Variables in Practice. An Observation of Teaching Variables in Introductory Programming MOOCs

Van Der Werf, Vivian; Zhang, Min Yi; Aivaloglou, Efthimia; Hermans, Felienne; Specht, Marcus

DOI
10.1145/3587102.3588857

Publication date
2023

Document Version
Final published version

Published in
ITiCSE 2023 - Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education

Citation (APA)
Van Der Werf, V., Zhang, M. Y., Aivaloglou, E., Hermans, F., & Specht, M. (2023). Variables in Practice. An Observation of Teaching Variables in Introductory Programming MOOCs. In *ITiCSE 2023 - Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education* (pp. 208-214). (Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE; Vol. 1). Association for Computing Machinery (ACM). https://doi.org/10.1145/3587102.3588857

Variables in Practice. An Observation of Teaching Variables in Introductory Programming MOOCs

Vivian Van Der Werf
Leiden University
Leiden, The Netherlands
Delft University of Technology
Delft, The Netherlands
v.van.der.werf@liacs.leidenuniv.nl

Min Yi Zhang
Leiden University
Leiden, The Netherlands
m.y.zhang@umail.leidenuniv.nl

Efthimia Aivaloglou
Delft University of Technology
Delft, The Netherlands
e.aivaloglou@tudelft.nl

Felienne Hermans
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
f.f.j.hermans@vu.nl

Marcus Specht
Delft University of Technology
Delft, The Netherlands
m.m.specht@tudelft.nl

ABSTRACT

Motivation. Many people interested in learning a programming language choose online courses to develop their skills. The concept of variables is one of the most foundational ones to learn, but can be hard to grasp for novices. Variables are researched, but to our knowledge, few empirical observations on how the concept is taught in practice exist. **Objective.** We investigate how the concept of variables, and the respective naming practices, are taught in introductory Massive Open Online Courses (MOOCs) teaching programming languages. **Methods.** We gathered qualitative data related to variables and their naming from 17 MOOCs. Collected data include connections to other programming concepts, formal definitions, used analogies, and presented names. **Results.** We found that variables are often taught in close connection to data types, expressions, and program execution and are often explained using the 'variable as a box' analogy. The latter finding represents a stronger focus on 'storing values', than on naming, memory, and flexibility. Furthermore, MOOCs are inconsistent when teaching naming practices. **Conclusions.** We recommend teachers and researchers to pay deliberate attention to the definitions and analogies used to explain the concept of variables as well as to naming practices, and in particular to variable name meaning.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Applied computing** → *E-learning*.

KEYWORDS

programming education, variables, naming practices, analogies

ACM Reference Format:
Vivian Van Der Werf, Min Yi Zhang, Efthimia Aivaloglou, Felienne Hermans, and Marcus Specht. 2023. Variables in Practice. An Observation of Teaching Variables in Introductory Programming MOOCs. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023), July 8–12, 2023, Turku, Finland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587102.3588857

ITiCSE 2023, July 8–12, 2023, Turku, Finland

ACM ISBN 979-8-4007-0138-2/23/07.
https://doi.org/10.1145/3587102.3588857

1 INTRODUCTION

Variables are a hard concept to grasp for novice programmers [13, 18, 21]. At the same time, variables are important for reading and understanding code, which are both core skills for a proficient programmer [10, 25, 26, 28, 29]. Variables are at the core of many programs, as they are able to store and retrieve data from memory ever since the introduction of "variable cards" in *The Analytical Engine*, which was designed in the 1830s and laid the foundation for modern-day programming. Many programming concepts expand on the concept of variables, for example, loops, functions, and control flow, and it is essential that variables are well understood by novice programmers. Furthermore, naming is an important aspect of variables, especially concerning the act of reading and understanding code. It is commonly accepted that meaningful identifier names help readers understand code more easily than when abbreviations or (random) letters are used [2, 23], although it has also been found that full names can be misleading if they do not correctly represent their contents or purpose [1, 7].

Little empirical research has investigated *how variables are taught*, hence we are interested in conducting an observational study to gain insights into current teaching practices regarding variables. Since online platforms such as *edX* and *Coursera* grow increasingly popular [22], we use Massive Open Online Courses (MOOCs) as a case study for our observation study. Our research questions are:

(1) **How is the concept of variables taught in introductory programming MOOCs?** We investigate (a) the connection to other programming concepts when variables are introduced, (b) how variables are defined, and (c) what analogies are used to explain variables.
(2) **How are variable naming practices taught in introductory programming MOOCs?** We examine naming practices that (a) are taught explicitly, and (b) are used by the instructors, and therefore taught implicitly.

During our analysis, we found that variables are often taught in close connection to data types, expressions, and program execution,

and are often explained using the 'variable as a box' analogy. This represents a stronger focus on 'storing values' than on naming, computer memory, and the flexibility gained by using variables. Furthermore, we found inconsistencies in the taught naming practices – if they were taught at all. We recommend teachers and researchers to pay deliberate attention to naming practices, specifically to meaning, and to the definitions and analogies used to explain the concept.

