Universiteit Leiden
The Netherlands

Opleiding Informatica

Using Program Animation
to resolve misconceptions
in K-12 students.

Stef Dubbeldam

Supervisors:
Felienne Hermans & Anna van der Meulen

BACHELOR THESIS

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

03/07/2019

Abstract

Programming can be difficult to teach to novices. It is not uncommon for students to hold misconceptions regarding one or more concepts in programming. Visualizations are often used to further explain a difficult subject. Programming education for younger students has also become increasingly popular over the past years. We created a tool called CrateCom that automatically transforms a simple Python program into a three-dimensional animation. The goal of this study is to examine whether CrateCom can resolve misconceptions regarding variable assignments in young students. To examine this, we test the students to see if they hold a misconception, after which we let them see CrateCom animate a program. Then we interview the students to measure the effect of the animations. We find that for multiple assignments to the same variable, the animations, without any need for an additional explanation, give the participants a clear idea of the mistake they made and what the correct answer is.

Contents

1 Introduction

Programming education is difficult. There are multiple steps between source code and the resulting output of that code, all of which come with their own intricacies. However, a student is not required to know everything about these steps in order to write a functioning program. It is up to the educator to decide which details will enhance the understanding of the students.
This consideration has become increasingly important since we started teaching the rudiments of programming to younger students. Where programming used to be taught to university students (aged 18+), a growing amount of programming education is now given at younger ages. The programming language Scratch is specifically designed to teach children as young as 8 [scr]. The programming language Python is also frequently used as introductory language, both for older and younger students. One can imagine that younger students are taught less of the technical aspects of programming than older students. As such, the workings behind programming can behave like a black box for the student, meaning the steps between source code and output are often unknown. This means that there is a large gap in knowledge of the students, which increases the probability of the student having a misconception about programming. A misconception is a misunderstanding of how something actually works. For example, a student might believe that a variable can hold multiple values at the same time, or that assigning one variable to another removes the value from the first variable [Bou86].
In this paper, we explore a potential solution to rectify some misconceptions a younger student might hold with regards to simple variable assignment in the Python language. By simple we mean that the right hand side of an assignment statement will only be either an integer value, or another variable. We have created an interactive tool called CrateCom that transforms a given Python program into an animation that illustrates the actions of the computer when a literal or a value of another variable gets assigned to a variable. Upon viewing this animation, students should be able to identify what mistake they made and what actually happens.
To test if CrateCom is successful in rectifying a misconception in younger students, we run an experiment in which the students first get tested on several misconceptions, then get shown an animation linked to a misconception, and then the students are questioned to see if the animation cleared the misconception. 10 students from two different high-schools participated in this study, aged 12-13 years. Due to the small sample size, it is difficult to draw any generalized conclusions. Still, our results show that participating students that held a misconception regarding multiple values in a variable, or a variable keeping its old value instead of the newly assigned value, realize upon interacting with CrateCom what their mistake was and what a better answer would be.

1.1 The misconceptions

Sorva has compiled a long list of misconceptions [Sor12], from which we will gather the misconceptions we want to test. As mentioned, we will only discuss variable assignments. The category of variable assignment, called VarAssign in Sorva's list, has 14 misconceptions. Some of those involve constructs not applicable to the Python language, such as typing problems or empty initial values. Others deal with problems outside our scope, such as the right hand of an assignment being an expression or the precision of a number on a computer. This leaves the following 5 misconceptions that we will examine:

1. A variable can hold multiple values
2. Assignment works in opposite direction
3. Assignment swaps values
4. Assignment moves one value to another
5. The natural language semantics influences the behavior of the variables.

The following three chapters briefly explain each of the misconceptions.

1.1.1 A variable can hold multiple values

We examine the following code:

```
number = 4
number = 5
print(number)
```

Students might incorrectly believe that after line 2, the value `4` is somehow still tied to the variable `number`, whereas in reality it is not. Therefor, the correct outcome of the print will be `5`. However, students holding this misconception might think that the print will output both `4` and `5`.

