Variable Evaluation: an Exploration of Novice Programmers' Understanding and Common Misconceptions

Tobias Kohn
ETH Zurich
Universitätstrasse 6
CH-8092 Zurich
kohnt@inf.ethz.ch

ABSTRACT

For novice programmers one of the most problematic concepts is variable assignment and evaluation. Several questions emerge in the mind of the beginner, such as what does $x = 7 + 4$ or $x = x + 1$ really mean? For instance, many students initially think that such statements store the entire calculation in variable x, evaluating the result lazily when actually needed. The common increment pattern $x = x + 1$ is even believed to be outright impossible.

This paper discusses a multi-year project examining how high school students think of assignments and variables. In particular, where does the misconception of storing entire calculations come from? Can we explain the students' thinking and help them develop correct models of how programming works?

It is particularly striking that a model of the computer as a machine with algebraic capabilities would indeed produce the observed misconceptions. The misconception might simply be attributed to the expectation that the computer performs computations the exact same way students are taught to in mathematics.

Keywords

Programming; misconceptions; variables; novices; learning

1. INTRODUCTION

Computer science is not only an important field in the sciences but also offers some unique and highly relevant contributions to general education. Accordingly, there are numerous efforts to establish computer science in K-12 education and make the merits of algorithmic thinking available to the general population.

Programming is a fundamental part of CS education. To be successful in teaching programming, however, it is imperative that we study the mistakes and comprehension of our students. With a good understanding of where the difficulties lie we can much better guide our students to success.

A particularly difficult topic in learning to program are variables and assignments. First reports date back more than thirty years (see, e. g., [3]). Numerous studies followed, most of which concentrated on students at university level (see the section about related work). It is highly possible, though, that the difficulties with variables and assignments are due to some deeper misconception about how a computer program is actually being executed. There might also be some differences between students at high school and university level.

In this paper we present some misconceptions of high school students in connection with variables, assignments, and the time of evaluation. For this study, we analyzed students' tests, collected during four years of teaching Python programming in high school. In our setting, the programming course is part of classes in advanced and applied mathematics. In addition to programming we also teach some math classes. This allows us to directly compare the students' work in mathematics and programming, even though a student rarely has the same teacher for both classes.

1.1 Mathematical Techniques in Programming

During classroom sessions some students distinctly applied techniques and methods from mathematics to programming. This might lead to incorrect results. Some students, for instance, started their programs to solve quadratic equations with the following two lines:

```
x = unknown
quEq = a * x**2 + b * x + c = 0
```

In informal interviews, students explained that they have to tell the computer first what a quadratic equation is before they can provide the actual solutions. Based on such instances of misapplication we looked for further examples and evidence that some misconceptions in programming might be due to mathematical practices applied in the wrong places.

Based on the collected information we assume that students apply a model of mathematical substitution to programming, leading to the observed mistakes. More precisely, when students perform calculations in mathematics they frequently substitute parameters and variables, making use of previously established relationships (e. g., knowing that $y = 2x$, we can replace y in subsequent calculations by $2x$. This works even in the case where we do not know the value of x). Roughly a third of our students in program-

SIGCSE '17, March 08 - 11, 2017, Seattle, WA, USA

ISBN 978-1-4503-4698-6/17/03…$15.00

DOI: http://dx.doi.org/10.1145/3017680.3017724

ming class applied the same model of variable substitution to programming at one point or another.

Python programs, however, are executed with a different model. The main difference is that in Python assignments are evaluated immediately, assigning a concrete value to a variable. Variables are not mere symbols as in mathematics but represent a concrete object.

1.2 Organization of the Paper

The rest of this paper is organized as follows. Section 2 gives an overview of related work with some comments on where our work differs from others. Section 3 presents the methodology used in this work. In Section 4, we present the students' answers we collected from four problem sets. Of particular high value is a question where students were asked to trace and determine the output of a program and explain their reasoning for reaching their respective answers. The discussion of these results together with our interpretation is then found in Section 5.

2. OVERVIEW AND RELATED WORK

The literature about misconceptions of novice programmers is quite extensive. With a focus on variables and assignments, however, we list some noteworthy and common misconceptions in the following three groups:

Syntax. Some students believe that assignments are symmetric, i. e. $x = 2$ and $2 = x$ are both valid. Others believe that a variable's name determines its contents, e. g., that a variable called "max" will automatically hold the maximum value of a list.