2 RELATED WORK

Analogies are used to explain programming concepts [12]. In education, an analogy, metaphor or notional machine is a 'tool' that supports learning by simplifying a concept through a representation that highlights the most important aspects of the concept, while obscuring less important aspects [12]. For example, 'variables as parking spaces' simplifies the concept of variables, transferring our knowledge about parking spaces to our comprehension of a variable. Waguespack [35] explains a variable of a particular data type as a 'container with the corresponding shape' (*shape* refers to the data type). In this explanation, a container can hold only a single value. This assumption is important since a common misconception is that variables can hold multiple values at the same time [6, 9, 18]. Any analogy might only partly or incorrectly represent a concept and thus can leave novice students with an incorrect understanding or misconception. Nevertheless, Doukakis et al. [11] found that using an analogy still appears preferable over using none.

Thirty years ago, teaching variable naming was rarely included in programming textbooks [20]. There is no recent research on this topic, however, since then a considerate amount of work focused on the effect of naming on program comprehension, code quality and coding skills [2–5, 8, 19, 23, 24, 31, 32, 34]. Some results indicate that meaningful identifier names are most beneficial for code comprehension and debugging, and that "better names" are associated with better code quality. Few studies with a focus on naming aim at improving programming education. In the context of improving online curricula, Glassman et al. [14] developed a tool and a quiz for their MOOC to assess naming on length and vagueness. They found that feedback on naming practices, as well as both good *and* bad examples, was highly valued by students. Other studies on variable naming in education found that novice programmers often fail to name variables correctly [15] and that Scratch students are misled by variables named with a letter, probably because of prior knowledge from their mathematics education [17].

Observation studies on variable naming often target code quality and efficiency, and are based on names "found in the wild", meaning used by professional developers and/or taken from open source projects [3, 16, 27]. Although Swidan et al. [33] investigated naming practices in Scratch, a programming language for children, to the best of our knowledge no empirical observations investigate classroom practices on teaching variables or their naming.

3 METHODS

To answer our research questions, we analyzed seventeen MOOCs on the platforms *Coursera* and *EdX* throughout March and April 2022. We searched for the MOOCs using the keyword *"programming"*, filtering for 'courses' and 'available now', excluding archived courses. Additionally, we applied the following selection criteria: the MOOC has to be (1) a beginner's course without programming prerequisites, that (2) focuses on (fundamental) programming skills and concepts. This information is obtained from the title and course descriptions. Furthermore, the course is (3) provided by a university, (4) taught in English, and (5) has at least one of its objectives to teach Python, Java, or C. Lastly, the course (6) should be freely available to anyone. Thus, courses that cover some programming but mainly focus on data science or web development are excluded, as well as courses that are created by companies.

Following the criteria, we looked at the first two pages of search results on both platforms, as we argue that these courses are most likely to be chosen by those interested in learning the skill. Arguably, these are the most popular and relevant courses, with a significant number of students enrolled and high ratings (between 4.3 and 4.8 out of 5 stars). This led to the selection of seventeen MOOCs.

Of these seventeen MOOCs, seven are dedicated to Python programming (labeled P1-P7), six to Java (J1-J6), and four to C (C1-C4). One course (C1) teaches multiple languages, but as the C language was focused on most, we treat the MOOC as teaching C. Most of the courses are offered by US institutions, including prestigious universities such as Harvard (P3, C1), Princeton (J4), and the University of Pennsylvania (P6, J5). Only P5, J1, and J2 are respectively from Canada, Hong Kong and Spain. Ten MOOCs are taught by multiple instructors. Two teachers were (co-)teaching two MOOCs within our selection. In total there are thirty instructors.

To collect our data, we enrolled for the selected courses as would a regular student. The data that we used were: (1) the pre-recorded video lessons, (2) the lecture slides, (3) the practice materials and exercises, and (4) the additional explanations in between the video lessons. We used MS Excel to collect all relevant quotes, examples, and screenshots from the MOOCs. Data collection was carried out by the second author of this work. The first and second authors worked in close collaboration for the analysis, and uncertainties were discussed and resolved during regular sessions.

For RQ1a, we looked at the concepts explained right before, together with, and right after the introduction of the concept of variables. For RQ1b, we gathered the formal definitions given to variables and counted often recurring terms. We are specifically interested in how the definitions answer "what is a variable" and "what does a variable do". For RQ1c, we collected the analogies addressing the concept of variables, including visualizations that we found in the course material. For of RQ2a, we collected the explanations, tips, and explicit examples concerning variable naming practices. From these we established two categories: language rules, and human guidelines (conventions and variable name meaning). For RQ2b, we analyzed the variable names that were presented to students in videos, exercises and other learning materials.