1.1.2 Assignment does not copy values from right to left

We examine the following code:

```
apples = 3
oranges = 7
oranges = apples
print(oranges)
print(apples)
```

The line `oranges = apples` copies the value from `apples` into `oranges`, overwriting the old value of `oranges`, making the value of both variables `3`. Therefor, the correct outcome of this program is `3, 3`, i.e. two individual prints both printing a `3`. However, students might believe that *assignment works in the opposite direction.* This means that the value of `oranges` is copied into `apples`, which gives the output `7, 7`. Students might also think that *assignment swaps values.* This means that assignment copies values from left to right and from right to left simultaneously, which would output `7, 3`. Another misconception students might hold is that *assignment moves one value to another.* The distinction here is that the value from `apples` is not copied but moved, leaving the variable of `apples` empty. As a result, the output of the last print statement would be undetermined.

1.1.3 The natural language semantics influences the behavior of the variables

We examine the following code

```
biggest = 17
smallest = 43
print(biggest)
print(smallest)
```

Since a computer does not take into account the meaning of the variable names, the output is simply `17, 43`. However, for students it might be difficult to disregard the semantics of variable names. They would see that `17` is smaller than `43`, so clearly the variable `biggest` holds value `43` and variable `smallest` holds value `17`, which in this case would reverse the order of the output: `43, 17`.

1.2 CrateCom

CrateCom creates a 3D environment which signifies a simplified version of the inside of a computer. Figure 1 shows a picture of an animation. We see that all of the constructs are simple shapes with uniform colors, except for the printer, which is a more sophisticated model of a printer. We have deliberately chosen for simplicity over fancy design, trading an engaging animation for ease of understanding. In the upper right corner we have our coding area. At the moment it is limited to seven lines, with 21 characters per line, which is not problematic for this research. In the picture, line three of the code is highlighted in green, to signify that the animation is currently executing line three. Below the coding area are three buttons which can be used to control the animation. The left-most button is used to reset the environment to the start state, the middle button is used for playing the whole animation start to finish, and the right-most button is used to play the animation one line at a time, meaning after each line the animation halts until a button is pressed. Below the buttons we see a value box with the value 4. Literals that are assigned to variables, as is the case on line three in the example code, are all created at this location by scaling up in size. Located in the bottom of the picture are seven memory crates, sitting side by side. In the picture, the left-most crate has the name `var_a` and it contains the value 5 as a result of the first line. The second crate has just been named `var_b`, and it's value is the newly created 4. After creation, value boxes will move sideways to a position right above their memory crate, after which they move down into the crate. If the crate already has an old value in it, the old value moves up out of the create, scale down in size to signify it is being deleted, after which the new value settles in the now empty crate. In front of `var_b` we can see a white arrow. For every line of code, before any boxes start moving or appearing, one or two white arrows will appear and point to the crate or crates that will be involved in the coming animation. During casual showing of the animation before it was test-ready, many people noted it was difficult to see when the name of a memory crate changed. The arrows were added as a crude way to direct the attention to the relevant parts of the animation. In the middle of the picture we see an inverted memory crate, on top of which resides the printer. Variables that get printed have their values copied. This copy moves towards and subsequently into the print-crate. When it has settled into the crate, it get deleted, and as this happens, the value is printed at the top of the printer. In the picture, a `5` is already printed as a result of line 1 and 2. Any subsequent prints will push the printing paper further up, appearing below previous prints, much like an actual printer.

