that the cat would say '3' which would be the right answer if this was the operator block used in code sequence C.5.

What we notice here is that the two operator blocks in Figure 4.6 sound very similar and do similar things. The English versions of these blocks are more clear than the Dutch version that we used during the interviews. So one of the reasons that none of the children gave the right answer for code sequence C.10 could be that the translation of the 'join' block is so that it reads as if two values need to be added together. If you read the Dutch 'join' block out loud it is very logical to think that this block has the same function as the operator 'addition' block.

Multiple operator block

With the code sequence C.17 we tested whether the children are able to understand what happens if there are two conditions in an if-statement. We did this by having two variables that will be tested in the if-statements, for this we used an AND-block 4.7. Both operator blocks have to turn out true for the if-statement to execute. In this case, one of the operator blocks will not be true.

Figure 4.7: This is an AND-block, Both conditions have to be true before this block has true as output.

Four of the eight children that we tested on this sequence understood that both operator blocks have to be true which they are not and therefore the cat will not say anything.
The other four all seemed to think that because B contains value 1, 1 is smaller than 2, so the operator block is true. The other operator block was already true so the whole if-statement is true.

For-loop

With code sequence 4.8 we tested the ability of the children to understand how for-loops works. We did this by assigning the value 10 to variable A. Then we add 10 to the value of A, this last step is repeated 3 times.

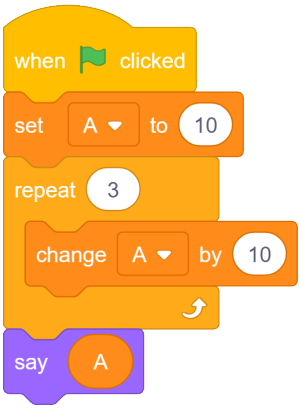

Figure 4.8: Sequence of blocks could lead to a misconception around for loops. This is the English version of the code sequence in Figure C.11.

Out of the five children that we tested on this sequence, three gave the right answer. They understood that 10 gets added three times. One of these three said 10 + 30 = 40. It is debatable whether this child understood what the for-loop does or just used his mathematical knowledge to come to the right answer.
One of the remaining two children said that at the end variable A contains value 10 since it says 'Change A to 10'. Here we notice that it is very important to read what the block says because that contains many clues about what it does. We asked the children to read the code sequence out loud, these seemed to help with understanding what the block does.
The other child said that the letter A will be said 30 times. So this child seems to think that the say-block belongs to the for-loop. The child might not have taken a good look at the sequence, saw 3 and 10 and guess 'something with 'A and 30'

4.2.1 Origin of Misconceptions

Based on the answers given by children, we notice multiple origins of misconceptions.
First off, we see that a lot of concepts are unknown, despite the fact that these concepts are basic concepts in other languages such as textual languages like Python. One of these concepts is the significant difference between a number represented in an amount and a number represented in a word. During the interviews we noticed that only half of the children understand that there is a difference between these two representations. Another concept is variables During the interviews, we noticed that half of the children do not fully understand. They think that a variable can hold multiple values or they think that variables get linked to each other when the value of one variable is assigned to another variable.
Secondly, we notice that children understand a block in Scratch when the block has the default version. For example, the 'say-block' has plain text 'Hello!' as the default value. When changing the block to another version, for example, a 'say'-block that says the content of a variable, only 40 percent of the children notice that there is a difference between the two. See Figure 4.4 for the default version and the version that contains a variable. We also notice that the children do not take the colours of blocks into account.
Third, we find that not all children that were interviewed read the code sequences carefully. Some already gave an answer before they completely read the code sequence. Not thoroughly reading the code sequence leads to missing blocks that are important for the final answer.

4.2.2 Resolving and Preventing

Based on the origins of the misconceptions, we provided possible solutions that can resolve or prevent misconceptions.