4 RESULTS

4.1 How is the concept of variables taught?

4.1.1 Connection to other programming concepts. Most often the analyzed MOOCs introduce variables at the beginning of the course; either right before, simultaneously with, or right after introducing data types and/or arithmetic expressions (see Table 1). In P3 and P5, variables and functions are explained together, and in C1 and

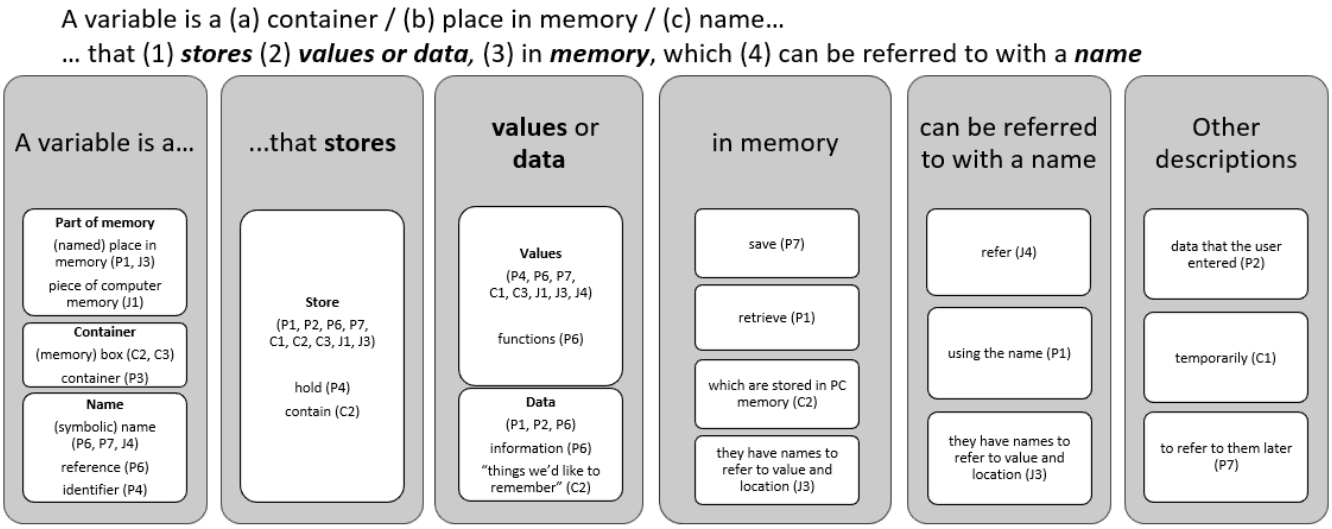


Figure 1: Analysis of variable definitions. The concepts are extracted from 12 definitions.

Table 1: Concepts related to variables and when they are introduced. DT=data types; EXP=expressions.

Variables are introduced...	MOOCs
...right before DT and EXP	C4, J1
...together with EXP, usually before DT	P1, C3, J2, J3, J6
...together with DT, usually before EXP	P2, P4, C2, J2, J5
...right after EXP and DT, but before FUNCTIONS	P6, P7, J4
...together with FUNCTIONS	P3, P5
...after FUNCTIONS	C1, C2

C2, variables are introduced after functions, control flow, and loops. P4 introduces variables together with the concept of control flow and code order: *"Variables are possibly the most fundamental element of programming. There really isn't much you can do without [them]. (...) We use variables to represent the information in which we are interested, like stock prices or user names, and we will also use variables to control how our programs run, like counting repeated actions or checking if something has been found."*

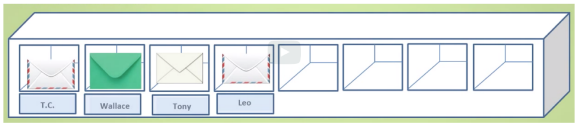
4.1.2 Definitions. A formal definition of a variable is given by twelve MOOCs (70%), from which we can identify four recurring elements: variables (1) **store**, (2) **values or data** in (3) **memory**, and (4) can be referred to with a **name**. As shown in Figure 1, almost all definitions agree on what variables do (storing values or data), but there is no consensus on what a variable is (container, name, place in memory). Both memory and naming are less frequently included than "storing values." Only P1 and J3 explicitly cover a 'complete' definition, for example, *"A variable is a named place (4) in the memory (3) where a programmer can store (1) data (2) and later retrieve the data using the variable name (4)" (P1).* An example of an incomplete definition is *"A variable is just a container for some value (2) inside a computer or inside of your own program" (P3).* Five MOOCs do not give a definition at all (P5, C4, J2, J5, J6).

An interesting finding is that, although not in the definition, C2 is the only MOOC to demonstrate that *flexibility* is also a major benefit, or even purpose, of using variables.

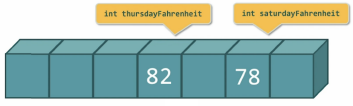
4.1.3 Analogies. Eight MOOCs (P3, P4, **P5**, C2, **C3**, **J1**, **J3**, **J6**) use analogies to explain the concept of variables. Of these, five (**bold**) also use matching visualizations. The identified analogies relate to the memory address and to the contents of a variable.