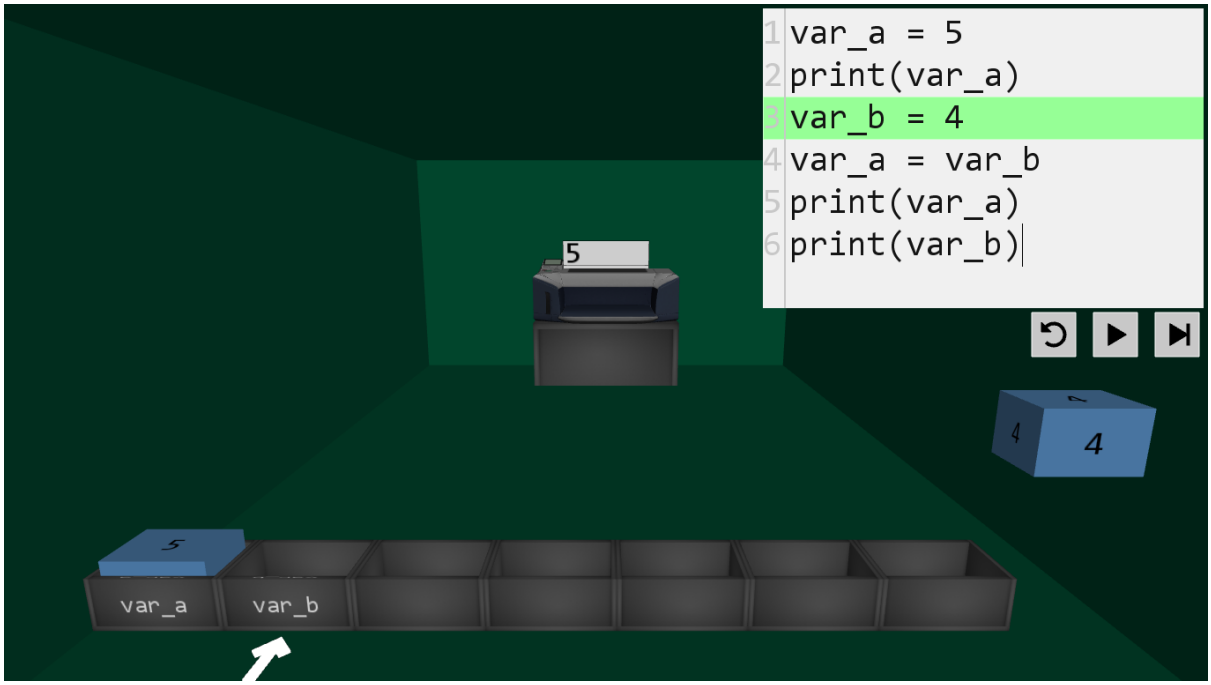

Figure 1: A still of an animation created by CrateCom

CrateCom relies heavily on the box metaphor, where a variable is conceptualized as a box, and the value of that variable is thought of as an object within that box. This metaphor is routinely used in programming education to explain how variables work [HSAS18]. It works because it relates an abstract action, assigning a value to a variable, to a very tangible one, placing an item in a box.

2 Related work

Earlier work shows that providing a novice with a concrete model related to the workings of a program will enhance the understanding of that program [May81]. Furthermore, giving an interactive animation helps novice programmers deal with their misconceptions surrounding variables. [DGT07]. Research also suggests that the mental model of a novice programmer surrounding basic concepts in programming can take many forms, and are not always correct [Sor08] [MFRW11]. When compared to describing a variable as a label that can be placed on one value, students taught with the box metaphor performed worse for the multiple value misconception [HSAS18]. When teaching with the box metaphor, Du Boulay noted that it is important to stress that the box can contain only one item[Bou86]. Other tools have been created with the focus on misconceptions. An example is UUhistle [Sir12] (pronounced 'whistle'). UUhistle is a visualization program where the student acts as the computer and has to create and order actions which a computer would do given a certain input program. UUhistle which was used to collect data on the sources of misconceptions. Analysis of this data shows that while a part of common errors was caused by the UI of UUhistle, many errors were similar to errors reported in earlier literature. Different means of identifying misconceptions in students have been used in previous research[KRPEH10] [SHS18] [Sir12].

3 Experiment

Our research question is:
What effect does program visualization have on misconceptions in k-12 students?
To answer this question, we will look at one specific tool, CrateCom, one specific category of misconceptions, variable assignment, and one specific demographic, 12-13 year old novice programmers. As such, our auxiliary research question is:
What effect does CrateCom have on misconceptions related to variable assignment in novice programmers aged 12-13?
The primary objective of this study is to see whether such a tool has any impact on the students understanding. Students are also asked to give general feedback on CrateCom, which we will discuss briefly and which can be used for further development of CrateCom.
For the purposes of this research, we limit ourselves to the syntax and constructs of the Python programming language. As mentioned before, Python is a common language used to teach to novice programmers. Furthermore, we will only look at integer variables. As the type of variables is a different facet of programming, and Python does not explicitly mention types, any mistakes related to the type of a variable will not be discussed in depth.

3.1 Setup

Before interacting with CrateCom, the students make a test to see which misconceptions they hold. This test consists of two parts. The first part is one exercise where the students are given the following code:

```
age = 4
var_b = age
number2 = 1
print(number2)
varc = number2
print(varc)
```

The students are then asked to underline every variable that they see. This is mainly to give the researcher some idea of the understanding of the students. The next part of the test consists of six pieces of code. These codes contain two types of statements:

- Assigning a literal or the value of another variable to a variable.
- Printing a variable.