Colours and Shapes
The colours and shapes of the blocks that Scratch provides have not been used much in relation to programming education. The colours and shapes of blocks reveal what a block does or in which other blocks it can be used. Using these two concepts in the teaching of programming could help understand the concept behind the block and the things that can be done with the block.
Reading
If children would read the blocks more carefully, the would have a better understanding of the concept of the blocks. We noticed that reading the block out loud helped with realising what the block does. The best example of this practice is the 'if-then-else'-block. When reading this block out loud, you say exactly what the concept behind the block is.
Teachers
We notice that it is important that the teacher knows what they are talking about. When the teacher has some prior experience with programming, they understand all the concepts and can explain them in such a way that the concept is clear for the children.
Tests
The knowledge of programming could, just like other subjects, be tested with tests. This way the teachers can keep track of the progress that the children make in their programming knowledge. The content of the lessons can then be adjusted to the outcome of the tests so that no misconceptions are formed and that acquired misconceptions can be resolved.

4.3 Tips for Teachers

Per concept, we propose a tip that can be used or way of explaining the concept. With all these tips we highly recommend to incorporate the colours of the blocks.

1. **'Say'- block**
   The 'say' block is a purple block. When using this block the sprite always gets a speech bubble next to him, even when nothing is said. The colour of the item that is said is very important for the actual thing that the sprite says. White means that it is literally said. Orange means a variable, value that is inside the variable is said. Green means operator, then you first need to do the chosen operation.

2. **Variables**
   Variables are a concept that is also part of other subjects such as mathematics. When you are teaching programming to children in primary school, they will not have had this concept yet. Therefore it is important that the variable gets explained carefully. To explain what a variable is, use recognisable things. You could compare a variable with a box. This box can only contain one thing. When a new value gets assigned to the variable the old value disappears because there is no room for more than one value.
   Make sure to always initialise a variable. Without initialisation, the program can do unknown things. This is also a good practice for understanding which value the variable currently contains.
   There is also the option of adding assigning the value of one variable to another. In Scratch when the block is 'set A to B' this means that the value of B gets copied and assigned to A. Now variable A en variable B have the same value. Variable B remains unchanged during this whole process.
   Every block that has anything to do with variables has an orange colour.

3. **Operator blocks**
   With operator blocks, you can add, subtract, divide and multiply two things. It needs to be clear that these operator blocks do things. When you put two things in, you get one output. This output can, for example, be put into a variable but it can also be said by the sprite.
   There are also operator blocks that compare two things. These blocks are mostly used to ask a question to the program. It is important to understand that these blocks do not set or change any variable that you try to compare. These operator blocks give true or false as output.
   Then there are also blocks that combine two operator blocks that compare things. Then the outcome of these inside operator blocks gets combined into a new output.
   Outside Operator blocks are always part of another block that is not an operator block.

4. **Nested blocks**
   When you have multiple blocks inside each other such as multiple loops or operators in operators, it is important to understand that the most inner block starts first. When we look at Figure 4.6 we see a variable inside an operator block that is inside a variable block. It is important to understand that the operator block creates an output that will be used as input for the outer variable block.

5. **If-then statements**
   These blocks are used when you have code that needs to be activated under certain conditions. It is important to understand that the if-then statement only gets activated when the condition is true, then the else-statement will not be activated. This is also the other way around. The main branch continues under the if-statement. Reading the if-statement and the condition out loud can help make clear what the idea is behind the block.
   An if-statement can be explained as if we are asking a question to the program. If the answer is yes, continue with the then-statement. If the answer is no, continue with the else-statement if there is one. After that, the main branch will be continued.
   An if-statement always needs to be combined with an operator block.

6. **Repeat block**
   Repeat blocks are there to not have to put in a piece of code multiple times if you want it to repeat multiple times. Repeat blocks are very easy to understand when you read them. It speaks for itself. You can put in exactly how many times you want to have the code repeated.
   The 'repeat until' block is a little more complicated. This block needs to be combined with an operator block. Every time you come at the top of the repeat block you need to ask the computer again if the operation is true. As long as the answer is yes, the repeat block gets activated.

Conclusion

In this research, we looked at programming misconception in Scratch by children between the ages 8 and 11 and the origins of those misconceptions. The measuring of the origins of the misconceptions was done by conducting interviews. During the interview, the children saw code sequences. They needed to explain what the output was and give the reasoning behind that answer. During the analysis, we found the origins behind the misconceptions.
We noticed that the children lacked knowledge of basic concepts. They did not notice differences between different blocks and they did not see differences between different concepts. We also noticed that the code sequences were not being read carefully which lead to not understanding what happens in the code sequence.

