Programming Misconceptions for School Students

Swidan, Alaaeddin; Hermans, Felienne; Smit, Marileen

DOI
10.1145/3230977.3230995

Publication date
2018

Document Version
Submitted manuscript

Published in
ICER '18

Programming Misconceptions for School Students

Alaaeddin Swidan, Felienne Hermans and Marileen Smit
Delft University of Technology
Delft, The Netherlands
{alaaeddin.swidan,f.f.j.hermans,m.i.e.smit}@tudelft.nl

ABSTRACT

Programming misconceptions have been a topic of interest in introductory programming education, with a focus on university level students. Nowadays, programming is increasingly taught to younger children in schools, sometimes as part of the curriculum. In this study we aim at exploring what misconceptions are held by younger, school-age children. To this end we design a multiple-choice questionnaire with Scratch programming exercises. The questions represent a selected set of 11 known misconceptions and relate to basic programming concepts. 145 participants aged 7 to 17 years, with an experience in programming, took part in the study. Our results show the top three common misconceptions are the difficulty of understanding the sequentiality of statements, that a variable holds one value at a time, and the interactivity of a program when user input is required. Holding a misconception is influenced by the mathematical effect of numbers, semantic meaning of identifiers and high expectations of what a computer can do. Other insights from the results show that older children answer more questions correctly, especially for the variable and control concepts. Children who program in Scratch only seem to have difficulties in answering the questions correctly compared to children who program in Scratch and another language. Our findings suggest that work should focus on identifying Scratch-induced misconceptions, and develop intervention methods to counter those misconceptions as early as possible. Finally, for children who start learning programming with Scratch, materials should be more concept-rich and include diverse exercises for each concept.

ACM Reference Format:
Alaaeddin Swidan, Felienne Hermans and Marileen Smit. 2018. Programming Misconceptions for School Students. In *ICER '18: 2018 International Computing Education Research Conference, August 13–15, 2018, Espoo, Finland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3230977.3230995

1 INTRODUCTION

It is known from existing research that learning programming is difficult [3, 4, 11]. One source of difficulties is holding programming misconceptions [4, 24], which affects performance in writing or understanding code. A programming misconception is having an incorrect understanding of a programming concept or a set of related concepts, typically affected by prior knowledge from domains other than programming such as mathematics and natural languages [20].

Studying programming misconceptions involves identifying their possible origins in order for both learners and educators to rectify relating concepts. Misconceptions have a harmful effect on the performance of students. The effect starts early [24] and may remain for a long time [17]. They have been found to cause failure in introductory programming courses and, in the long run, even cause students to drop out of programming education [12]. Previous studies focused nevertheless on introductory courses in universities. Nowadays, CS education and programming is increasingly introduced to younger students in primary and secondary schools [2, 10]. Many countries have already integrated programming activities into their school curriculum [9]. Moreover, new programming languages and programming environments are implemented especially for younger children. An example is Scratch[1], a block-based language which is developed by MIT with the aim of teaching children how to program. While CS education is moving down to schools, little is known on whether children develop certain misconceptions at this stage. In this study we aim at exploring the programming misconceptions held by school-age children. We developed a multiple-choice questionnaire containing 11 questions representing a selected set of programming misconceptions known from previous research [21]. In total, 145 children aged between 7 and 17 participated in our study. The participants, who were required to have an experience in programming, additionally provided reasoning for their answers in open-ended texts. From the data collected in this survey, we aim to answer the following research questions:

- **RQ1** Which programming misconceptions are the most common among Scratch novice programmers?
- **RQ2** How do children holding those misconceptions explain their answers? How do their explanations differ from the ones of children understanding the concept correctly?
- **RQ3** How do age and previous programming knowledge affect the holding of a misconception or the correct understanding?

2 BACKGROUND

Research defines a programming misconception as an incorrect understanding of a concept or a set of concepts, which leads to making mistakes in writing or reading programs [20]. Misconceptions can be related to basic, yet fundamental, programming concepts, not only to advanced concepts. Apart from syntactic mistakes, there

ICER '18, August 13–15, 2018, Espoo, Finland

ACM ISBN 978-1-4503-5628-2/18/08…$15.00
https://doi.org/10.1145/3230977.3230995

[1] https://scratch.mit.edu/

seems an agreement among researchers on particular concepts being difficult for learners. Those language-independent concepts include variables, loops, and conditional statements [4, 7, 11, 13, 17]. An example of a programming misconception: the belief that a variable can hold multiple values, or the belief that a variable's assignment goes the opposite direction [4, 21]. In object-oriented languages, common difficulties are related to the scope and visibility of variables, modularization and decomposition and inheritance [8, 12].

Research focused on understanding where a misconception originates from. A programming misconception does not mean that the learner has a complete lack of knowledge, rather it indicates partial but self-interpreted knowledge which comes from domains other than programming [4]. Some of the known origins include the use of particular analogies in explaining a concept, the ambiguous and double meanings of some of the programming keywords in English as a natural language, and mathematics [4, 15, 19]. Du Boulay [4] introduced what he called the *"notional machine"* as one origin of programming misconceptions. The notional machine refers to the general properties that a student assumes of the machine executing their code. Having an incorrect understanding of the notional machine of a programming language is believed to be the cause of many misconceptions [13, 20]. For example errors were found as a result of what Pea [14] called the *"superbug"* which is the assumption that *"there is a hidden mind somewhere in the programming language that has intelligent, interpretive powers"* [14, p. 32], or *"forgetting about alternative branches because they are too obvious to merit consideration"* [22, p. 6].