An example of such a code is:

```
var_a = 4
var_a = 5
print(var_a)
```

The students are asked to write down what they think the output of the pieces of code will be. These codes do not contain any errors, as that is not the focus of this research. Both of the tests can be found in the appendix.

During the tests, the students are allowed to ask questions to the researcher. However, the researcher only helped when their questions did not relate to any of the misconceptions.
After the tests, the students are introduced to CrateCom. The researcher gives a short verbal explanation about CrateCom, relating the objects are or will appear on the screen to the code. The researcher explains that the upper right area is for code input, that the grey crates at the bottom are where the variables will be, that there will be blue boxes in the screen which will signify the values, and that the object in the middle will print the print statements.
After this explanation, the researcher chooses a test where the student made an error, and enters the corresponding program in to CrateCom, and shows the animation. Lastly, after the animation finishes, the researcher asks the student to compare their answers to what the animation showed, and explain any differences. The impact of CrateCom on misconceptions in the students is to be derived from this explanation. 10 students from two different high-schools aged 12-13 years participated in this study.
The interviews were recorded and analyzed. Anonymized textual excerpts from these interviews will be presented in this paper. As the students were underage, the caretakers of the students had to sign a consent form prior to the study. Both the parents and the students could decide at any point to stop the student from partaking in the study.

4 Results

For the first test, the underlining of the variables, the students consistently made a number of mistakes, and overall performed poorly. As this part of the test was mainly for the researcher to get an idea of the level of understanding of the students, it was not examined any further, and will only be discussed in chapter 5.

For the second test, testing for misconceptions, only the misconception *a variable can hold multiple values* was present among the students. However, five common mistakes were made, including a different misconception not in our original list of five misconceptions. Table 1 shows which misconceptions or mistakes were made and how often those misconceptions or mistakes occurred. The mistakes in this table can be divided into four different categories:

1. Multiple assignments to the same variable.
2. The 'linking' of variables.
3. Type errors.
4. General programming concepts.

For the coming chapters, we will refer to mistakes made when assigning multiple values to a variable as category 1 mistakes, and mistakes made when the type of the variable got confused between integer and string as category 3 mistakes.
After completing the tests, the students watched CrateCom animate one or more of the programs in which they made a mistake. When the animation finished, the students were asked to describe and explain the difference between the outcome of the animation and the answer they gave on the test. For the purposes of this experiment, only errors made in category 1 will be discussed here. The

Misconception	**Number of students**
A variable can hold multiple values.	2
Assignment works in opposite direction.	0
Assignment swaps values.	0
Assignment moves one value to another.	0
The natural language semantics influences the behavior of the variables.	0
Additional mistakes and misconceptions	
When a variable with a value gets a new assignment it keeps its old value instead of getting the new value.	2
Multiple assignments to the same variable concatenates the numbers, creating a new number.	1
A variable can only be printed immediately after assignment.	1
Statements such as `var_a = var_b` result in a string of the variable name being stored.	5
Statements such as `var_a = var_b` 'links' the variables.	5

Table 1: Occurrences of misconceptions and mistakes on the second test

errors from the other category fall outside of the scope of this project, but will be briefly discussed in the next chapter.

Two questions from the tests were related to category 1 mistakes, Question 1:

```
var_a = 4
var_a = 5
print(var_a)
```

And Question 3:

```
var_a = 4
var_b = 3
var_b = var_a
print(var_a)
print(var_b)
```

For mistakes related to category 1, Question 1 was used for examining the influence of CrateCom, and Question 3 was used for further verification of the students understanding.
We discuss the three cases seen in Table 1; one where students thought a variable can hold multiple values, one where students thought the first assignments takes priority over the second, and one where the student thought that the numbers get concatenated and create a new number. Contextual notes or translations are interspersed with square brackets.