We propose a few solutions for resolving and preventing. First, we advise making of the colours and shapes of the blocks during the explanation of the concepts behind the blocks. These things reveal what a block can do and with which other blocks a block can be used. Then we advise making sure that the basic concepts of programming are well understood. This can be tested via tests, these also help to keep track of the expanding programming skills of the children. Explaining the basic concepts can be done by comparing them to concepts that the children do know already. Lastly, we find it important that the teachers have experience with programming, this helps with explaining the basic concepts.

In conclusion, it is very important for the teaching of programming education that the basic concepts are clear. Without basic concept we are not teaching them how to programming, we are teaching children how to make games. Games can be a tool but they should not be leading in programming education. On top of that, it is very important that we try to keep track of the expanding programming skills by doing regular tests. By doing so, misconceptions can be prevented and acquired misconceptions can be found and solved during the next lesson.

Further Research

It would be interesting to see whether the same misconceptions can be found in other programming languages used in programming education in primary schools. The background of the children has not been taken into account during this research, this might contain some interesting information in relation to the misconceptions that the children have.

Bibliography

[1] PO Doorbraakproject Slimmer Leren met ICT, "Programmeren in het PO." https://maken.wikiwijs.nl/74282/Programmeren_in_het_PO#!page-1843082, 2015.

[2] Computing At School, "Computing in the national curriculum - a guide for primary teachers." https://community.computingatschool.org.uk/resources/2618/single.

[3] N. Smith, C. Sutcliffe, and L. Sandvik, "Code club: bringing programming to uk primary schools through scratch," in *Proceedings of the 45th ACM technical symposium on Computer science education*, pp. 517–522, ACM, 2014.

[4] M. Lucassen, "Leren programmeren: waarom eigenlijk?." https://www.vernieuwenderwijs.nl/leren-programmeren-waarom-eigenlijk/, April 2017.

[5] MIT Media Lab, "Scratch." https://scratch.mit.edu, 2006.

[6] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. S. Silver, B. Silverman, *et al.*, "Scratch: Programming for all.," *Commun. Acm*, vol. 52, no. 11, pp. 60–67, 2009.

[7] MIT Media Lab, "Scratch Statistics." https://scratch.mit.edu/statistics/.

[8] K. Brennan, "ScratchEd Online Community Archive." https://scratched.gse.harvard.edu/index.html, 2009.

[9] J. P. Smith III, A. A. Disessa, and J. Roschelle, "Misconceptions reconceived: A constructivist analysis of knowledge in transition," *The journal of the learning sciences*, vol. 3, no. 2, pp. 115–163, 1994.

[10] K. S. Taber, "Alternative conceptions/frameworks/misconceptions," *Encyclopedia of science education*, pp. 37–41, 2015.

[11] M. J. Sanger and T. J. Greenbowe, "Common student misconceptions in electrochemistry: Galvanic, electrolytic, and concentration cells," *Journal of Research in Science Teaching: The Official Journal of the National Association for Research in Science Teaching*, vol. 34, no. 4, pp. 377–398, 1997.

[12] J. Clement, "Using bridging analogies and anchoring intuitions to deal with students' preconceptions in physics," *Journal of research in science teaching*, vol. 30, no. 10, pp. 1241–1257, 1993.

[13] J. Sorva, "Notional machines and introductory programming education," *ACM Transactions on Computing Education (TOCE)*, vol. 13, no. 2, p. 8, 2013.

[14] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Proceedings of the 41st ACM technical symposium on Computer science education*, pp. 107–111, ACM, 2010.

[15] P. Bayman and R. E. Mayer, "A diagnosis of beginning programmers' misconceptions of basic programming statements," *Communications of the ACM*, vol. 26, no. 9, pp. 677–679, 1983.

[16] A. Swidan, F. Hermans, and M. Smit, "Programming misconceptions for school students," in *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pp. 151–159, ACM, 2018.

[17] S. Grover and S. Basu, "Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic," in *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*, pp. 267–272, ACM, 2017.

