Universiteit Leiden
The Netherlands

Opleiding Informatica

Teaching of Variables in Programming MOOCs:
An Exploratory Study

Min Yi Zhang

Supervisors:
Vivian van der Werf & Efthimia Aivaloglou

BACHELOR THESIS

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

23/06/2022

Abstract

Programming is a complicated subject that demands special effort and particular strategies to understand and teach. Thus, studying how programming concepts are taught is essential and begins with the fundamental concept of the variable. This study's main objective is to understand the current state of teaching variables in online programming education. To achieve this, we conducted a study to analyse how variables are taught in introductory programming MOOCs. We looked at when variables were introduced in the courses, what definitions and teaching strategies were used for explaining variables, what topics instructors introduced concerning variables, and how much time they were given. We also looked at how instructors used variables in example programs, including whether there were any differences in how variables are taught between three programming languages: Python, Java, and C. The results of our study confirm that the basic programming concept of a variable indeed deserves special attention and shows some interesting patterns and gaps found in the programming MOOCs.

Contents

1 Introduction

1.1 Problem Statement

Programming is becoming a highly valued skill. The market for programmers has increased significantly in the past few years, as has student interest in this topic. As a result, introductory programming courses have become extremely popular. However, learning to program in any language is not a simple task. Programming has traditionally been regarded as a complex topic because its concepts are tough to comprehend, particularly for beginners. The significant dropout rate in these introductory programming courses confirms this [KM06]. The question arises: *Why is it so difficult to understand programming?* One explanation is that programs deal with abstract concepts that have nothing to do with real-world problems [LAMJ05].

Although a lack of real-world problems may contribute, even basic programming concepts like variables remain tricky for many students to understand [DR08]. Because many other programming concepts, including control flow and loops, expand on the concept of a variable, it is a significant building block that demands much attention. Thus, it is essential that novices thoroughly understand this concept of variables; otherwise, they might have trouble later learning other, more complicated programming concepts.

So, what is it that makes understanding variables so challenging? Variables are often difficult to grasp since they are very abstract. To write efficient programs and reason about them, one should develop a consistent mental model of what is going on inside the computer and understand how program instructions are executed [BB15, BM83]. Hands-on experience in programming alone appears to be insufficient for novices to comprehend fundamental programming concepts productively. Thus, understanding students' mental models is essential for instructors to implement effective teaching strategies.

The concept of variables is also troublesome for novices because they appear to have some misconceptions [Koh17]. When instructors teach new programming concepts, they frequently introduce metaphors. These metaphors may be beneficial in explaining the abstract world of programming, but can also lead to misunderstandings. For instance, instructors use a "box" metaphor to describe a variable, which may lead novices to believe that a variable can hold multiple values [HSAS18].

Another problem that novice and experienced programmers face is variable naming [Gie17]. Both groups have difficulty coming up with 'good' variable names. After all, it is the task of the programmer to ensure that the names they assign to their variables are clear to other readers. This is something that must be learned, but it is unclear to what extent it is being taught. How can we help students and instructors overcome these obstacles?

The questions posed above leave room for further research. It is now simpler than ever to become a self-taught programmer as the number of online resources continually increases. Even the oldest or dying programming languages still have many online courses or videos available to learn from. There is, however, one potential drawback to self-teaching, which makes investigating this field even more interesting: aspiring programmers must be highly disciplined. Learning to program is

a never-ending learning process; most progress is made when one continues with it [Wei22]. As a result, studying how programming concepts are taught is beneficial and starts with the basic concept of the variable.

To gain a picture of the current state of teaching variables in online education, the main objective of this thesis is to analyse how variables are taught in introductory programming courses. To achieve this, we analysed several online introductory programming courses. In particular, this research will use MOOCs (Massive Open Online Courses). Online learning platforms like Coursera and edX have attracted increasing attention in recent years [Kok20], and there is still a lot to explore in the programming field. Unlike previous research that concentrates on approaches and methods that instructors can use to improve the teaching of the concept of variables, this thesis starts at the root and investigates the current state-of-the-art in the teaching of variables. Besides gaining a current picture that may show interesting patterns or gaps, the results of this study might also help instructors obtain a better understanding of the approaches and techniques used by other instructors to teach this basic concept of a variable. Instructors may identify opportunities for improving their teaching.