Most common is the analogy 'variable as a box', or variations of it, such as 'variables as a mailbox' (J1, Figure 2a). Boxes are often drawn with values inside and labels attached to the boxes (see Figure 2). The analogy also appears as only a visualization (P1) and without any visualizations but during 'live' coding: *"what we'd like to memorize is an integer value. Suppose I want to memorize this '17' right here. To do so, I need to first create a variable. So a memory box with room to store the 17 in. And then I need to place that 17 into this memory box" (C2).*

Some instructors explicitly relate variables to the computer's memory. J1 introduces a figure (Figure 3a) to explain that the current



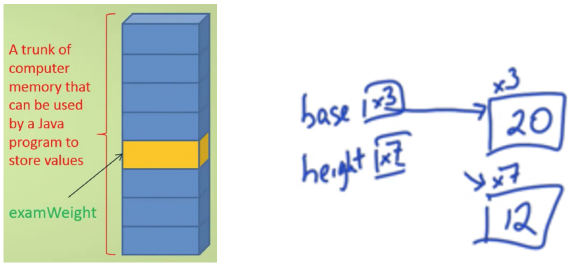
(a) J1, Variables as a mailbox. *"Each mailbox is labeled by its owner (or identifier) and different kinds of mails (or values) can be put into the mailbox"*



(b) J3, Variables as a labeled box

(c) P1, Variables as a box

Figure 2: Visualizations of the analogy "variables as a box"



(a) J1, memory (b) P5, box analogy and memory

Figure 3: Visualizations connected to computer memory

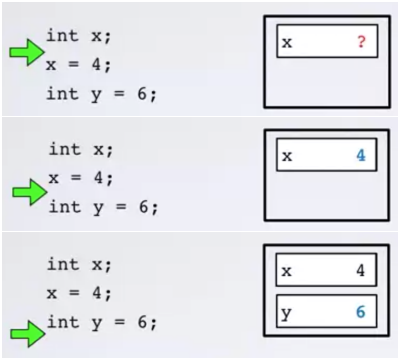


Figure 4: Three different stages of program execution (C3)

value of a variable can be retrieved by referring to its name; in the illustration, an arrow is drawn from the variable name to this piece of memory. P5 draws a variable name with a box (Figure 3b) and emphasizes that the value does not go into that box but lives at a particular memory address. To clarify, he draws another square with the value '20', picking an arbitrary memory address marked with 'x3', explaining that the assignment statement takes 'x3', and puts it in the box associated with the variable name. P5: *"So 'base' contains 'x3' and (...) what that means is that [it] points to the memory address 'x3' where the value '20' lives. (...) Python keeps track of [the variable's] value in that little box, and its value is a memory address."* He continues *"[the memory address is] arbitrary, Python is in charge of that choice, and so I don't need to worry about exactly what the memory address is as long as I know that this relationship between variables and their values exists."*

A recurring visualization addresses the declaration and initialization of variables, by also covering the topics of program execution and tracing (J4, C3, J6, Figure 4). After highlighting the importance of drawing pictures of what happens during program execution, and in line with the box-analogy, the instructors of C3 and J6 create a box labeled with the variable name for a first statement, entering a question mark as the variable is still uninitialized. They then illustrate that executing a second statement will put a value in the previously created box. As a result, the question mark disappears.

Another analogy we encountered is that of a "question-and-answer" format, which connects well to variable naming practices. P4: *"If you're ever confused about what a variable means, treat it as a question."* As an example, the instructor shows that "num cats" can become the question "Number of cats?". *"The value, then, is the answer to this question."*.

4.2 How are naming practices taught?

4.2.1 Which practices are taught? We observe two primary aspects that are explicitly taught: (1) 'syntax rules' that *need* to be applied for the language to interpret correctly, and (2) 'human guidelines' that *can* be applied to aid a human interpreter. The first category, syntax rules, includes legal characters, reserved keywords, and case sensitivity. The latter category, human guidelines, breaks down into two subcategories: standardized *conventions* and the *variable name meaning*, both supporting code readability. An overview of which topic was represented per MOOC is provided in Table 2. MOOCs mostly focus on conventions and syntax whereas variable name meaning is covered in only half of the MOOCs, often superficially and with the interpretation of 'meaningful' varying per course and context. Three MOOCs do not cover naming practices at all and two more only cover syntax rules.