[18] R. T. Putnam, D. Sleeman, J. A. Baxter, and L. K. Kuspa, "A summary of misconceptions of high school basic programmers," *Journal of Educational Computing Research*, vol. 2, no. 4, pp. 459–472, 1986.

[19] J. Sorva *et al.*, *Visual program simulation in introductory programming education*, pp. 359–368. Aalto University, 2012. ISBN:978-952-60-4626-6.

Appendix A

Interview Protocol

Interview

Leuk dat je mee wil doen met dit onderzoek! Ik ga je een paar basisvragen stellen en daarna gaan we kijken naar wat stukjes Scratch.

Hoe oud ben je?_____

Meisje/Jongen/Anders

Waar ben je goed in op school?

_____

Waarmee heb je al eens geprogrammeerd?

_____

Wat voor dingen heb je al eens in Scratch gedaan?

_____

_____

Appendix B

Consent Form

Universiteit Leiden

Toestemmingformulier Programmeeronderwijs

Beste ouder(s)/verzorger(s)

Vanuit de Universiteit Leiden zijn wij bezig met een onderzoek over programmeeronderwijs op de basisschool. Daarom willen wij uw kind vragen om mee te doen aan dit onderzoek.

In dit onderzoek willen we graag meten welke concepten van het programmeeronderwijs wel en niet goed begrepen worden door de kinderen. Specifiek zullen we kijken naar de programmeeromgeving Scratch.

De kinderen worden kort geïnterviewd waarbij ze een paar vragen beantwoorden over de programmeeromgeving Scratch. Hierna zullen er nog een paar vervolgvragen volgen m.b.t de eerder gegeven antwoorden.

Vertrouwelijkheid
Namen van kinderen zullen niet worden gebruikt in de publicatie van het onderzoek.

De kinderen mogen altijd stoppen met het onderzoek als zij dat willen.

Toestemming

Ik begrijp de achtergrond van het onderzoek en ik begrijp dat de resultaten geanonimiseerd zullen worden.

Als ouder/verzorger van _____(naam kind) stem ik in met deelname aan dit onderzoek

Geboortedatum kind:_____ Groep:_____

Datum:_____ Handtekening:_____

Appendix C

Code Sequences

These are the code sequences used to measure the misconceptions with the children. They are in Dutch since the children use Scratch in Dutch. The code sequence below are based on one or more misconceptions from the list provided by Sorva [19].

Figure C.1: Sequence of blocks which represent misconception 9 [19]

Figure C.2: Sequence of blocks which represent misconception 9 [19]

Figure C.3: Sequence of blocks which represent misconception 9, 17 and 23 [19]

Figure C.4: Sequence of blocks which represent misconception 16 [19]

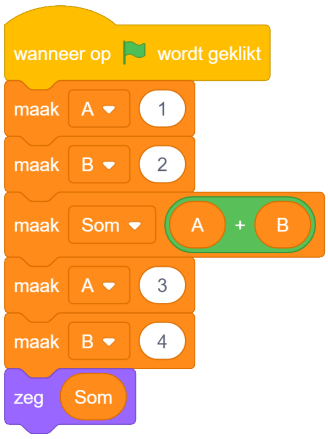

Figure C.5: Sequence of blocks which represent misconception 23 [19]

Figure C.6: Sequence of blocks which represent misconception 24 [19]

Figure C.7: Sequence of blocks which represent misconception 24, 26 and 160 [19]

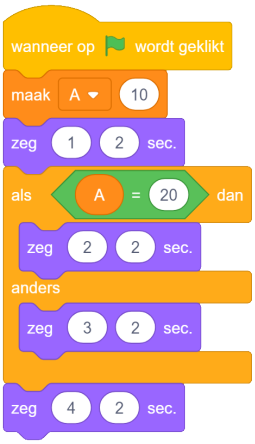

Figure C.8: Sequence of blocks which represent misconception 27 and 28 [19]

Figure C.9: Sequence of blocks which represent misconception 32 [19]

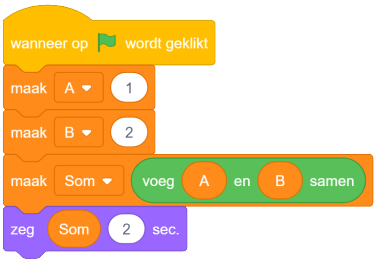

Figure C.10: Sequence of blocks which represent misconception 17 [19]

Below are all the code sequences that show concepts in Scratch that we also wanted to test. They are not directly related to one specific misconception from the list provided by Sorva [19].