The Nature of Variables. Variables are sometimes seen as boxes that can hold more than one value or have a memory. Some students also think that after a variable has been accessed, it is "empty" afterwards.

An interesting misconception is that variables hold unevaluated expressions such as $2 + 3$ or $x + 1$. The actual value would then be computed when the variable's value is accessed. This notion is discussed in detail in this paper.

Assignments. Due to the syntactical similarity with equations, some students believe that assignments are actual equations to be solved by the computer. Others believe that an assignment such as $x = y$ links the two variables together, so that x and y will always hold the same value.

This last misconception of $x = y$ linking x and y together becomes much more relevant when references are introduced. For instance, in Python x and y might indeed be linked together if y is a list.

Interestingly, some students believe that the entire "equation" is stored. This misconception is very similar to the notion that a variable holds the entire unevaluated expression. The difference between this misconception and the idea that a variable holds an unevaluated expression might lie in that some students do not think in terms of variables at all but assume that the computer stores equations independently of variables.

2.1 Studies about Misconceptions

A study by Du Boulay includes a rather comprehensive discussion about inappropriate analogies for assignment and variables [3]. According to Du Boulay, some problems arise due to the syntax. The use of the equal sign '=' suggests a symmetry so that it would be possible to write either $x = 2$ or $2 = x$. In addition, the similarity with equations is problematic, leading to the believe that $x = y$ links the two variables together.

Other problems with variables are a result of the use of the "box"-metaphor, leading to the belief that a variable can hold more than one value. Of particular interest is the misconception about assignments such as $x = 7 + 4$: "*A further misunderstanding concerns the assignment of values derived from an expression LET A = 7+4. A student may understand 'A' to hold 7+4 as an unevaluated expression rather than 11. In some [BASIC dialects] 'A' is only allowed to hold numbers, so this misunderstanding can be tackled by stressing the idea that a variable can hold only one number.*" [3]

We believe, however, that in the view of students the notion of a variable "holding only one number" is not necessarily in conflict with the variable standing for the unevaluated expression $7 + 4$. From a mathematical perspective there is no contradiction between saying that x stands for a single number and writing $x = 7 + 4$, because $7 + 4$ eventually evaluates to a single number. Hence, given the mathematical training of typical programming students, stressing that a variable can only hold one number might not be enough to avoid misconceptions.

Bayman and Mayer studied the misconceptions of novice Basic programmers [1]. They note that students get confused about the equal sign in assignments, seeing it as an equation rather than as an assignment. They also note that "*a major misconception is to think that the equation is stored in memory.*"

For example, their subjects were asked to explain the statement "LET A = B + 1". 43 % answered that the statement would write the equation into memory and 33 % said that it would write $B + 1$ to memory space A (with 30 % giving the correct answer of writing the *obtained value* from $B + 1$ to memory space A). We find it noteworthy that a third of the students understood the basic nature of an assignment (and that it is not an equation) and still believed that the entire expression $B+1$ would be stored in memory. This corresponds quite well with our own findings.

Similarly, Putnam et al. [12] found two errors related to the assignment, including the assumed symmetry mentioned above. One student had a "*deeper misconception involving the use of an assignment statement as a counter. He declared that the statement LET C = C + 1 was impossible. He had previously assigned the value of 0 to C and interpreted the statement as 'LET 0 equal 0+1.'*" [12]

Samurçay notes that programming students start with an "*initial but insufficient model*" of variables derived from mathematics and that "*the programming variable is a new concept for the student*" [13].

According to Samurçay, particularly in the context of loops the mathematical model of a variable is inappropriate because "*the mathematical description of a variable is a static one.*" The focus of Samurçay's study, however, is on the difficulties of different tasks and roles of variables. More recent studies then propose to explicitly teach the roles of variables to enhance the students' understanding [2, 6, 11].

2.2 Recent Studies about Difficulties

Some recent studies about the difficulties of novice programmers used surveys to determine the most difficult topics for programming novices, e. g. [4, 7]. In contrast to the various misconceptions about variables, the studies found

that variables rank among the least difficult topics, together with selection and loop structures. The studies, however, did not provide a means to detect students' misconception. Hence, students might just not be aware of their difficulties.