1. Syntax. Nine out of ten MOOCs covering syntax rules mention which characters variables may contain: i.e. in Python variable names cannot start with a number character or contain spaces. Four MOOCs mention that certain words are reserved keywords, such as 'if', 'for', and 'return', as explained by J3: *"To avoid confusing the compiler, you can't use reserved words as identifiers. Reserved words are words that are already given specific meanings in Java."* Finally, six MOOCs bring up case sensitivity, for example, P1: *"And it's case sensitive, but we don't want you to depend on that. So 'spam', 'Spam' with one upper case, and 'SPAM' all are different variable names, but you're not doing anybody any favor if you think that's being clever."*

Table 2: Overview of taught variable naming practices, ordered by meaning, conventions then syntax. When meaning is discussed, conventions are always discussed too, but not the other way around. Syntax sometimes stands on its own

MOOC	syntax	conventions	meaning
P1	x	x	x
P4	x	x	x
C2	x	x	x
C4	x	x	x
J1	x	x	x
J2	x	x	x
P2		x	x
P7		x	x
C1		x	x
P5	x	x	
J3	x	x	
J5		x	
P6	x		
C3	x		
P3			
J4			
J6			
Total (N=17)	10	12	9

2a. Conventions. Twelve MOOCs introduce naming conventions, such as using underscores or camel case words. For example, P5 states, *"every programming language has a set of conventions for how to choose a name, much like websites have a particular style and layout. In Python most variable names use only lowercase letters with underscores to separate words, we call this pothole case."* Three (P7, C1 and J2) also specify using capitalized words for constant variables only as a subtle visual reminder: *"Kind of like you're yelling, but it really just visually makes it stand out. So (...) like a nice rule of thumb that helps you realize, oh, that must be a constant. Capitalization alone does not make it constant, but [it] is just a visual reminder that this is somewhere, somehow a constant" (C1).*

2b. Variable name meaning. Nine MOOCs bring up the topic of meaningful or mnemonic names. They generally highlight that names are important for the readability of the code, but not so much for the language interpreter, as explained by P1: *"I emphasize that one of the key things about variable names is that you get to name them. We have a technique called mnemonic, and the idea is that when you choose a variable name, you should choose a variable name to be sensible. Python doesn't care whether you choose mnemonic variable names or not. The name that you choose for a variable does not communicate any additional information to Python (...) so mnemonic variables are only for humans."*

Even though teachers mention to use meaningful names, the interpretation of 'meaningful' varies between teachers and contexts. One example is choosing between letters and words. Some teachers find the use of letters as variable names not that meaningful, and rather replace 'x', 'y' and 'z' by *"more meaningful names such as 'radius', 'area', 'scores', if possible" (J1).* Also 'i' and 'j' could be more meaningful by using 'row' and 'column' in certain contexts, P7: *"Notice that if we chose 'row' and 'column', we'd immediately know by reading the names that they're indices. And more importantly, we'd know if we're looking at the vertical index or the horizontal index" (...) "I think that you'll find, if you pay a little bit of attention to your variable names, this won't be much of a burden, and it'll lead to significantly better code".* P7 thus indicates that smart naming improves code quality and belongs to good programmer's practice. However, the instructor of C1 expresses that choosing a good name is all about context. In particular, in the context of arithmetic expressions ('x + y'), letters can be *"totally fine,"* as changing 'x' and 'y' to 'first number' and 'second number' *"isn't really adding anything semantically to help my comprehension" (C1).*

One teacher (C4) spends extra time to highlight the importance of context when it comes to good naming practices. He first shows names that are usually considered 'bad', such as 'grx33', 'pp_25' and 'i_am_FourWords', pointing to that these names give no clue of what they might be, and the latter even mixes two conventions. However, he also says: *"we are not going to necessarily know if [the names are] good or bad, (...) they're going to have to have the right context. So 'data' may or may not be good, it may not be adequately descriptive. Maybe you need 'height data', or 'weight data', or something else."* He continues with several other examples, one of them being: *"(...) Again, if you were writing some movie with Superman and Mxyzptlk, [Mxyzptlk] might be an appropriate identifier."*

To summarize, about half of the MOOCs address variable name meaning, but what is considered meaningful, or 'good naming' is hard to define: it differs per instructor and context. Only the instructors of C1 and C4 acknowledge the importance of *context* when choosing a name and sought to clarify this by presenting several examples.

4.2.2 Which practices are used by teachers? Most instructors apply the naming convention that is related to the language, so for Python and C that means lowercase letters and an underscore between words, and for Java it means using the camel case style. However, the instructors of MOOCs P4, C2 and C3 use the opposite style. Whereas the instructors of C2 and C3 do not give a clarification, P4's teacher tells the students: *"Each programming language has its own accepted style. In Python, you should use underscores. In Java and C#, you would use camel case. Other languages have their own conventions. 'But wait!' you say, 'You are using camel case in the videos!' That's right! I learned to program first in C++, and then in Java, and then in C#, three languages that use camel case instead of underscores. Old habits are hard to break!"*

Although nine MOOCs address the importance of meaningful variable names, only the instructor of P1 takes the students along to experience that importance. In the beginning of his course, he uses "silly" names such as 'eee', 'sval', 'xr' and 'nsv'. Later in the course, he uses examples such as 'count', 'largest_so_far', 'sum', and 'found'. He highlights that some names could be good names e.g. 'sval', 'fval' (string value, float value), but confusing to novices because they are unfamiliar with the terminology. However, in his explanations, he also stresses the unimportance of a name to the language interpreter, which reflects a stronger focus on 'names do not matter' than on 'what is a meaningful name': *"So you'll notice as I write, especially in these first two chapters, some of my codes use really dumb variable names and some of them use really clever ones. So I go back and forth to emphasize to you that the name of a variable, as long as it's consistent within a program, doesn't matter. And Python is perfectly happy" (P1).*