Figure C.11: Sequence of blocks could lead to a misconception around for loops

Figure C.12: Sequence of blocks could lead to a misconception around for loops and if statements

Figure C.13: Sequence of blocks could lead to a misconception around for loops around if statements

Figure C.14: Sequence of blocks could lead to a misconception around not initialising variables

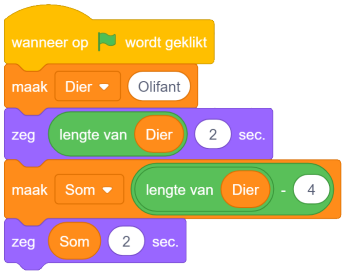

Figure C.15: Sequence of blocks could lead to a misconception around operator blocks in Scratch

Figure C.16: Sequence of blocks could lead to a misconception around operator blocks in Scratch

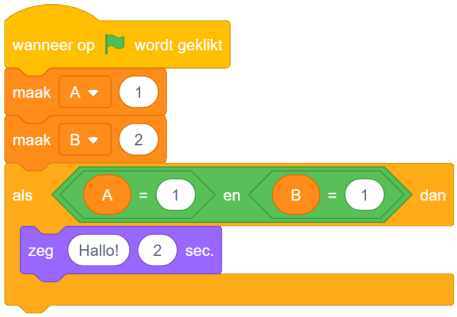

Figure C.17: Sequence of blocks could lead to a misconception around operator blocks in Scratch

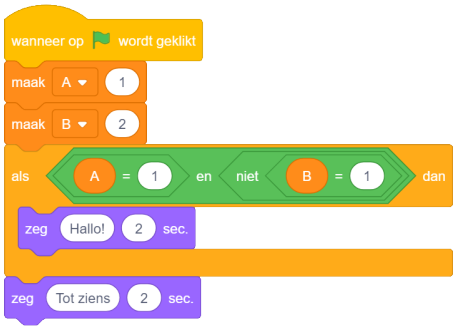

Figure C.18: Sequence of blocks could lead to a misconception around operator blocks in Scratch

Appendix D

Correct Result of the Code Sequences

1. Code Sequence C.1. *Correct Answer*: 20
2. Code Sequence C.2. *Correct Answer*: Appel
3. Code Sequence C.3. *Correct Answer*: Appel 2 seconds, Banaan 2 seconds
4. Code Sequence C.4. *Correct Answer*: 2 2 seconds, 2 2 seconds
5. Code Sequence C.5. *Correct Answer*: 3
6. Code Sequence C.6. *Correct Answer*: Hallo! 2 seconds, Tot ziens! 2 seconds
7. Code Sequence C.7. *Correct Answer*: Tot ziens! 2 seconds
8. Code Sequence C.8. *Correct Answer*: 1 2 seconds, 3 2 seconds, 4 2 seconds
9. Code Sequence C.9. *Correct Answer*: 1 2 seconds, 2 2 seconds, 3 2 seconds 4 2 seconds, 5 2 seconds, 6 2 seconds
10. Code Sequence C.10. *Correct Answer*: 12 2 seconds
11. Code Sequence C.11. *Correct Answer*: 40
12. Code Sequence C.12. *Correct Answer*: 20 2 seconds, 30 2 seconds, 40 2 seconds, 40 2 seconds
13. Code Sequence C.13. *Correct Answer*: 20 2 seconds, Kleiner 2 seconds, 30 2 seconds, Kleiner 2 seconds, 40 2 seconds, 40 2 seconds
14. Code Sequence C.14. *Correct Answer*: Doei 2 seconds
15. Code Sequence C.15. *Correct Answer*: 7 2 seconds, 3 2 seconds
16. Code Sequence C.16. *Correct Answer*: 1 2 seconds
17. Code Sequence C.17. *Correct Answer*: 'Nothing'
18. Code Sequence C.18. *Correct Answer*: Hallo! 2 seconds, Tot ziens 2 seconds