1.2 Research Questions

The main research question of the thesis is: *How are variables taught in introductory programming MOOCs?*

To address this question, the following five sub-questions were formulated (RQ1-RQ5):

RQ1. When are variables introduced in the courses?

RQ2. What definitions and teaching strategies are used for explaining variables?

RQ3. What topics do instructors introduce concerning variables, and how much time are they given?

RQ4. How do the instructors use variables in example programs?

RQ5. Is there a difference in how variables are taught between Python, Java and C?

1.3 Structure

The rest of this paper is organised as follows. Section 2 reviews the work related to this research. Section 3 describes the research methods used to analyse the programming MOOCs. Section 4 presents the answers to the research questions, followed by a discussion in Section 5. Finally, some concluding thoughts and directions for further research are offered in Section 6.

2 Related Work

A considerable amount of literature has focused on approaches that instructors may adopt to assist their teaching of programming. In the following paragraphs, we will review some of these works concerned with methods for improving the teaching of programming concepts, precisely the basic programming concept of a variable. A particular focus is on Analogies and Visual Metaphors (2.1), Roles of Variables (2.2), and Variable Naming (2.3).

2.1 Analogies and Visual Metaphors

Fincher et al. [FJM+20] argue that understanding what code does is hard to visualise. For instance, computer software, including a compiler, is not graphically depicted. Novice programmers simply understand that 'something' happens inside the computer. One approach instructors have discovered to solve this problem is through the construction of notional machines. Notional machines are representations or analogies that highlight significant behaviour. In the programming field, a notional machine is a pedagogic device used to aid in the comprehension of some component of programs or programming. For instance, Fincher et al. [FJM+20] provided a notional machine that explains variables as an analogy to parking spaces.

However, Doukakis et al. [DGT07] remark that instructors have used inadequate analogies to teach variables, such as the 'box' or the 'mailbox' analogies, resulting in misconceptions. However, they also comment that difficulties arose when no analogies were used. As a result, they investigated if presenting students with a selected analogy for explaining variables might improve their understanding of the concept of variables. This newly introduced analogy comprised several rotating cylinders containing numbers, characters, or Boolean values, depending on the variable type. The analogy was animated and interactive to enable students to experiment and was integrated into a learning object. The learning object was divided into six sections that students could view in any order they wanted. These sections included the definition of a variable, variable type such as integer variable, and the declaration of variables. Thirty-three students participated in the study, twenty-one of whom were allocated to the analogy group and twelve to the control group. The result favoured the group with an interactive animated analogy over the group involving a learning object without this analogy. Doukakis et al. [DGT07] concluded that this group showed fewer misunderstandings and a greater understanding of variables. Therefore, although analogies can be misleading, using one appears preferable to using none.

In a similar vein, Waguespack [WJ89] demonstrates in his paper a system of visual metaphors utilised in a Pascal introductory programming course. These visual metaphors express programming concepts such as variables, arrays, records, and modules, which can sometimes be challenging for novices to understand. The author is convinced that the best way to understand programming concepts lies in the models used by instructors and their students to convey abstraction in programming [WJ89]. Therefore, his classes used visual metaphors to construct a mental model to show and explain programming abstractions. Drawing on six years of experience, he notes that these metaphors enhance students' learning and a general idea of programs. So, what metaphor did he use to explain the programming concept of a variable? Waguespack [WJ89] defines variables as containers that can hold values. Since a container has the capacity for only one value, assigning a new value to a

variable would require that the old value be removed first before this new value may take its place. The analogy of a "container" seems appropriate for explaining the concept of variables. Moreover, it could replace misleading analogies, such as defining variables as boxes.

2.2 Roles of Variables

Another approach that can assist novices in learning to program is the theory about the roles of variables developed by Sajaniemi in 2002 [Saj02]. These roles of variables define stereotypical uses of variables that appear regularly in programs. He states that in novice-level programming, just ten of these roles are essential in covering 99 per cent of all variables. What makes this even more appealing is that these roles may be expressed in a concise and understandable manner. To provide a few examples, consider a variable used to count the number of times a specific piece of code is executed. According to Sajaniemi, this variable would have the role of a 'stepper.' Alternatively, a variable might be used to accumulate the effect of individual values. Consider a variable that maintains track of a bank account's balance, where deposits and withdrawals are summed. Thus, having the role of a 'gatherer.' Likewise, there are 'constant', 'follower', 'most-recent holder', 'most-wanted holder', 'one-way flag', 'temporary', organiser' and 'other' [Saj02].