Another comprehensive study by Lister at al. [8] analyzed students' reading and tracing skills. They found that the students' knowledge in programming is often fragile, but they also noted that they "*see few comprehension errors due to misconceptions.*" [8] It is interesting to note that tracing on paper and thereby documenting changes in variables increases the likelihood of getting the correct answer. However, as the assignments used in the study's programs contained little dependencies on other variables, it is hard to draw firm conclusions about the misconceptions concerning the evaluation of variables.

3. METHODOLOGY

As part of a four-semester course in applied and advanced mathematics we teach introduction to programming for two semesters with two hours each week. Students with advanced placement in math and the sciences are required to take this class during 10th grade. The class has been taught four times with about ten students each year using Python.

The syntax of Python turned out to pose virtually no problems for the students, confirming that Python is a suitable language for programming at high school [4].

The comparatively low number of students per class allows us to closely observe the students' progress and conduct informal interviews in case of problems and misunderstandings. The entire class emphasizes hands-on with little instruction from the teachers.

During the four-year period we kept a journal of students' problems, questions and noteworthy ideas. This journal was the initial source and motivation for identifying common misconceptions.

We then looked at the tests written by students in those four years in order to find further examples and assess how many students exhibit the patterns of misconception. Each year the students were required to write five to six tests with paper and pencil. During the tests students had no access to a computer. Previously the tests were written using computers but especially weaker students were not able to complete the tests in time as they spent most of their time correcting minor syntactical issues. The questions mostly varied over the years so that not all students answered all questions used in this study.

From the tests we have chosen four problems to include in our study (where N is the number of students who provided a solution):

[P1] Write a program to solve the quadratic equation $ax^2 + bx + c = 0$. Make sure you handle special cases such as when no solutions exist. ($N = 20$)

[P2] Write a program to find all Pythagorean triples $a^2 + b^2 = c^2$ with $a, b, c < 100$. ($N = 24$)

[P3] Write a program to draw the graph of the function $f(x) = x(x-5)(x+5)/25$ in the range between -40 and 40. ($N = 20$)

[P4] Trace the given program, write down the output and explain your solution. ($N = 10$)

Ethical review. This study was conducted on the answers students had given during their regular tests. None of the questions were actually designed for study purposes. In addition, answers were only used in anonymized form, and no personal information has been shared with any third party. Since no student has been affected by this study in any way, Switzerland requires no review by the ethics commission.

4. RESULTS

Based on classroom experience we assumed that some students have the misconception that a variable's value is not evaluated during assignment, but that the evaluation is rather deferred until the variable is actually accessed or used. Several studies reported similar misconceptions, e. g., [1, 3] (see section about related work).

From the tests conducted in our programming class we then selected four problems and examined the student's answers. Our goal was to find concrete examples where the students might believe that the evaluation of a variable's value is deferred until the variable is read. We also wanted to find out how many of the students might have this misconception. The results of this examination are presented in this section.

4.1 Testing for Computability after the Computation

The formula for the solution of a quadratic equation does not necessarily yield real results. If the discriminant is negative no real solutions exist and Python throws an exception "math domain error". In problem [P1], students were therefore asked to write a condition and test for a negative discriminant prior to calculating its root.

Since all students had recent experience with the quadratic formula from math class we expected little problems with this question. Students also learn in math class that some quadratic equations have no solutions, and that this can be recognized by a negative discriminant. Complex numbers (and solutions) are only discussed later in the curriculum.

Instead of the correct solution we found some students testing whether the root of the discriminant is non-negative (i. e., $\sqrt{D} \geq 0$ instead of $D \geq 0$) and some testing for the coefficient a to be non-zero in order to avoid a division by zero. For our study about programming misconceptions, however, we were interested in those who calculated the solution before testing whether the solutions could be calculated at all.

Performing a test for computability after the actual computation does not make sense, and students should certainly be aware of that. Yet, this order can be explained if a student is not aware that the assignment does perform the computation, but rather assumes a model of deferred evaluation.

The typical pattern of calculating the solution prior to testing for problems is exhibited in the following, slightly simplified, program of a student (in reality, students were asked to retrieve both solutions and write a complete program):

```
x = (-b + sqrt(b**2 - 4*a*c))/(2*a)
if b**2 - 4*a*c >= 0:
    print "The solution is", x
```

We included the problem of solving a quadratic equation in the tests of two distinct classes (ten students each). In the first class, four students exhibited the above pattern of

calculation before testing and one student did not perform any testing in his code at all.