Finally, the variety of misconceptions make it difficult for educators to take them fully into account. In this regard, Sorva [21] provides a comprehensive list of programming misconceptions collected from various studies [4, 6, 12, 19, 20]. In this study, we use Sorva's list as the starting point to investigate Scratch misconceptions in school students.

3 SETUP

The goal of our study is to explore the common misconceptions among school-age children in Scratch. To this end, we perform a questionnaire-based study. Participants are given a set of multiple choice questions; each question is a programming exercise in Scratch which tests the holding of a known misconception. Questionnaires have been used to assess the holding of programming misconceptions in previous research [12, 18]. Participants in most cases need to predict the outcome of the script to choose an answer. Figure 2 shows an example question from our study. We provide the full questionnaire (English version) here[2]. In the following sections we describe the study setup in detail.

3.1 Participants

In total 145 children took part in the study. Figure 1 shows the distribution of the participants over the reported age, ranging between 7 and 17 years. Among the participants, 102 (70.3%) are boys, and 51 (29%) are girls; one participant did not specify gender.

[2]http://cli.re/g1zyQM

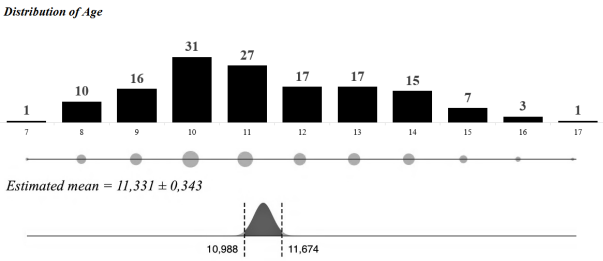


Figure 1: Age distribution of the participants. 70.3% of them are primary school students (12 years old and younger)

3.2 Study environment

We ran this experiment at NEMO science museum [3] in Amsterdam. Children visiting the museum were asked to join an experiment on programming but received no further information on what the experiment would measure. Participants did not get financial compensation for participation, but did receive a certificate for their efforts. The experiment was assigned a 25-minutes window per child and was run in a separate room in the museum that seated 8 children at a time. In total we spent 14 days at the museum, running the experiment for about 5 hours each day. During the experiment each participant had to fill answers to our web-based questionnaire in which the questions were put in static images. The children had access to the machine and the Internet, but we did not observe any participant opened other applications or pages than the questionnaire. The number of participants reported in this study is a subset of the total children who visited our booth, we filtered out participants who did not have programming experiences or who indicated guessing the answers. Asking children in such setup allowed us to study the holding of misconceptions on a sample that is less-dependent on specific teaching methodologies compared to an experiment run in a school.

3.3 Misconceptions selection

Sorva provides a comprehensive list of 162 programming misconceptions known from the literature [21]. We used the list as our starting point to investigate Scratch misconceptions. We followed a two-step approach to achieve that. **First, we selected the most common misconceptions from the list**. A misconception is common if indicated by at least two separate research works. This step reduced the list to 17 programming misconceptions. **Second, since we study Scratch misconceptions, we filtered out the misconceptions that do not fit Scratch as a language**. For example, we eliminated two misconceptions that concern the use of a loop control variable inside the loop's body. In Scratch, loops use a static value and no variable is necessary to iterate through the body. This resulted in 11 misconceptions, shown in Table 1.

3.4 Questions

Before presenting the misconception questions to the participants, we require the participant to answer five close-ended questions in the questionnaire. The answers to those questions indicate the

[3]https://www.nemosciencemuseum.nl/en/

Table 1: Programming misconceptions included in our study from [21].

Code[4]	Description	Prerequisite Concept(s)[5]	Question Description (Pseudo-code)	Misconception Choice(s)
M9	A variable can hold multiple values at a time	Variables	Set [X] to 10; Set [X] to 20; Say [X];	Gobo says 10, 20
M11	Primitive assignment works in opposite direction	Variables	Set [a] to 10; Set [b] to 20; Set [a] to [b];	a=20, b=0 a=20, b=10
M14	A variable is a pairing of a name to a changeable value. It is not stored inside the computer	Variables	We create a variable called message, Where is the variable stored?	Variable is not stored anywhere Variable is only visible on the screen
M15	Primitive assignment stores equations or unresolved expressions	Variables	Set [X] to 1, Set [counter] to [X+1]; Say [counter];	Gobo says X+1
M17	Natural-language semantics of variable names affects which value gets assigned to which variable	Variables	Set [big] to 1; Set [small] to 100; Set [big] to [small];	big=100 small=1 big=100 small=0
M23	Difficulties in understanding the sequentiality of statements	Variables	Set [number1] to 0; Set [number2] to 0; Set [total] to [number1] + [number2]; Set [number1] to 4; Set [number2] to 2; Say [total];	Gobo says 6
M26	A false condition ends program if no `else` branch exists	IF ELSE	IF touching the color [black] then { Say *"Auw!!"*; }; Say *"I am moving"*;	Gobo does not say anything
M30	Adjacent code executes within loop	Variables & Loops	Set [counter] to 0; Repeat 5 { Change [counter] by 1;}; Say [counter];	Gobo says 1 till 5
M31	Control goes back to start when condition is `false`	IF ELSE	Say *"I am moving"*; IF touching the color [black] then { Say *"Ouch!!"*; }; Say *"Done!!"*;	Gobo says *"I am moving"*
M33	Loops terminate as soon as condition changes to `false`	Variables & Loops	Set [number] to 1; Repeat until [number]=3 { Change [number] by 1; Say [number]; };	Gobo says 2
M150	Difficulties understanding the effect of input calls on execution	None	Ask [*"How old are you?"*] and Wait; Say *"Nice! I will move now"*; Move [10] Steps;	Gobo says *"How old are you?"* and immediately Says "*Nice! I will move now*" and Moves 10 steps.