Kuittinen and Sajaniemi [SK05] carried out a classroom experiment in 2005 with three conditions to test the influence of using this theory regarding the roles of variables in teaching novices programming. One group of students was taught in the traditional form the course had previously been delivered. The second group was instructed using roles of variables throughout the course. Lastly, in addition to employing roles in instruction, the third group used a role-based animator in exercises. The role-based animator referenced in this study was first presented by the same authors in another paper [SK03] two years earlier. They explain that PlanAni, the role-based animator's moniker, employs role images to visualise variables and role-based animation to visualise operations. A role image, a visualisation used for all variables having that role, provides information on how the variable's consecutive values are connected with one another and with other variables. The authors provide several good examples to help understand this notion of a role image. For instance, a variable having the role of 'a fixed value' is represented by a stone, because this provides the sense that a value is difficult to change. Or another instance is that of a 'most-wanted holder' role, illustrated by flowers of different colours: a bright colour for the current value since this is the best value found thus far and a grey colour for the previous or next best value. The results show that using roles of variables offers students a new conceptual framework, allowing them to mentally handle programs in a manner comparable to skilled code comprehenders [SK05]. The application of the role-based animator appears to facilitate the students in embracing the knowledge of the roles. Thus, it appears helpful to introduce roles of variables in teaching novices and was therefore included in our research.

2.3 Variable Naming

Keller [Kel90] claims in his study that every programmer encounters the difficulty of choosing meaningful names for variables. However, this topic is rarely covered in programming textbooks: he continues that since programs are mainly interpreted by humans, meeting the compiler's constraints for syntactical correctness is a must. It is, however, not a sufficient standard. According to a

study conducted by Deissenboeck and Pizka [DP06], identifiers (containing variables, functions, structures, classes, et cetera) make up the vast majority (seventy per cent) of source code in terms of characters. Accordingly, the chosen names for variables play a vital role in the readability and comprehension of programs [AF17]. Nearly every programming language permits programmers to choose names, sometimes resulting in meaningless variable names arbitrarily. Fortunately, coding practices and naming conventions exist to tackle this issue. However, they are often ambiguous, and simply requiring that they be meaningful or self-descriptive is inadequate. As Deissenboeck and Pizka [DP06] identify in the study mentioned above, precise rules for concise and consistent naming are needed.

As a result, receiving feedback on variable names becomes educational. Glassman et al. [GFSM15] acknowledge in their study that existing traditional feedback procedures, including hand-grading students' source code for coding style, are time-consuming, inefficient, and unscalable to the capacities of MOOCs, where thousands of students can take part at once. Even more so because variable naming is an essential part of writing understandable and manageable code, so reasonably, many instructors would want to leave feedback. However, for instructors to determine the quality of a variable name, they must first understand the context and the role that variable plays within the surrounding code. To address this, Glassman et al. [GFSM15] developed Foobaz, a user interface to enable instructors to provide personalised feedback on the variable names of students at scale. It allows them to assess the quality, whether the variable names are misleading, vague or too short, of the variable names chosen by students in their code while considering the function a variable name has. According to the findings, the interface assisted instructors in providing personalised variable name feedback on thousands of students from an introductory programming MOOC on the MOOC platform edX. The student reactions were also promising; it encouraged them to consider what makes up variable names being 'good' or 'bad', in this case, whether the names were misleading, vague or too short.

Having presented several approaches to teaching programming presented by different instructors, it would be interesting to see if the MOOCs we will analyse also use these or similar approaches in their courses when explaining the programming variables. Or do they have any additional teaching strategies we have not seen before in the existing research? Another concern is whether instructors pay enough attention to the basic programming concept of a variable. In this study, we attempt to answer these questions using the five sub-questions presented in the previous chapter. As such, we hope to acquire an understanding of the current state of teaching variables in online introductory programming education.

3 Methods

This chapter will discuss the data collection process and present the criteria for selecting the MOOCs from the wide assortment of MOOCs and MOOC platforms available on the internet, followed by a description of these MOOCs. After that, we will describe our analysis procedures and operationalisation.