In the second class, after a more thorough discussion in class of the problem, no student made this particular mistake (but not all answers were correct).

4.2 Dependence on other Variables

In the context of a looping structure, deferred evaluation of a variable's value might even lead to the variable changing its value with each iteration. An assignment such as y = 2∗x with a dependency on a second variable x is then believed to automatically reflect changes to x. In essence, y becomes an implicit function with the parameter x. We found evidence for this pattern in two problems.

The first problem [P2] asked students to write a program that finds all Pythagorean triples (a, b, c) with $a^2 + b^2 = c^2$ and $a, b, c < 100$. Most students solved this problem straightforward with nested loops and a test such as

```
if a**2 + b**2 == c**2:
```

Some students calculated $c = \sqrt{a^2 + b^2}$ and tested whether it was an integer using **if** c % 1 == 0.

In total, 24 students answered this question. Three students used the pattern of deferred and thereby repeated evaluation. A typical example of such a student's solution looks as follows:

```
c = math.sqrt(a**2 + b**2)
for a in range(1, 100):
    for b in range(1, 100):
        if c % 1 == 0:
            print a, b, c
```

The second problem [P3] asked students to draw the graph of a given (mathematical) function as a sequence of line segments. Due to the mathematical function (in our case $f(x) = x \cdot (x-5) \cdot (x+5)/25$) and previous classroom experience we expected this question to be especially prone to the pattern of deferred evaluation. A typical program of a student would then look like this:

```
x = -40
y = x *(x-5)*(x+5) / 25
setpos(x, y)
for i in range(80):
    x += 1
    lineto(x, y)
```

Of the 30 students who had this question in their tests, ten did not provide a viable solution (they either did not answer the question at all or they tried to draw the graph by calculating the necessary values by hand first). Six of the remaining 20 students showed the pattern of deferred evaluation.

Some students had both problems [P2] and [P3] in their tests. However, surprisingly, no student showed the pattern of deferred evaluation in both questions.

4.3 Tracing Values

One class of ten students was asked to determine the output of the following Python program in problem [P4]. In addition to giving simple numbers as output, we asked the students to explain their solutions.

```
x = 5
def f(x):
    return x*x

g = x*x
x = 8
print f(2)
print f(x)
print g
```

The correct answer is that the program prints the numbers 4, 64 and 25. Of the ten students three came up with the correct answer. Another three students said that the last **print**-statement would also output 64. The remaining four students had completely different answers. The three students who thought that the last number would be 64 all reasoned that g equals x∗x and that at the point of printing x holds a value of 8. They described a two-stage process where g is first replaced by x∗x and, subsequently, the x is replaced by 8.

Two of the remaining students reasoned that the **print** f(x)-statement would not work at all. One wrote that the x would have to be replaced by some concrete numeric value for the function to be evaluated (instead of just leaving the abstract parameter). The other wrote that f(x) could not be evaluated since x had two possible values (5 and 8).

Another student claimed that the output would be three times 64 because the x = 5 would not make any sense at all as it was in the wrong place. The last student believed that the output of **print** f(x) would also be 25, hence ignoring the assignment x = 8.

In other words, three of the ten students answering this question believed that the value of g would not be evaluated until it was actually printed. Another three students said that the given code violated some rule regarding variables.

4.4 Summary

The following table summarizes the frequency of the pattern of deferred evaluation in the four problems studied. For each problem we give the number of students exhibiting the pattern as well as the total number of students who provided a solution.

[P1] Quadratic equation	4	20
[P2] Pythagorean triples	3	24
[P3] Graph of a function	6	20
[P4] Tracing values	3	10

5. DISCUSSION

With a total number of only about 40 students our figures are not sufficient to give a reliable statistical result. In addition, problem [P2] of finding Pythagorean triples is probably a bad example for statistical analysis because most students saw no need for using an assignment that might show the pattern, anyways. In case of problem [P3] of drawing the graph of a function, some mistakes might be due to a simple oversight instead of an underlying misconception. However, the figures show that the pattern of deferred evaluation occurs frequently enough to warrant a discussion.