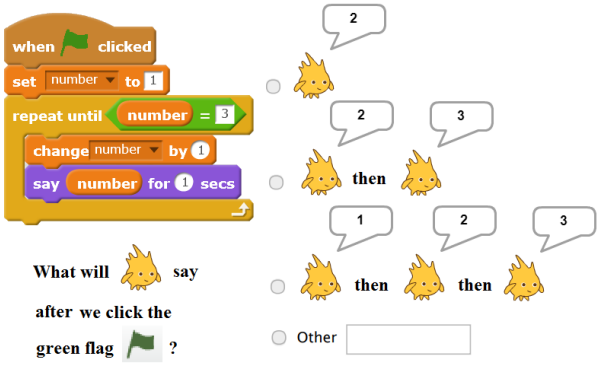


Figure 2: Question and possible answers for M33

participant's familiarity with these programming concepts: variables, IF statements and loops. We use the answers in an automatic branching logic so that we present a set of three to eleven multiple-choice misconception questions (See Table 1) that fit the knowledge of the participant. Each question is designed to elicit one of the well-known programming misconceptions. For this purpose we use programming problems similar to the ones suggested by Ma [12] for Java students. In our case, we design the question in Scratch both in English and Dutch. Scratch enables programming in one's native language, which eliminates the cognitive load for reading a foreign language for the local children and enables them to focus on the programming challenge [5, 24]. Figure 2 shows an example of one of the questions related to the misconception of a loop terminating as soon as the condition becomes false (M33). We ask the participants to predict the outcome of the program by asking *"What will Gobo say when we click the green flag?"*, where Gobo is one of the characters known in Scratch. The answers are categorized into *Holds_Correct* (Gobo says 2 then 3), *"Holds_Misconception"* (Gobo says 2), and *Other_Wrong* (Gobo says 1 then 2 then 3). An open-ended question follows so that the participant can explain the reasons behind the chosen answer. We used the open-ended text to filter the results. The aim of this filtering is to eliminate answers

[4]We add a prefix, M, to the original numbers assigned to the misconception in Sorva's list. This is for the sake of easy referencing in the paper.
[5]This column indicates how we assign questions to participants based on the familiar concepts they report.

for which children admitted that they either guessed the answer or did lack understanding of the question. We note that we initially received 1,306 answers from 178 participants. Due to the filtering process, 545 answers were eliminated. The number of participants included in the study went down to 145 since for some participants their whole answer set was eliminated.

4 RESULTS

This section provides an overview of the answers to the study's research questions based on the questionnaire's data.

4.1 Most common misconceptions

[RQ1] Which programming misconceptions are the most common among Scratch novice programmers?

To answer RQ1, we analyze the answers for each misconception question. Figure 3 presents the percentage of participants who selected the misconception choice per question. The three most common misconceptions are related to different concepts: (i) the sequentiality of executing code (M23), (ii) the variable holding multiple values at a time (M9), and (iii) the human-computer interactivity and its effect on execution (M150). Moreover, we notice that among the least common misconceptions are the misconceptions related control statements: loops and conditions (M31, M33 and M26). In these cases, however, otherwise wrong choices are popular among participants, which indicates the general difficulty to understand those concepts.

4.2 Insights from children explanations

[RQ2] How do children holding those misconceptions explain their answers? How do their explanations differ from the ones of children understanding the concept correctly?

First we quantitatively analyze the open-text provided by participants per their answer category for the top three common misconceptions (see Table 2). To highlight the thinking process of the children both holders of a misconception or the correct concept, we further explore the open-ended answers provided by participants for M14 and M26 in addition to the ones in Table 2. M14 is chosen because two aspects of the misconception are provided in the multiple choices. M26 is chosen because although a choice representing the misconception is not provided separately (See Section 5.4), participants still indicated holding the misconception through their provided open-ended text.

M23: Difficulties in understanding the sequentiality of statements
Participants with the misconception=56.2%, Correct=42.5%, n=73

Misconception: Participants show a focus on the mathematical operation itself, not on the sequence. This is shown in the the top three words as the the words include *"values"* and *"add"*. Additionally, we find explanations such as *"because if you add number 1(4) and number 2 (2) [then] it will say 6"* or simply *"Basic math man"*. Some participants assumed an automatic aspect of the operation: *"when you change values of these variables total value changes also"*, and *"4 + 2 = 6 the computer should calculate that for you"*.

Correct: Participants are able to identify the sequential nature of the code. In the most frequent words we find the word *"before"* which indicates an order. One participant, for example, explains: *"Because total is set to [no.1] + [no.2] so it equals 0. The variables changed after that are irrelevant"*. Another participant suggested a *"fix"* to the code: *"If the block set [total] to [number1]+[number2] was put lower then it would have worked"*.

M9: A variable can hold multiple values at a time/*"remembers"* old values
Participants with the misconception=42.9%, Correct=42.9%, n=63

Misconception: Most participants referred to the code in the question as their reason without extra highlights. The frequent words used include the words *"first"* and *"numbers"* which shows the attention these participants give to the old value of the variable and both numbers used in the exercise respectively. However, one participant, despite choosing the misconception answer explains: *"I'm not sure, but if two [instances] of the same variable are used with different numbers, if possible, [the result] will give both numbers. Otherwise it would be 20 because that was the last change"*.