3.1 Data Collection

To examine how variables are taught in online introductory programming courses, seventeen MOOCs on the MOOC platforms Coursera and edX were analysed. We used the MOOC platforms: edX and Coursera, since these two platforms are among the more popular MOOC platforms and provide an extensive database of programming courses. We searched for these courses throughout the months of March and April 2022.

Since this research focuses on the teaching of variables, we selected only introductory or beginner-level programming courses. While reviewing the programming course prerequisites, we excluded MOOCs that required extensive prior knowledge of mathematics or computer science. We included MOOCs with no or some basic mathematics, such as algebra, as a qualification in our analysis. Moreover, the MOOCs needed to be taught in English and at least one of their objectives learning a programming language, specifically Python, Java, or C; all other programming languages were excluded. The MOOCs had to teach fundamental programming concepts, including but not limited to data types, variables, control structures, and functions. These concepts are more pertinent to computer science than specialisations, such as data science or web development.

Furthermore, the filter parameters 'learning type' in edX and 'learning program' in Coursera were set to 'courses' to only search for courses. We set the filter parameter for the MOOCs' availability in edX to 'available now' (March-April 2022), and we decided not to include archived courses in our analysis. We used the keyword 'programming' on the MOOC platforms edX and Coursera to obtain the selected MOOCs. Additionally, we only considered MOOCs created by other universities. Hence, MOOCs created by companies, including Google, were excluded to limit the amount. Furthermore, we only focused on MOOCs that were free for anyone to enrol in. Ultimately, the first twenty-four search results shown on the edX and Coursera platforms, using these filters and criteria, were selected for this study. These are the most popular and relevant courses, with a significant number of students enrolled and high ratings. Because they are displayed on the first two pages, we suspected that students who want to learn about programming are likelier to choose one of these courses.

3.2 Description of the MOOCs

A MOOC, as mentioned earlier, is an abbreviation for Massive Open Online Course and is a distance learning course offered by various universities around the world [Pop22]. MOOCs frequently include two enrolment choices: free auditing or paid enrolment [Bow21]. The former provides students free access to video lessons, readings, assessments, and discussion boards, whereas the latter enables students to access all content, such as paid content features, including a certificate of completion. Another set of MOOCs is pay-only, as said. We selected the first option for our research: free

audition or entirely free.

Moreover, MOOCs provide several benefits; they are open to anyone as it allows students to study a subject in-depth without the limits of a conventional academic course. No prior qualifications are needed, and students can easily access the weekly online materials and progress through the online course at their own pace. Although, most MOOCs are only accessible for a limited period, which varies for every MOOC. Furthermore, MOOCs rarely feature end-of-course assessments or examinations that must be passed. Instead, there are typically voluntary weekly online quizzes or peer-reviewed assignments.

Table 1 shows the final selection of MOOCs from the MOOC platforms edX and Coursera. Additionally, the platform Coursera also provided the course ratings for each course. Students can rate a course on a scale of one to five stars and leave reviews for instructors and other students. From the list of selected MOOCs, the lowest rating was 4.3, the highest was 4.8, and the average was 4,6.

	Platform	**Course Name**	**Institution**	**Language**	**Students**	**Instructors**
P1	edX	Programming for Everybody (Getting Started with Python)	The University of Michigan	Python	495.668	1 Male
P2	edX	Basics of Computing and Programming	New York University	Python	73.602	2 Male
P3*	edX	CS50's Introduction to Programming with Python	Harvard University	Python	111.332	1 Male
P4	edX	Computing in Python I: Fundamentals and Procedural Programming	The Georgia Institute of Technology	Python	213.584	1 Male
P5	Coursera	Learn to Program: The Fundamentals	University of Toronto	Python	294,180	1 Female, 1 Male
P6*	Coursera	Introduction to Python Programming	University of Pennsylvania	Python	28,969	1 Male
P7	Coursera	Python Programming Essentials	Rice University	Python	76,267	2 Male
C1*	edX	CS50's Introduction to Computer Science	Harvard University	C, Python, SQL, JavaScript, CSS, HTML	3.608.205	3 Male
C2	edX	C Programming: Getting Started	Dartmouth College, IMT	C	151.438	1 Female, 1 Male