Of particular importance are the answers to problem [P4]. They show that, indeed, some students' mental models of the evaluation and assignment of variables are wrong. Roughly a third of the students seems to have assumed that an assignment is evaluated lazily at the time when the variable's value is actually used.

5.1 Interpretation of the Results

From a programmer's perspective students conceptually use two different data types for their variables. The first type represents numbers (we simplify here and do not distinguish between integers and floating point numbers). The second type represents calculations and is similar to functions. Indications for the distinction between these two types can be found in two places. Students who thought that g would be evaluated to x∗x first and then to 8∗8 had no problems with the two assignments x=5 and x=8 in [P4]. In addition, our students showed no discernible effort in understanding and using the variable x in problem [P3] when drawing the graph of $f(x)$.

Another interpretation is that students do not primarily think in terms of variables and their values as a programmer would. The assignment g=x∗x is not regarded as an instruction to evaluate x∗x and store the result as the value of a variable g. Rather, the assignment establishes a relationship that is later used to replace the symbol g in calculations.

The notion of using relationships such as $g = x \cdot x$ to replace g in subsequent calculations is exactly what we teach students in math class. The pattern of deferred evaluation actually corresponds quite directly to the way students solve mathematical problems and perform calculations on their own. We hence surmise that they expect the computer to mimic the same methods they use. Our findings suggest that students attribute algebraic capabilities to the executing machine. Students obviously fail to see that the notional machine in imperative programming is a purely numerical machine with no knowledge of algebraic relationships.

5.2 The Equation $x = x + 1$

In connection with the difficulties of variables and assignment we often find the example of $x = x + 1$. For instance, Du Boulay notes that "*Beginners are often puzzled by such assignments [. . .] precisely because they have not understood the asymmetry and the sequential nature of the execution of even this single assignment. The 'x' on each side of the '=' sign are not treated in the same way. One stands for a location and the other for a value.*" [3] This passage implies that the students' difficulties arise because they assume the computer replaces the x's on both sides of the equation by the current value of x, leading to an equation with numbers only and no variables. Yet, according to our model the assignment is still confusing even when students understand the different nature of the variables on the left and the right hand side, respectively.

From a strict mathematical point of view, $x = x + 1$ is an unsolvable equation and thus does not make sense for a novice programmer. But even when students do understand that this is an assignment and not an equation, it still does not make sense in a "mathematical" model of substitution where variables can be replaced by their assigned expression. In this case, we get an infinite expansion of x leading to $x+1$ leading to $x + 1 + 1$ etc.

In order to avoid the problem of $x = x + 1$ altogether we consequently used the statement x += 1 instead. At first students seemed to work well with this alternative notation. However, closer inspection revealed that many either dropped the equal sign and simply wrote x + 1 as a statement to increment the value of x. Or they attempted to expand the statement in invalid ways, e. g. (x += 1) ∗ 3.

5.3 The Notional Machine

Du Boulay notes that learning to program does not only include mastery of the notation (syntax and semantics) of a given programming language [3]. Rather, the learning process also includes mastery of the "*notional machine*" and acquiring standard structures or plans.

The notional machine is an abstract concept describing how a program is executed, and differs from the actual implementation. When reasoning about objects, for instance, we usually see them as an inherent feature of the system like Python or Java; we see them as a feature of the *notional machine*. How the objects themselves are implemented is not important and might even differ, depending on the underlying system.

The observed difficulties with variables and assignments can indeed be attributed to a misconception about the notional machine executing the programs. Students seem to believe that the machine is capable of algebraic manipulation of the expressions used in the program: during evaluation a variable would be replaced by its defining expression instead of a previously determined numeric value.

In other words: students seem to expect the computer to perform computations the same way they have been trained in mathematics. The substitution of variables by expression is a very common operation in mathematics. However, variables are then only thought of as symbols representing something else, while in programming variables have a much narrower meaning.

5.4 Proposed Solutions

To address the students' misconceptions and provide them with a suitable model for the notional machine we equipped our Python environment with a visualizing debugger. The basic design followed Guo's "Online Python Tutor" [5]. Initial experience shows that the visualizer is a very helpful tool for teaching and explaining certain concepts about the notional machine. We also found that our students made slightly fewer mistakes that could be attributed to a misconception about variable evaluation.