Correct: One of the most frequently used words is the word *"variable"*. This might indicate that participants have a basic understanding of what a variable is and thus they use the term more frequently. Moreover, we notice from the explanations of some of the participants referring to the last change made to the variable's value. Examples on this include: *"X= 20 is after X= 10 and the later one will overwrite the earlier one"*. However, one participant shows a full awareness of this aspect of variables: *"variable X is changed to 10, then 20, and a variable can only have one value"*.

M150: Difficulties understanding the effect of input function calls on execution
Participants with the misconception=39.1%, Correct=35.7%, n=115

Misconception: The word *"order"* is the second most frequent mentioned by these participants, highlighting the importance they give to the sequential execution that respects the blocks' order. One participant explains for example: *"This is the order from up to bottom"* and another says: *"because [the answer] is in the correct order"*. Moreover we notice that some participants use the word *"answer"* identifying the question-answer nature of the program. However they still provided the wrong answer because of a variety of wrong assumptions such as that since it is not possible for the participant to fill an answer then the computer will continue. Another assumption is that the question is directed towards the computer, therefore the computer will answer and continue the execution: *"Gobo says i am .. years old and directly says ...[continues the next blocks]"*, and another participant: *"I think that in the game Gobo is asked how old he is, then he ...[continues the next blocks]"*.

Correct: We notice again two forms of reasoning when participants give the correct answer. One has a direct approach and relies on the word *"wait"* being present in the question and in the answer text, stating that the choice comes because *"there is wait"* in both question and answer. The second shows that participants have more precise recognition of the question-answer nature of the program, and hence the need for an answer from the user so that the program can continue. One participant explains: *"if he asks you then you have*

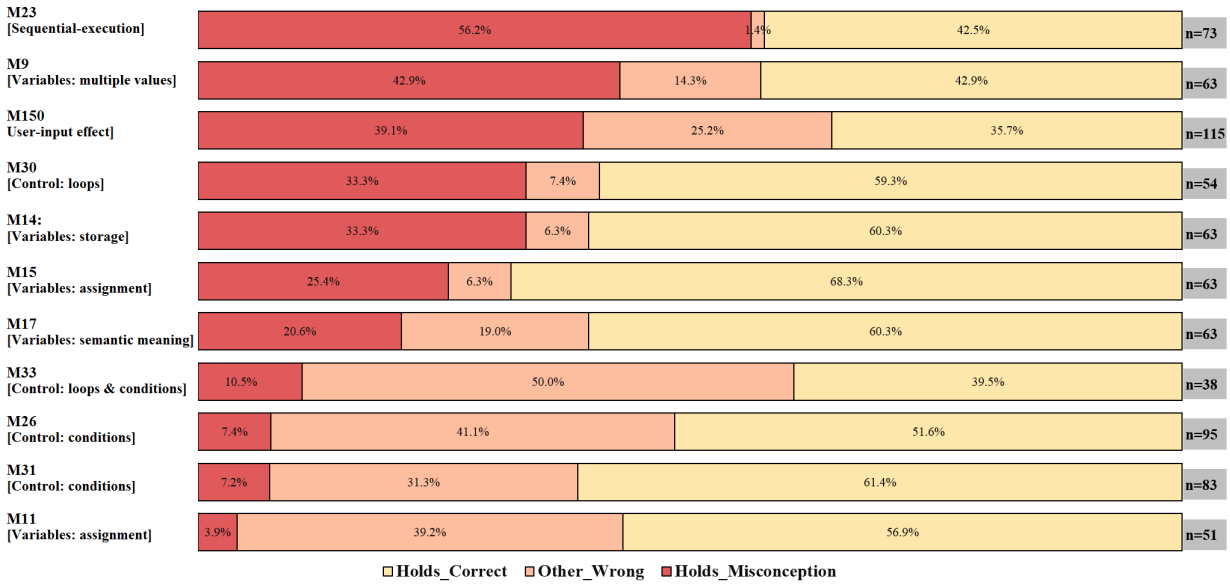


Figure 3: Misconceptions and their answer distribution, ordered from most to less common. M14, M26 in addition to the top 3 misconceptions are further discussed by exploring participants open-ended text.

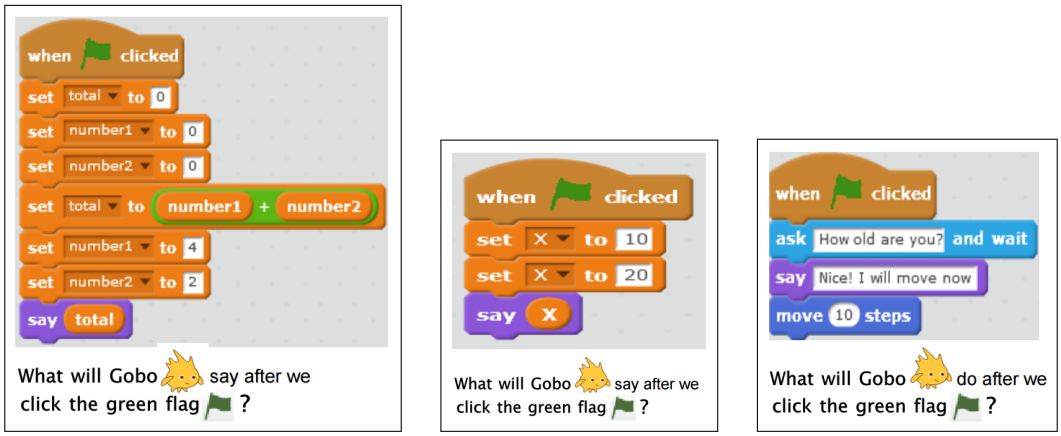


Figure 4: The questions representing M23 (Left), M9 (Middle) and M150 (Right)