C3*	Coursera	Programming Fundamentals	Duke University	C	171,973	2 Female, 1 Male
C4	Coursera	C for Everyone: Programming Fundamentals	University of California, Santa Cruz	C	189,526	1 Male
J1	edX	Introduction to Java Programming - Part 1	The Hong Kong University of Science and Technology	Java	262.712	3 Male
J2	edX	Introduction to Java Programming: Starting to code in Java	Universidad Carlos III de Madrid	Java	353.214	2 Female, 1 Male
J3	edX	Introduction to Object-Oriented Programming with Java I: Foundations and Syntax Basics	The Georgia Institute of Technology	Java	12.930	1 Male
J4	Coursera	Computer Science: Programming with a Purpose	Princeton University	Java	164,534	2 Male
J5*	Coursera	Introduction to Java and Object-Oriented Programming	University of Pennsylvania	Java	11,681	1 Male
J6*	Coursera	Java Programming: Solving Problems with Software	Duke University	Java	309,978	1 Female, 3 Male

Table 1: List of selected MOOCs from the MOOC platforms edX and Coursera

As shown in the Table 1, most programming courses are offered by institutions in the United States. Some of these courses are from prestigious universities such as Harvard, Princeton, and the University of Pennsylvania. In our list of MOOCs, we have eight MOOCs dedicated to Python programming, six to Java programming, and three to C programming. One MOOC was devoted to C, Python, and other programming languages. The total number of students who have ever enrolled in the course ranges from 11,681 to 3,608,205.

In addition, we also looked at the instructors who teach these MOOCs. We examined seventeen MOOCs, ten of which were taught by more than one instructor. These online programming courses were taught by thirty instructors in total, with three instructors each (co-)teaching two MOOCs on our list. Among this group of thirty instructors were twenty-three male and seven female instructors. Sixteen of the twenty-one people who had shared their previous education in their biography had a computer science background, whether it was a bachelor's, master's, or doctorate. Moreover, the instructors come from a wide variety of research fields. A few examples of these research fields are

computer graphics, artificial intelligence, systems software, computer architecture, computer science education, learning management systems and multimedia computing.

3.3 Analysis and Operationalisation

To start our analysis, we first signed up for accounts on both edX and Coursera. Afterwards, we enrolled in the specified courses and followed these courses as regular novice students attempting to learn more about programming. As such, we watched the pre-recorded video lessons, downloaded the lecture slides ensuring that we could read the material without interruption, studied the videos on practice exercises, and read the additional explanation between the videos. For the analysis, we used the spreadsheet program MS Excel. We made the following operationalisation to answer the five sub-questions presented in the first chapter.

RQ1. When are variables introduced in the courses?
To answer this question, we noted when the instructors first started explaining the programming concept of a variable, whether it was at the start of the online course, in the middle, or at the end, and if it was the first programming concept discussed. Moreover, we looked at what was introduced before, after, and simultaneously with variables to delve deeper into this question. We suspected that there could be a pattern in how variables are presented concerning order and structure.

RQ2. What definitions and teaching strategies are used for explaining variables?
We looked at how instructors defined variables. Afterwards, the following four questions were derived:

1. What do variables do?
2. What do variables store/hold/save?
3. Where do variables store/hold/save values/data?
4. How can we refer to variables?

A definition that answers all four questions can be regarded as a complete definition. Moreover, we also examined what teaching strategies they employed to teach the concept of a variable. Some teaching strategies were described in the previous chapter, for instance, analogies and visualisations. We were interested if they used these or similar teaching strategies.

RQ3. What topics do instructors introduce concerning variables, and how much time are they given?
We were specifically interested in the following topics; variable assignment, variable naming, and variable role. In particular, the first topic received special attention since it is a subject where many novices struggle and create misconceptions. Therefore, we noted whether the instructors addressed any common misconceptions. Regarding variable naming, we examined how extensively they describe the naming rules for variables and if they are sufficient. Concerning time, we measured how many minutes and seconds instructors spent explaining variables. We contrasted this with another fundamental programming concept, namely data type, to better understand what the

numbers of minutes signify.

RQ4. How do the instructors use variables in example programs?
How instructors used variables was solely assessed on variable naming. We tried to find out how instructors name variables, examining the variable names in the video lessons, the lecture slides, and the videos on practice exercises. We investigated if the naming of variables on lecture slides differed in practice when instructors had to program on the spot. Moreover, we also examined whether the instructors followed coding practices and the naming conventions of the programming languages. Finally, we were interested in any standard abbreviations that instructors may use.