4.1 A variable can hold multiple values

Two students made this error. For Question 1, both students answered `4 5`, meaning both a `4` and a `5` would be printed by just one print statement. For Question 3, the first student answered `4, 4 3`, meaning the first print would output `4` and the second print would output `4 3`. The second student answered `4, 4 var_a`, meaning the first print would output `4` and the second print would output `4 var_a`. As we can see the second student also makes a category 3 error, but this will not be further examined here. When asked to explain their reasoning behind their answer on Question 1, both answered very similarly:

'First it was `4`, then it was `5`, so now it is `4 5`.'
'First you make it `4`, then you make it `5`, so now it will be `4 5`.'

After watching the animation for Question 1, both students remarked that the computer 'replaces' the old value with the new value. When asked to reevaluate their answer for Question 3, and in the second students case after explaining their category 3 mistake, both students corrected their answer to `4, 4`

4.2 A variable keeps its old value instead of taking the new one

Two students made this error. For Question 1 both gave `4` as answer, and for Question 3 both gave `4, 3` as answer. However, when asked to explain their answer for Question 1, both students gave a different reason. When we asked the first student why the answer was `4`, the given reason was as follows:

'`var_a` is already `4`, so [the next statement] does not work.'

When we asked the student what would happen with the second line in the program, or in particular the value `5` of the line `var_a = 5`, the student did not know the answer. Then the student watched the animation of Question 1, and as the line `var_a = 5` was animated, the student remarked:

'So the previous goes out [of the box], and the new one gets added [to the box].'
'Added? What do you mean with added?' - Researcher
'The new one replaces the previous one.'

The student was then asked to reevaluate their answer of Question 3. Their new answer was `4, var_a`, meaning the student correctly identified and rectified their category 1 error, but also makes a category 3 error.

The second student gave a different reason for their answer:

'Because it [the computer] will read the first one [the first assignment] and then print that.'

This student is under the impression that when a print statement is executed, the computer will read the program top to bottom, and print the first value it finds for the designated variable. This view is clearly incorrect, but also a relatively sophisticated display of what is called 'the

notational machine' [Sor13] that this student has acquired up to that point. The student ascribes certain actions to the computer that it performs when a certain program is executed.
When we asked this student what would happen with the second line in the program, or in particular the value `5` of the line `var_a = 5`, the student answered with 'It will not be printed'. After we showed the student the animation for Question 1, we asked what mistake they made, and they answered:

'The computer will read the program, and then see that it has been changed, and then print that.'

When asked to reevaluate their answer for Question 3, this student made the exact same adjustment as the student discussed above, and answered `4, var_a`.

4.3 Multiple assignments concatenate the numbers, creating a new value

Only one student made this error. For Question 1, the given answer was `45` (forty-five), and for Question 3, the given answer was `4, 3var_a`. The reasoning behind the answer to Question 1 was as follows:

'`var_a` gets printed, and both [the numbers] are `var_a`, so I think first this one [4] comes, and then that one [5], so then it becomes forty-five.'

We showed this student the animation for Question 1, and asked to explain why the correct answer is `5`:

'Because the new value [overwrites] the old.'

We then showed the student the animation for Question 3, and asked for the explanation of the correct answer `4, 4`. As we can see in the initial answer, this student also made an error of category 3, claiming that the variable name would be stored as a string. However, the explanation did not mention this discrepancy, and instead focused on the multiple-value error:

'Because `var_a` is 4, and `var_b` first is `3`, but then it gets replaced by a `4`.'

4.4 Overall opinion of the students about CrateCom

All of the students were asked to give their opinion on CrateCom. Only one student gave a negative review, claiming it was 'boring'. Interestingly, this student was also the only student who completed both of tests without making any errors. All of the other students were positive about CrateCom, in varying degrees. We noted that as students learned more through CrateCom, they were more positive. One student noted: 'I first did it wrong, and now I understand why I did it wrong'. Another student noted: 'It is handy. It is clearer with the illustrations'.