Table 2: Most common words in the open-end text provided by children for the top three misconceptions

	M23: sequential-execution			M9: variable's multiple values			M150: user-input effect		
	Rank1	Rank2	Rank3	Rank1	Rank2	Rank3	Rank1	Rank2	Rank3
Holds Correct	before	numbers	values	then	set	variable	wait	answer	until
Holds Misconception	value	then	add	then	first	numbers, makes	answer	order	program
Other Wrong	Not applicable			then	say	picture	what	know	does

to type an answer, then he will respond", and another participant says: *"because you have not typed anything"*.

M14: A variable is (merely) a pairing of a name to a changeable value (with a type). It is not stored inside the computer

Participants with the misconception=33.3%, Correct=60.3%, n=63

Misconception: In total, 33.3% of the participants (n=63) chose a misconception choice (see Figure 3). In a detailed manner, 7.9% of the participants chose the answer indicating that *"the variable is not being stored anywhere"*. Participants who provided their reasons here seem confused by the built-in option provided by Scratch: Cloud variable (stored on server), which was shown as part of the question. Two participants highlighted that their choice came as a result of this option not being ticked, which is the default in Scratch. 25.4% of participants chose the answer that indicates *"the variable is only visible on the screen"*. For these participants, the location at which the storage occurs is important and needs to be sensed. One participant says: *"because you can't see it anywhere else"*, and another one: *"there is nowhere for it to be so it just sits there"*. Another participant, despite choosing the wrong answer, indicates an analytical approach that is one step-away from being correct. The participant is uncertain where a variable should be stored because the program is not run, saying: *"the code will only set to value when run, and as the code is not yet running, the variable is moot"*.
Correct: Participants who answered correctly vary in their reasoning. Some give a concrete reason, for example *"It sits in the RAM memory"*, and *"all computers store data in the hardware, to know what they need to do"*. Others focus on the need to save the whole program for Scratch to *"remember"* it later.

M26: A false condition ends program when no else branch
Participants with the misconception=7.4%, Correct=51.6%, n=95

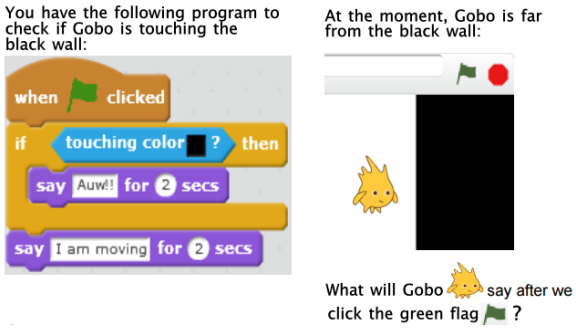


Figure 5: The question representing M26

Misconception: Although we have not provided an answer option to represent the misconception (See Sect. 5.4), some participants (7.4%) show they hold the misconception after analyzing the text they wrote. For those, the code does not execute at all because the condition is `false`. For example, one participant says: *"he [Gobo] does not touch the black, so nothing happens there"*. Another participant finds it illogical to execute the code: *"he [Gobo] is not touching the black so why would he do the commands that only apply to him if he is touching the black"*.
Correct: Participants highlight the false condition as a motive to their answer, from the opposite perspective to the participants with the misconception. The condition being `false` means the program runs, but parts of it are skipped. For example, one participant states: *"he skips the Auw part because he is not standing on the black"*, and another participant agrees: *"Gobo says only I am moving because he does not touch the black wall"*.

4.3 Effect of age and previous programming knowledge

[RQ3] How do age and previous programming knowledge affect the holding of a misconception or the correct understanding?

4.3.1 Age factor. For the effect of age analysis, we exclude the age points 7 and 17 because only one participant in each of these age categories answered the questionnaire. Results show that a positive correlation exists only between age and holding the correct concept (Spearman's Rank Correlation p-value =0.005). In words, the older the child the more they answer correctly. Additionally, when considering the category of the misconception according to Sorva's original classification (see Table 1), positive correlation is found between age and correctly answer the misconception questions under the *"Variable"* category (Spearman's Rank Correlation p-value=0.015) and *"Control"* category (Spearman's Rank Correlation p-value=0.048).