The instructor of C2 chooses to 'lead the way' by using more meaningful names right from the start all the way through to the end of the course. Examples of these names are 'age', 'balance', 'numberOfHazelnuts', and 'distanceTraveled'. In contrast, we observed that sometimes 'meaningless' names (for novices) are chosen to explain or show a specific concept, for example, C1 introduces the concept of variable scope with the help of 'foo'. Lastly, in almost all MOOCs, single letters, such as 'a', 'b', 'c', 'x', 'y', and 'z', are used to refer to variables holding numbers. Only the instructors of P4 and J1 do not use single letters and instead choose names such as 'aNumber', 'aInt", and 'aDouble'.

5 DISCUSSION

We investigated teaching practices regarding the concept of variables and their naming, by systematically observing how variables are taught in introductory programming MOOCs.

We found that variables are usually taught together or in close connection with data types and arithmetic expressions, most often at the very beginning of the course. Furthermore, we recognized a pattern in how the concept of variables is defined. A definition is given in twelve MOOCs (70%) and centers around '**storing**' '**data**'. Some MOOCs also refer to a computer's **memory** and/or **naming**. Although there is a clear consensus on what variables do, we have

seen no consensus on what variables are (i.e. part of memory, box, name). This might partly be due to the definitions of variables found in literature, such as 'containers that hold values' [35] and the origin of the concept from *The Analytical Engine*, where variables were used to store data. However, accessing, re-using, and modifying data is as important to the concept, but is little represented in our observation. Consistent with [30], the 'variable as a box' analogy was often used while explaining variables. Since we already know this analogy might cause certain misconceptions [9, 18], we suggest teachers should keep this in mind. We also found a questions-and-answer format as an analogy and some definitions that refer to variables as references. It would be interesting to further investigate how these latter analogies influence students learning, as well as the effect of a shift from 'storing data' towards other aspects of variables such as how and why we use them.

Naming practices are explicitly taught in most MOOCs, with syntax rules and conventions more often attended to than meaningful variable naming. Only half of the MOOCs allocate time to such naming practices, and when meaning is touched upon, a discussion on 'what is meaningful' is rarely implemented. Although plenty of provided examples concern syntactically acceptable names, few examples concern variable name meaning. This shows that not much seems to have changed since 1990 when Keller [20] established that choosing meaningful names for variables is rarely covered in programming textbooks. Moreover, our results could explain why many novice programmers fail to name variables correctly [15].

When we look at implicit naming practices, in particular, how instructors use naming in the provided materials, we found that not all instructors used the naming convention style that is generally accepted for the respective language. However, Shariff and Maletic [32] indicate that underscore-styles versus camel case-styles do not influence a programmer's accuracy, which suggests that this inconsistency should not make much difference in student learning.

Furthermore, we found that instructors used different approaches regarding the meaning of a name. Only few MOOCs explicitly provided good and bad examples throughout the course or led the way from the start by always using names conveying the meaning of the content. In most MOOCs, letters were primarily used in demonstrations of code, and sometimes meaningless names were chosen to explain a certain concept. These findings reflect issues regarding naming practices in programming education. Firstly, they stress that meaningful naming practices are not very common in online education [20]. Secondly, the dichotomy in using letters as names, both when explicitly taught or solely used in the provided materials, is also reflected in the literature. For example, Lawrie et al. [23, 24] found that full word names are more effective for comprehension than letters, whereas Beniamini et al. [3] conclude that letters *can* be meaningful when they convey information that is commonly attributed to that letter. However, we observed that there sometimes exists no consistency in naming practices, which may leave students confused.

To conclude, it is becoming clear that appropriate variable names impact how quickly and how well a code is understood [2–5, 8, 19, 24, 31, 34]. We, therefore, might assume that, for this reason alone, they influence how learners learn a programming language, yet, still little is known about how these practices are taught and how exactly they influence learners. Based on our results, we feel a strong urge for both teachers and researchers to pay more attention to variable name meaning as part of naming practices.

5.1 Limitations

Since our study only covered the free-to-follow part of MOOCs, we did not include premium content features such as additional tests, assignments, or videos. It is possible that we have missed certain practices because of this, however, we wanted to examine what was available to everyone. Nevertheless, our answers on RQ2b, which naming practices do teachers use themselves, may have been influenced most, since it would have been valuable to see how instructors named their variables on the spot, a practice that was not always freely available. Furthermore, most of the MOOCs were created at US institutions, which might not be representative for online courses made and followed in other parts of the world. This is most likely an effect of our selection criterion that the MOOCs should be taught in English, or the fact that we used *edX* and *Coursera*. It would be interesting to compare our results with courses from other parts of the world, taught in local languages. Especially on the topic of variable name meaning, we would expect to see compelling variances.