5 Discussion

5.1 The first test

Overall, the first test of underlining the variables was done poorly. Table 2 shows what mistakes were made and how often those mistakes occurred.
Initially we thought that perhaps the test was poorly designed. Maybe if the question was phrased differently, the students would have performed better. However, the answers of these students follow previous research which showed that the mental models of students of fundamental programming concepts, can take many forms, and are not always correct [Sor08] [MFRW11]. This includes models surrounding assignments and variables. We can infer several mental models from the results of this test.
The first mistake we noticed is that only two students underlined variables in print statements. This suggests that the mental model of a variable is strongly linked to an assignment statement in the students. Secondly, four students thought that a variable is what we call the left hand side of an assignment statement (that in which a value gets stored is the variable), while two students thought that a variable is what we call the right hand side of an assignment statement (the value that is stored is the variable). For example with a program:

```
number = 12
age = number
score = 6
```

The first four students would consistently underline the left hand side, missing the variable `number` on the right hand side in the second line:

```
number = 12
age = number
score = 6
```

The other two students however would consistently underline the right hand side, underlining the literals `12` and `6`:

```
number = 12
age = number
score = 6
```

We suspect that the students who underlined the whole statement either misheard the question, or did not completely understand the question and resorted to underlining everything when they spotted a variable.
As this question was not relevant for the main goal of this research, not much thought was put in the answers of the students at the time of the experiments. However, upon further examination of these results we now recognize that further research into this topic can yield interesting results.

Mistake	Number of students
Underlines only the left hand side of an assignment statement, even when the right hand side is also a variable.	4
Underlines only the right hand side of an assignment statement, even if it is a literal.	2
Underlines the entirety of the assignment statement.	2
Does not understand what 'Underline the variables' means	1
Does not underline variables in print statements.	7

Table 2: Occurrences of mistakes on the first test

Misconception	Number of citations
MC1	5
MC5	4
MC2	3
MC3, MC4	1

Table 3: Number of different papers that cover these misconceptions, according to [Sor12]

5.2 Unencountered misconceptions

We were surprised that of the original misconceptions, only one was present in the participating students. However, if we look at Table 3 where we ranked the five misconceptions from our list on how many different researchers encountered them, as gathered from the references in Sorva's list [Sor12], we can see that the misconception that *a variable can hold multiple values* is indeed the most common one, but not by a lot. These numbers only serve as an indication, as they of course do not reflect the exact amount of papers that cover these misconceptions. If we try and take a closer look at the occurrences of these misconceptions, we encounter another problem. Some papers mention definitive numbers when it comes to occurrences of misconceptions [SHS18], while others instead use words as 'many' or 'some' to describe these numbers [Bou86], so it is difficult to gauge exactly if one is more common than the other.

5.3 Different mistakes

The mistake that variables can only be printed immediately after assignment is a fundamental error in that students' notational machine [Sor13]. It has more to do with the students concept of how a computer executes a program on a basic level, and less with the specific misconceptions related to variables which are the focus of this study.
For the mistake that the name of a variable gets stored for statements like `var_a = var_b`, it could be said that CrateCom offers a good way of examining how students will react when an animation shows the value rather than the name of a variable being stored without any further explanation. However, as we mentioned in chapter 3, type errors fall beyond the scope of this research.

5.3.1 New misconception

One misconception encountered that was not in our original list of five misconceptions was the 'linking' of variables. For example, if we look at Question 6:

```
var_c = 12
var_d = 6
print(var_c)
var_d = var_c
print(var_c)
var_c = 3
print(var_d)
```

Students claimed the last print would output 3, as `var_d` was equated to `var_c`, thereby 'linking' it to `var_c`, and `var_c` was subsequently changed to 3, thereby also changing `var_d`. Of the five students that made this mistake, only three got to see the animation for Question 6. After watching the animation, two of the three students were unable to explain exactly why the correct answer is 12. The third student made an attempt by saying 'because here [pointing at line 5] `var_d` is still twelve.', but when asked to explain more, the student was still confused and was unable to do so. This misconception was noted before by Du Boulay [Bou86], but does not appear directly in Sorva's list [Sor12]. However, this misconception is related memory management and to understanding the sequentiality of statements, the latter of which is noted by Du Boulay and which appears in Sorva's list. Du Boulay goes on to note that pointers can behave like this, and that the thought that variables behave this way is not that odd.

5.4 Differences between classes

The participants came from two different high schools. The two classes were at different stages in their programming education curriculum. Class A had already received a two hour weekly programming lesson for three months, while Class B had received a two hour weekly programming lesson for only three weeks. As such, Class A was already using Python Turtle to draw complicated shapes, while Class B had only just learned about variables. Table 4 shows the mistakes that were made per class. The manner in which Class B learned about variables was through changing a string of the name of an animal and printing it. As such, assigning numbers to variables was new to them, but it was not new to Class A. This can explain why every student taken from Class B made the mistake of storing the name of a variable as a string (see Table 1), while only one of the students from Class A did the same.