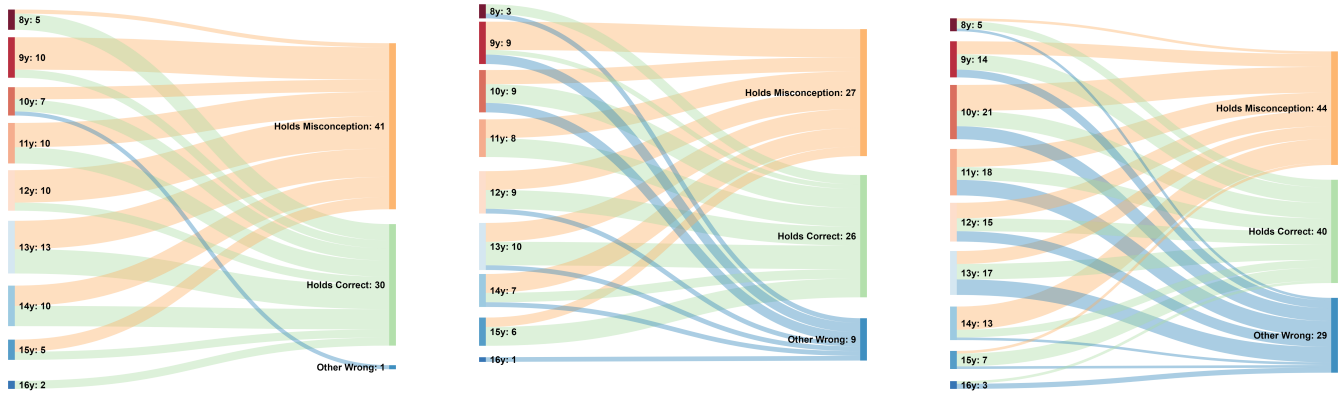
In the alluvial diagrams (See Figure 6) we observe how age groups contribute to the answers for the top three misconceptions. The contribution is represented by the thickness of the flow from source to destination. The diagrams show that children younger than 12 are more likely to hold a misconception than older children. However, the relation is only significant in holding the correct answer as stated above, and not in holding a misconception.

4.3.2 Previous programming knowledge. The reported knowledge of programming languages (Figure 7) shows that almost two thirds of the participants programmed before with Scratch, while the remaining third used programming with a variety of other languages such as Lego, Alice, Python or Javascript. Moreover, the participants reported where they learned programming: at school (62%), at home (28%) or other places such as friends, communities or courses (10%). Since we investigate misconceptions in Scratch, we explore how knowing Scratch in particular compares to other programming languages when it comes to holding a programming misconception. The results show the following:
Knowing Scratch and other languages: is found to decrease the tendency to holding a misconception (Pearson Product-Moment correlation, r=-0.077, p-value=0.035) and increase the tendency to holding correct concepts (Pearson Product-Moment correlation, r=0.151, p-value<0.001).
Knowing other languages: is found to increase the tendency to holding a misconception (Pearson Product-Moment correlation, r=0.071, p-value=0.049).
Knowing Scratch only: knowing only Scratch does not correlate with the holding of a misconception. However, results show that it correlates with answering incorrectly under the *"Other_Wrong"*category (Pearson Product-Moment correlation, r=0.081, p-value=0.025).

5 DISCUSSION

5.1 General observations

While some could argue that it is expected that younger children hold such misconceptions, their origins could be different from



(a) M23 (b) M9 (c) M150

Figure 6: Alluvial diagrams for the top three misconceptions showing the flow of answers from age groups 8y-16y towards the answer categories

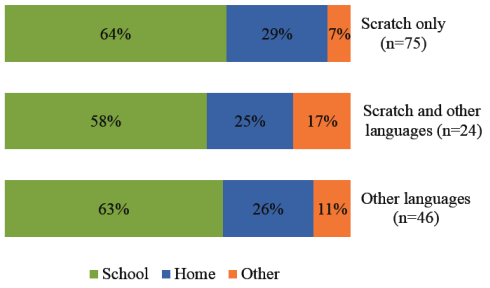


Figure 7: Reported programming languages versus where the child learned them: schools are the primary source of learning programming, while almost one-third of the children indicate home as the learning place for programming.

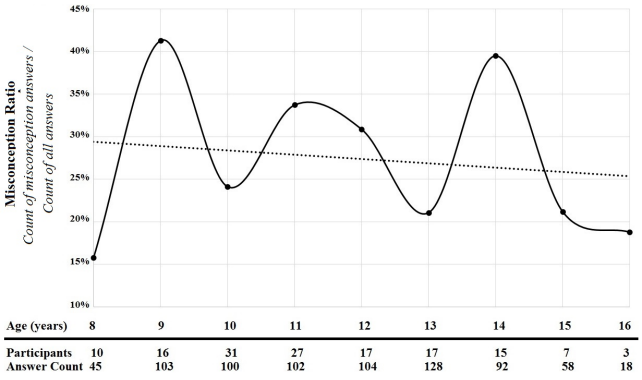


Figure 8: The ratio of misconception answers to the total answers per age (year)

misconceptions in university level students. First, the relation with mathematics seems more pervasive: whenever a question contains numbers, some participants chose to sum the values, even when there is no sum operation in the question (such as M9). The semantic meaning of variables is another recurring aspect that is present in questions other than M17. For example, for misconceptions M33 and M31, since we use the variable name *"counter"*, some participants believe that the program must *"count"* the sequence from one till the end regardless of the loop's condition and the operation inside.

Looking further into the effect of age on misconceptions, we present Figure 8 which shows the misconception ratio for the children per age. The results show that there exists a sudden increase in the holding of a misconception at age=14. The increase is significant compared to the neighboring age groups (13y,15y). Unfortunately, the increase cannot be explained by the currently collected data and need further research.

5.2 Scratch-specific issues

The programming language has its role in eliciting some misconceptions. In previous research this contributed not only to syntactical errors, but also to errors related to the difficulty of understanding the notional machine of this programming language. In our study, we notice that the use of particular blocks in our programming problems caused common and worthy remarks from the participants. First, the *"Say"* block in Scratch caused confusion whenever it was used in the questions. Despite it being a basic and common block, it led to errors when integrated with a variable instead of plain text. Many participants in this case indicated that the program will say the variable's identifier, for example *"X"* or *"total"*, instead of its value. This signifies both a lack of understanding of the variable concept, and the effect of the lingual imperative sense of the verb *"Say"*. A second difficulty we note being related to Scratch is the block used to set a variable to a value: *"Set [variable] to [value]"*. The use of the preposition *"to"* adds ambiguity over the direction of the assignment. Finally, the order of the blocks should be respected because, for some children, the *"visually-attached"* blocks meant that the program will execute in the order from top to bottom even when condition or repeat block exist.