6 CONCLUSION

We gained insight into how variables are taught in introductory programming education, in particular in MOOCs teaching Python, C, or Java. We found that the concept of variables is embedded through connections with other concepts such as data types, expressions, and program execution. There is a strong focus on storing data, whereas memory and naming are less well represented. Even flexibility as a benefit or purpose of variables is rarely mentioned. Furthermore, naming does not get consistent attention. Only a few MOOCs discuss the topic consistently with special attention to the meaning and context of names, whereas other MOOCs show inconsistency between taught and used practices, or show no discussion regarding meaningful naming at all.

Based on our results, we stress the importance for both teachers and researchers to pay more attention to naming practices, in particular to variable name meaning, and think about how these might influence the learning process of students. For future work we suggest extending our research by including observations from courses offered by tech companies and on YouTube, as many programmers might learn their skills there. Furthermore, we have conducted in-depth interviews with teachers of secondary-level and university-level education to complement the current research, with a special focus on naming practices (in review). Finally, it could be interesting to connect our results to known misconceptions, as suggested by [18].

ACKNOWLEDGMENTS

This project was funded by the Leiden-Delft-Erasmus Centre for Education and Learning.

REFERENCES

[1] Venera Arnaoudova, Massimiliano Di Penta, and Giuliano Antoniol. 2016. Linguistic antipatterns: what they are and how developers perceive them. *Empirical Software Engineering* 21, 1 (Feb. 2016), 104–158. https://doi.org/10.1007/s10664-014-9350-8

[2] Eran Avidan and Dror G. Feitelson. 2017. Effects of Variable Names on Comprehension: An Empirical Study. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 55–65. https://doi.org/10.1109/ICPC.2017.27

[3] Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach, and Dror G. Feitelson. 2017. Meaningful Identifier Names: The Case of Single-Letter Variables. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 45–54. https://doi.org/10.1109/ICPC.2017.18

[4] Dave Binkley, Dawn Lawrie, Steve Maex, and Christopher Morrell. 2009. Identifier length and limited programmer memory. *Science of Computer Programming* 74, 7 (2009), 430–445. https://doi.org/10.1016/j.scico.2009.02.006

[5] Scott Blinman and Andy Cockburn. 2005. Program Comprehension: Investigating the Effects of Naming Style and Documentation. In *AUIC*.

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

[7] Bruno Caprile and Paolo Tonella. 2000. Restructuring program identifier names. In *Proceedings 2000 International Conference on Software Maintenance*. IEEE, San Jose, CA, USA, 97–107. https://doi.org/10.1109/ICSM.2000.883022

[8] Roee Cates, Nadav Yunik, and Dror G. Feitelson. 2021. Does Code Structure Affect Comprehension? On Using and Naming Intermediate Variables. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. 118–126. https://doi.org/10.1109/ICPC52881.2021.00020

[9] Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, and Matthias Hauswirth. 2021. A Curated Inventory of Programming Language Misconceptions. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (Virtual Event, Germany) *(ITiCSE '21)*. Association for Computing Machinery, New York, NY, USA, 380–386. https://doi.org/10.1145/3430665.3456343

[10] Malcolm Corney, Raymond Lister, and Donna Teague. 2011. Early Relational Reasoning and the Novice Programmer: Swapping as the "Hello World" of Relational Reasoning. In *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114* (Perth, Australia) *(ACE '11)*. Australian Computer Society, Inc., AUS, 95–104.

[11] Dimitrios Doukakis, Maria Grigoriadou, and Grammatiki Tsaganou. 2007. Understanding the programming variable concept with animated interactive analogies. In *Proceedings of the 8th Hellenic European Research on Computer Mathematics & its Applications Conference, HERCMA'07*.

[12] Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L. Pearce, and Andrew Petersen. 2020. Notional Machines in Computing Education: The Education of Attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE-WGR '20)*. Association for Computing Machinery, New York, NY, USA, 21–50. https://doi.org/10.1145/3437800.3439202

[13] Michelle Gienow. 2017. Code n00b: The (Variable) Naming Is the Hardest Part. https://thenewstack.io/code-n00b-naming-hardest-part/.