5.5 Threats to validity

There are some threats of validity to this experiment. The first one has already been mentioned, which is the small sample size. Another threat to validity is that the creator of CrateCom himself asked the questions to the students. This could result in some of the questions being leading questions, focusing on a desired result from the students, which might influence them to give an answer that is less genuine or authentic than when a student would answer a paper questionnaire for example. The researcher also sometimes mentioned that he made CrateCom himself, which could in turn get the students to alter their answer when asked for their opinion on CrateCom.

Test one	**Class A**	**Class B**
Underlines only the left hand side of an assignment statement, even when the right hand side is also a variable.	3	1
Underlines only the right hand side of an assignment statement, even if it is a literal.	1	1
Underlines the entirety of the assignment statement.	1	1
Does not understand what 'Underline the variables' means	0	1
Does not underline variables in print statements.	4	3
Test two		
A variable can hold multiple values	2	0
When a variable with a value gets a new assignment it keeps its old value instead of getting the new value.	0	2
Multiple assignments to the same variable concatenates the numbers, creating a new number.	0	1
A variable can only be printed immediately after assignment.	0	1
Assigning a variable to another variable results in a string of the variable name being stored.	1	4
Assigning a variable to another variable links the variables.	4	1

Table 4: Occurrences of mistakes on the first test divided by class

6 Future work

Ideally, CrateCom can be used individually and independently by a student to learn the fundamentals of programming. For that to be possible, CrateCom has to be developed further, as it now covers but a small subset of these fundamentals. Furthermore, CrateCom has to be tested more rigorously to determine whether it can actually be used this way successfully and effectively.

6.1 Further Research

Once CrateCom gets developed to a certain standard [Dub], it can be measured against other forms of education to determine if it would be better or not at resolving misconceptions. Also, perhaps some configurations of the 3D world work better than others. Multiple different implementations could be tested against each other for optimal results.
Furthermore, as CrateCom expands, more and more potential solutions to misconceptions can be incorporated in CrateCom and examined for their proficiency.
This study involved students aged 12-13. However, once CrateCom is more sophisticated it can be examined for different age groups to examine if such a tool works better or worse for younger or older students.

6.2 Expanding CrateCom

The most difficult part about creating or expanding CrateCom is to find a decent representation for the concept you are trying to explain. It must convey the process or idea clearly and practically, without obscuring what is happening or unnecessarily complicating it. Before we settled on the subject of variable assignment and our limits in scope, we had multiple ideas we were willing to test.

6.2.1 Expressions

One misconception is that *primitive assignment stores equations or unresolved expressions.*[Sor12], meaning that `x = 5 + 7; print(x);` would output `5 + 7` instead of `12`. To combat this, one can imagine a calculator, which as input gets all the necessary values, and outputs the resolved expression. This calculator could work as black box, meaning it would get all values in no particular order, and output the answer. It could also require the values to be inputted in mathematical order, and show intermediate results.

6.2.2 Lists

Many introductory examples in Python use code snippets containing lists. So far, CrateCom uses the left-right axis to store different variables and the up-down axis to move values between variables. Perhaps the forward-backward axis can be used to implement lists. For example, the closest box shows the name of the list and the element at index 0, and every box behind that contains the element at the corresponding index. Or the first box is empty and shows the name of the list, and all subsequent boxes have as name their index and as value the value at that index.

6.2.3 Boolean Expressions

With regards to integer numbers, Boolean expressions check if some number is greater than (or equal to), lesser than (or equal to), or equal to some number. Perhaps this can be shown be using a balance, and having the state of the balance, meaning left-heavy, right-heavy, or equal, indicate if the Boolean expression holds or not.