5.3 Reflections on and implications of the results

Misconceptions are considered one area of difficulty in learning how to program. Our results suggest that as we move towards younger children we see more difficulties in answering the questions correctly. This is especially the case for the concepts of variables and control. This result confirms previous research that have shown that children younger than 11 have difficulties understanding those concepts in addition to the concepts of parallelization and procedures [9, 16]. Our results shows, however, some different observations than those in university level students: i) children tend to perform mathematical operations (mostly summations) whenever numbers are present in the exercises, and ii) tend to make assumptions based on their understanding to the semantic names of variables. Consequently, we believe that more diverse exercises should be developed to include operations on strings and booleans, in addition to carefully selecting identifier names in those exercises.

Moreover, our findings suggest that children who indicated programming in Scratch only have difficulties in understanding the concepts correctly and tend to choose other wrong answers. This result highlights two issues. First, educators and researchers are encouraged to identify and realize new misconceptions induced by Scratch as a language. We provided a few observations from our dataset (See Section 5.2), but more research is still needed. Second, our results suggest that younger children start learning programming in Scratch (age positively correlates with the number of programming languages reported in addition to Scratch), and they do it primarily in schools. Those younger *"Scratchers"* seem to have more difficulties to correctly understand the concepts in our study. This result confirms previous research which found Scratch projects to have low percentage of conceptual constructs such as variables, procedures and conditional statements [1, 16, 23]. As a result, we believe that primary education providers are advised to develop more nontrivial and concept-rich materials in Scratch.

Finally, a few participants showed signs of a struggle between contradicting thoughts while reasoning their answers (See Section 4.2). This indicates our belief that holding a misconception is not binary: you either understand the concept or not. On the contrary, it can be a step into grasping the complete concept, and when the confusion is identified it becomes easier to provide the missing piece of information by educators.

5.4 Threats to validity

Like all studies, this paper has some limitations. An external threat to validity comes from the reported experience in programming which the participants provided. In the questionnaire we ask five questions about the previous experience in programming, including questions to identify knowledge of particular concepts. The participants could have still misjudged their own experience and gave false indications. Moreover, a construction threat to validity comes from using multiple choice questions because some children would have guessed the answer despite lacking the knowledge. We eliminated to the minimum those two threats by adding an open-end text following each multiple-choice question, then strictly filtered out any answer for which the participant indicated in the open-end text that they guessed the answer or lacked the understanding of the question or the knowledge to answer it. An internal threat to validity is the design of the question and possible answers for M26. The answers we provided did not include an answer which reflects the misconception: in this case that the program will do nothing. Despite this design issue, 7.2% of 95 participants who answered this question hold the misconception, which was based on the text they provided using the *"Other"* answer option.

6 CONCLUSIONS

Our paper aims at exploring programming misconceptions held by school-age students. The study is based on a multiple-choice questionnaire with programming exercises in Scratch. 145 children participated in the study, aged between 7 and 17 and have some previous experience in programming. The results show that younger learners in school-age indeed hold misconceptions, which caused them to make errors when tracing a small script in Scratch. The top three common misconceptions span over multiple concepts; M23: the difficulty of understanding the sequentiality of statements, M9: the difficulty of understanding that a variable holds one value at a time, and M150: the difficulty to understand the interactive nature of a program when a user input is required. The origins of these misconceptions vary and include the great influence of numbers and mathematical operations in the mindset of the children, the influence semantic meaning of the variable identifier, and the wrongful expectation of what a computer can do, i.e. misunderstanding the notional machine. When analyzing the age effect we found that older children tend to answer the exercises correctly. Moreover, results show that knowing Scratch in addition to at least one other programming language positively influence choosing a correct answer. While children who reported programming in Scratch only had more tendency to choose other wrong answers. Finally, examples we observed in the experiment signify that holding a misconception is indeed a step towards holding the correct and complete concept. Our findings suggest that educators and school teachers should count for the misconception effect as early as possible. Additionally, we should realize new misconceptions induced by Scratch as a language, due to, among other reasons, the use of visually-attached blocks and the use of special keywords in its block set. Finally, Scratch material and lessons should integrate more concept-rich exercises that highlights areas such as variables and control of execution. This is especially needed for children who start learning programming in Scratch. In future work we have two main directions. First we intend to explore and identify Scratch-specific misconceptions. Second, we aim to study in more depth the effect of learning another programming language on Scratch learners. Finally, we aim at developing and testing teaching materials and methods that have less possibility of inducing misconceptions in children.

ACKNOWLEDGMENT

We would like to thank the team of Science Live program at NEMO and all the staff for their support. We also thank the colleagues and student volunteers who took part in running the experiment at the museum.