[14] Elena L. Glassman, Lyla Fischer, Jeremy Scott, and Robert C. Miller. 2015. Foobaz: Variable Name Feedback for Student Code at Scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 609–617. https://doi.org/10.1145/2807442.2807495

[15] Abdul Rahman Mohamad Gobil, Zarina Shukor, and Itaza Afiani Mohtar. 2009. Novice difficulties in selection structure. In *2009 International Conference on Electrical Engineering and Informatics*, Vol. 02. 351–356. https://doi.org/10.1109/ICEEI.2009.5254715

[16] Remo Gresta, Vinicius Durelli, and Elder Cirilo. 2021. Naming Practices in Java Projects: An Empirical Study. In *XX Brazilian Symposium on Software Quality* (Virtual Event, Brazil) *(SBQS '21)*. Association for Computing Machinery, New York, NY, USA, Article 10, 10 pages. https://doi.org/10.1145/3493244.3493258

[17] Shuchi Grover and Satabdi Basu. 2017. Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 267–272. https://doi.org/10.1145/3017680.3017723

[18] Felienne Hermans, Alaaeddin Swidan, Efthimia Aivaloglou, and Marileen Smit. 2018. Thinking out of the Box: Comparing Metaphors for Variables in Programming Education. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education* (Potsdam, Germany) *(WiPSCE '18)*. Association for Computing Machinery, New York, NY, USA, Article 8, 8 pages. https://doi.org/10.1145/3265757.3265765

[19] Johannes Hofmeister, Janet Siegmund, and Daniel V. Holt. 2017. Shorter identifier names take longer to comprehend. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 217–227. https://doi.org/10.1109/SANER.2017.7884623

[20] Daniel Keller. 1990. A guide to natural naming. *ACM SIGPLAN Notices* 25 (1990), 95–102.

[21] Tobias Kohn. 2017. Variable Evaluation: An Exploration of Novice Programmers' Understanding and Common Misconceptions. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 345–350. https://doi.org/10.1145/3017680.3017724

[22] Ilker Koksal. 2020. The Rise Of Online Learning. https://www.forbes.com/sites/ilkerkoksal/2020/05/02/the-rise-of-online-learning/?sh=28f26e472f3c.

[23] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. 2006. What's in a Name? A Study of Identifiers. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*. 3–12. https://doi.org/10.1109/ICPC.2006.51

[24] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. 2007. Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering* 3, 4 (Dec. 2007), 303–318. https://doi.org/10.1007/s11334-007-0031-2

[25] Raymond Lister, Colin Fidge, and Donna Teague. 2009. Further Evidence of a Relationship between Explaining, Tracing and Writing Skills in Introductory Programming. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education* (Paris, France) *(ITiCSE '09)*. Association for Computing Machinery, New York, NY, USA, 161–165. https://doi.org/10.1145/1562877.1562930

[26] Mike Lopez, Jacqueline Whalley, Phil Robbins, and Raymond Lister. 2008. Relationships between reading, tracing and writing skills in introductory programming. In *ICER'08 - Proceedings of the ACM Workshop on International Computing Education Research* (New York, New York, USA). ACM Press, 101–111. https://doi.org/10.1145/1404520.1404531

[27] Christian D. Newman, Reem S. AlSuhaibani, Michael J. Decker, Anthony Peruma, Dishant Kaushik, Mohamed Wiem Mkaouer, and Emily Hill. 2020. On the generation, structure, and semantics of grammar patterns in source code identifiers. *Journal of Systems and Software* 170 (2020), 110740. https://doi.org/10.1016/j.jss.2020.110740

[28] Thomas Pelchen and Raymond Lister. 2019. *On the Frequency of Words Used in Answers to Explain in Plain English Questions by Novice Programmers*. Association for Computing Machinery, New York, NY, USA, 11–20. https://doi.org/10.1145/3286960.3286962

[29] Jorma Sajaniemi. 2002. An empirical analysis of roles of variables in novice-level procedural programs. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*. 37–39. https://doi.org/10.1109/HCC.2002.1046340

[30] André L. Santos and Hugo Sousa. 2017. An Exploratory Study of How Programming Instructors Illustrate Variables and Control Flow. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '17)*. Association for Computing Machinery, New York, NY, USA, 173–177. https://doi.org/10.1145/3141880.3141892

[31] Andrea Schankin, Annika Berger, Daniel V. Holt, Johannes C. Hofmeister, Till Riedel, and Michael Beigl. 2018. Descriptive Compound Identifier Names Improve Source Code Comprehension. In *Proceedings of the 26th Conference on Program Comprehension* (Gothenburg, Sweden) *(ICPC '18)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/3196321.3196332

[32] Bonita Sharif and Jonathan I. Maletic. 2010. An Eye Tracking Study on camelCase and under_score Identifier Styles. In *2010 IEEE 18th International Conference on Program Comprehension*. 196–205. https://doi.org/10.1109/ICPC.2010.41

[33] Alaaeddin Swidan, Alexander Serebrenik, and Felienne Hermans. 2017. How do Scratch Programmers Name Variables and Procedures?. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 51–60. https://doi.org/10.1109/SCAM.2017.12

[34] Barbee E. Teasley. 1994. The effects of naming style and expertise on program comprehension. *International Journal of Human-Computer Studies* 40, 5 (1994), 757–770. https://doi.org/10.1006/ijhc.1994.1036

[35] Leslie J. Waguespack. 1989. Visual Metaphors for Teaching Programming Concepts. *SIGCSE Bull.* 21, 1 (feb 1989), 141–145. https://doi.org/10.1145/65294.71203

Received 22 January 2023; accepted 6 March 2023