6.2.4 General improvements

First and foremost, it has to handle errors. At the moment, it only works on correct programs. However, students will make mistakes as a part of their learning, and these mistakes need to have an adequate animation to show where the mistake was made and how to possibly solve it.
Secondly, more data types have to be supported. As discussed, lists are an option, but Strings and perhaps floating point numbers can also be considered.
A minor point is that CrateCom can now only support programs of seven lines or less. When the scope of CrateCom expands, so should the amount of lines it can process.
As of yet, the control mechanism for the animation is rather simple. In the future this could be extended so that the student can freely skip forward and backward between program states, with the animation updating accordingly.
We kept the design deliberately simple. However, as mentioned above one student noted that it was boring when a long program was being animated. Perhaps the environment can get neater textures, better animation quality, or other enhancements that make the animation more engaging without decreasing the simplicity.
Lastly, the values in the animation do not have a clear mechanism by which they move. We discussed if perhaps a robot arm, or a similar entity, should physically move the values towards their destination. We hypothesize that by giving the computer a more machine-like appearance, it will give the students a stronger sense of what a computer will do when a statement, like a variable assignment, is given, because it is 'just a machine' that executes a given instruction. A possible downside of this could be that it increases the complexity of the animation, possibly making it more difficult to comprehend.

7 Conclusion

Our goal was to find a way to resolve misconceptions surrounding variables in K-12 students. To this end, we created CrateCom, a tool that creates a 3D animation of a basic Python program. With CrateCom we ran a small study with 10 students aged 12-13 from two different high-schools. We tested them for five misconceptions, let them watch CrateCom animate a program for which they held a misconception, and measured their response. We found that after watching an animation the participants understood, without any further explanation by an expert, why a variable can not hold multiple values, or why a second assignment to a variable overrules a previous assignment, or why a second assignment does not concatenate previous assignments. Furthermore, students who learned of their mistakes through the animation thought of it as a positive experience.
We also found that after watching an animation participants could not adequately understand why assigning one variable to another does not link the variables. This is a more sophisticated error, and with further explanation the students understood the sequentiality of programming statements,

but CrateCom itself was unable to convey this idea properly.
We reiterate that the sample size of this study is small, so it is difficult to draw any generalized conclusions. We have outlined future work and research to expand and improve CrateCom, so that in the future CrateCom could be used in programming education.

Appendix A Test 1

```
age = 4
var_b = age
number2 = 1
print(number2)
varc = number2
print(varc)
```

Appendix B Test 2

```
1)
var_a = 4
var_a = 5
print(var_a)

2)
number1 = 2
number2 = 1
print(number1)
print(number2)

3)
var_a = 4
var_b = 3
var_b = var_a
print(var_a)
print(var_b)

4)
big = 2
small = 7651
print(small)
print(big)

5)
last = 4
middle = 5
first = 6
print(first)
print(middle)
print(last)

6)
var_c = 12
var_d = 6
print(var_c)
var_d = var_c
print(var_c)
var_c = var_d
print(var_d)
```

References

[Bou86] Benedict Du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73, 1986.

[DGT07] Dimitrios Doukakis, Maria Grigoriadou, and Grammatiki Tsaganou. Using animated interactive analogies for understanding the programming variable. 2007.

[Dub] Stef Dubbeldam. Cratecom. https://github.com/SirStof/CrateCom.

[HSAS18] Felienne Hermans, Alaaeddin Swidan, Efthimia Aivaloglou, and Marileen Smit. Thinking out of the box: comparing metaphors for variables in programming education. 10 2018.

[KRPEH10] Lisa Kaczmarczyk, Elizabeth R. Petrick, J East, and Geoffrey Herman. Identifying student misconceptions of programming. pages 107–111, 01 2010.

[May81] Richard E. Mayer. The psychology of how novices learn computer programming. 3 1981.

[MFRW11] L. Ma, J. Ferguson, M. Roper, and M. Wood. Investigating and improving the models of programming concepts held by novice programmers. *Computer Science Education*, 21(1):57–80, 2011.

[scr] Scratch, mit media lab. https://scratch.mit.edu/parents/. Accessed: 2019-05-26.

[SHS18] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. Programming misconceptions for school students. pages 151–159, 08 2018.

[Sir12] Teemu Sirki. Recognizing programming misconceptions, 2012.

[Sor08] Juha Sorva. The same but different students' understandings of primitive and object variables. In *Proceedings of the 8th International Conference on Computing Education Research*, Koli '08, pages 5–15, New York, NY, USA, 2008. ACM.

[Sor12] Juha Sorva. Visual program simulation in introductory programming education, 2012.

[Sor13] Juha Sorva. Notional machines and introductory programming education. *Trans. Comput. Educ.*, 13(2):8:1–8:31, July 2013.