REFERENCES

[1] Efthimia Aivaloglou and Felienne Hermans. 2016. How Kids Code and How We Know. *Proceedings of the 2016 ACM Conference on International Computing Education Research - ICER '16* (2016). DOI: http://dx.doi.org/10.1145/2960310.2960325

[2] Erik Barendsen, Nataï£¡a Grgurina, and Jos Tolboom. 2016. A New Informatics Curriculum for Secondary Education in The Netherlands. *Informatics in Schools: Improvement of Informatics Knowledge and Perception* (2016), 105–117. DOI: http://dx.doi.org/10.1007/978-3-319-46747-4_9

[3] A. Berglund and R. Lister. 2010. Introductory Programming and the Didactic Triangle. In *Proceedings of the 12th Australasian Conference on Computing Education*. 35–44. http://dl.acm.org/citation.cfm?id=1862219.1862227

[4] B. Du Boulay. 1986. Some Difficulties of Learning to Program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73. DOI: http://dx.doi.org/10.2190/3LFX-9RRF-67T8-UVK9 arXiv:https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

[5] S. Dasgupta and B. Hill. 2017. Learning to Code in Localized Programming Languages. *Proceedings of the 4th ACM Conference on Learning @ Scale* (2017). DOI: http://dx.doi.org/10.1145/3051457.3051464

[6] D. Doukakis, M. Grigoriadou, and G Tsaganou. 2007. Understanding the Programming Variable Concept with Animated Interactive Analogies. *Proceedings of the 8th Hellenic European Research on Computer Mathematics & its Applications* (2007).

[7] P. Fung, M. Brayshaw, B. Du Boulay, and M. Elsom-Cook. 1990. Towards a taxonomy of novices' misconceptions of the Prolog interpreter. *Instructional Science* 19, 4-5 (1990), 311–336. DOI: http://dx.doi.org/10.1007/bf00116443

[8] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. 2008. Identifying important and difficult concepts in introductory computing courses using a delphi process. *ACM SIGCSE Bulletin* 40, 1 (2008), 256. DOI: http://dx.doi.org/10.1145/1352322.1352226

[9] F. Hermans and E. Aivaloglou. 2017. Teaching Software Engineering Principles to K-12 Students: A MOOC on Scratch. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. 13–22. DOI: http://dx.doi.org/10.1109/ICSE-SEET.2017.13

[10] Hai Hong, Jennifer Wang, and Sepehr Hejazi Moghadam. 2016. K-12 Computer Science Education Across the U.S. *Informatics in Schools: Improvement of Informatics Knowledge and Perception* (2016), 142–154. DOI: http://dx.doi.org/10.1007/978-3-319-46747-4_12

[11] E. Kurvinen, N. Hellgren, E. Kaila, M. Laakso, and T. Salakoski. 2016. Programming Misconceptions in an Introductory Level Programming Course Exam. *Proceedings of the ACM Conference on Innovation and Technology in Computer Science Education* (2016). DOI: http://dx.doi.org/10.1145/2899415.2899447

[12] L. Ma. 2007. *Investigating and improving novice programmers mental models of programming concepts*. PhD Thesis. University of Strathclyde, UK. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.444415

[13] L. Ma, J. Ferguson, M. Roper, and M. Wood. 2011. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education* 21, 1 (2011), 57–80. DOI: http://dx.doi.org/10.1080/08993408.2011.554722

[14] Roy D. Pea. 1986. Language-Independent Conceptual ï£¡Bugsï£¡ in Novice Programming. *Journal of Educational Computing Research* 2, 1 (1986), 25–36. DOI: http://dx.doi.org/10.2190/689t-1r2a-x4w4-29j2

[15] R. Putnam, D. Sleeman, J. Baxter, and L. Kuspa. 1986. A Summary of Misconceptions of High School Basic Programmers. *Journal of Educational Computing Research* 2, 4 (1986), 459–472. DOI: http://dx.doi.org/10.2190/fgn9-dj2f-86v8-3fau

[16] Linda Seiter and Brendan Foreman. 2013. Modeling the learning progressions of computational thinking of primary grade students. *Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13* (2013). DOI: http://dx.doi.org/10.1145/2493394.2493403

[17] Simon. 2011. Assignment and sequence. *Proceedings of the 11th International Conference on Computing Education Research* (2011). DOI: http://dx.doi.org/10.1145/2094131.2094134

[18] Simon and S. Snowdon. 2011. Explaining program code. *Proceedings of the 7th International workshop on Computing Education Research* (2011). DOI: http://dx.doi.org/10.1145/2016911.2016931

[19] D. Sleeman, R. Putnam, J. Baxter, and L. Kuspa. 1986. Pascal and High School Students: A Study of Errors. *Journal of Educational Computing Research* 2, 1 (1986), 5–23. DOI: http://dx.doi.org/10.2190/2xpp-ltyh-98nq-bu77

[20] J. Sorva. 2008. The same but different students' understandings of primitive and object variables. *Proceedings of the 8th International Conference on Computing Education Research* (2008). DOI: http://dx.doi.org/10.1145/1595356.1595360

[21] J. Sorva. 2012. Visual program simulation in introductory programming education. PhD Thesis, Aalto University. (2012). http://urn.fi/URN:ISBN:978-952-60-4626-6

[22] Juha Sorva. 2013. Notional machines and introductory programming education. *ACM Transactions on Computing Education* 13, 2 (2013), 1–31. DOI: http://dx.doi.org/10.1145/2483710.2483713

[23] Alaaeddin Swidan, Alexander Serebrenik, and Felienne Hermans. 2017. How do Scratch Programmers Name Variables and Procedures? *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (2017). DOI: http://dx.doi.org/10.1109/scam.2017.12

[24] D. Teague, M. Corney, A. Ahadi, and R. Lister. 2013. A Qualitative Think Aloud Study of the Early Neo-piagetian Stages of Reasoning in Novice Programmers. In *Proceedings of the 15th Australasian Computing Education Conference*. 87–95. http://dl.acm.org/citation.cfm?id=2667199.2667209