However, most students were rather reluctant in using the visualizer on their own and according to a short survey they did not find it particularly helpful. Ma et al. [9] also note that "*Visualization techniques have been used for over 20 years and have, arguably, not been as successful as hoped for.*" The observed improvement might therefore be due to other reasons.

More promising results were achieved in other classes (not part of this study), where students explicitly learned to trace programs by hand and keep tally of the variable's values. This is also in accordance with the findings of Lister et al. [8] that documenting the changes in variables helps students correctly predict a program's output. It should be noted, however, that manual tracing of a program might not suffice. Even in the classes which had learnt to manually trace programs, some students did not have or use a correct model of the notional machine.

It is interesting to note that a study about educational videos found that, in order to be effective as learning tools, the videos must first confront the learner with possible misconceptions [10]. The study compared learning outcomes of physics students after being exposed to different videos. The study indicates that students who watched the videos without a discussion of common misconceptions seem unaware of

the discrepancies between their own mental model and the presented models. Hence, "*misconception treatments should activate students' prior knowledge and help them recognize any disparity between their ideas and correct scientific theories.*" [10]

Applied to our case, it might simply not be enough to use visualizations and show students how the notional machine works. In order to build viable mental models, students must be prompted to interactively work with the notional machine and recognize differences from their current mental models. A possible way to achieve this interaction could be to teach students how to trace a program with paper and pencil, and check the obtained results.

Future research will have to determine both how frequent these misconceptions are on a wider base, and how effective different teaching methods turn out to be in addressing these misconceptions.

6. CONCLUSIONS

Teaching and learning programming clearly is a complex task. Many studies report that students' knowledge of programming is fragile [8] or that they still hold improper models of relatively simple concepts [9]. Some studies about students' difficulties specifically note that variables and assignment are a source of misconceptions [1, 3, 9, 13, 12].

Our work confirms that variables and assignment can be problematic for novice high school programmers. We found a pattern of deferred evaluation of variables in the programs of about a third of our students. As a possible explanation we pointed out that the students' mistakes are consistent with a model of the notional machine exhibiting algebraic capabilities.

In order to address such misconceptions care must be taken that programming students are fostered in developing a correct and viable model of the notional machine. A first step towards this goal is to help students identify discrepancies between their own beliefs and correct models. We propose to use interactive visualizations and to explicitly teach students how to trace a program by hand.

7. REFERENCES

[1] P. Bayman and R. E. Mayer. A diagnosis of beginning programmers' misconceptions of basic programming statements. *Commun. ACM*, 26(9):677–679, Sept. 1983.

[2] P. Byckling and J. Sajaniemi. Roles of variables and programming skills improvement. *SIGCSE Bull.*, 38(1):413–417, Mar. 2006.

[3] B. Du Boulay. Some difficulties of learning to program. *Journal of Educational Computing Research*, 2:57–73, 1986.

[4] L. Grandell, M. Peltomäki, R.-J. Back, and T. Salakoski. Why complicate things?: Introducing programming in high school using python. In *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, ACE '06, pages 71–80, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[5] P. J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[6] M. Kuittinen and J. Sajaniemi. Teaching roles of variables in elementary programming courses. *SIGCSE Bull.*, 36(3):57–61, June 2004.

[7] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen. A study of the difficulties of novice programmers. *ITiCSE '05 Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 14–18, 2005.

[8] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *SIGCSE Bull.*, 36(4):119–150, June 2004.

[9] L. Ma, J. Ferguson, M. Roper, and M. Wood. Investigating the viability of mental models held by novice programmers. *SIGCSE Bull.*, 39(1):499–503, Mar. 2007.

[10] D. Muller, J. Bewes, M. Sharma, and P. Reimann. Saying the wrong thing: improving learning with multimedia by including misconceptions. *Journal of Computer Assisted Learning*, 24(2):144–155, 2008.

[11] U. Nikula, J. Sajaniemi, M. Tedre, and S. Wray. Python and roles of variables in introductory programming: Experiences from three educational institutions. *JITE*, 6:199–214, 2007.

[12] 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*, 2(4):459–472, 1986.

[13] R. Samurçay. The concept of variable in programming–its meaning and use in problem-solving. *Educational Studies in Mathematics*, 16(2):143–161, 1985.