RQ5. Is there a difference in how variables are taught between Python, Java and C?
To address this sub-question, we applied cross-analysis. After we analysed all the courses, we examined whether there were any differences in teaching variables across the programming languages Python, Java, and C. We posed this question because we expect differences based on the different nature of the languages; the programming languages C and Java are compiled programming languages, whereas Python is an interpreted programming language. Another distinction is that C and Java are statically typed, whereas Python is dynamically typed. Another notable difference is the naming convention. Java has lower camel case as a naming convention, whereas, in C and Python, names should be lowercase, with words separated by underscores.

The operationalisation described above was assigned to different columns, while the seventeen MOOCs were assigned to the rows of the spreadsheet (see Appendix for an example). Then, one by one, we analysed the various MOOCs by answering these questions in the associated cell in MS Excel. The analysis started in April and lasted until the end of May 2022.

4 Results

This chapter presents the results of our analysis and answers the five sub-questions posed in the introduction. We have assigned each MOOC an abbreviated name, as stated in the table in the previous chapter. We will use this abbreviated name to refer to these MOOCs in the remaining sections. The findings are organised into five sections, each addressing one of the five sub-questions.

4.1 Introduction of Variables in the Courses

This section will answer the following sub-question: *When are variables introduced in the courses?*

The instructors in the seventeen analysed MOOCs introduced the programming concept of a variable early on in the online course. In particular, nine MOOCs (P3, P5, P6, P7, C3, J2, J4, J5 and J6) introduced variables in the first week of the course, and eight MOOCs (P1, P2, P4, C1, C2, C4, J1 and J3) in the second week. Furthermore, this concept of a variable was the first programming concept discussed in seven MOOCs (P2, P5, C3, J1, J2, J5 and J6) on our list. Two more programming concepts often introduced at the start of the course were comments and reserved words. For instance, the instructor of P1 described comments, reserved words and variables as *"the atoms, the nuggets, the little pieces that make up Python."*

Moreover, we discovered a pattern in which variables are presented structurally. See Figure 1.

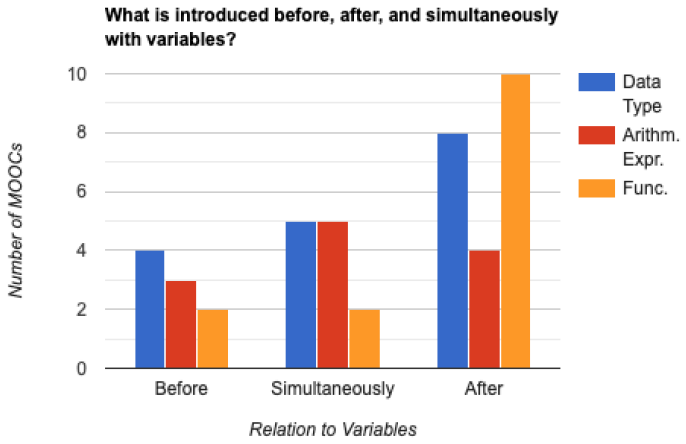


Figure 1: Introduction of Variables in the MOOCs

As expected, the programming concept of a data type is closely connected to variables, as the variable's data type determines the values it may contain. We observed that four MOOCs (P6, P7, C1 and J4) introduced data types shortly before variables, and five MOOCs (P2, P4, C2, J2 and J5) introduced data types simultaneously with variables. The remaining eight MOOCs (P1, P3, P5, C3, C4, J1, J3 and J6) introduced data types after variables. Another programming concept often introduced with variables is expressions, specifically arithmetic expressions. An expression is a combination of variables, constants, and operators that can be evaluated to a value. Our analysis observed that this concept was presented alongside variables in five MOOCs

(P1, C3, J2, J3 and J6) with three MOOCs (P6, P7 and C1) introducing expressions before variables and four MOOCs (P2, P4, C4, J1, J3 and J5) introducing expressions shortly after variables. Lastly, we noticed that two MOOCs (P3 and P5) introduced variables together with functions.

Another interesting observation is that most MOOCs (14 out of 17) explained the programming concepts of control flow, functions, and loops later in the course after introducing variables. However, two MOOCs (C1 and C2) introduced these concepts before discussing variables. In contrast, the instructor of P4 interestingly remarked on the following when introducing variables, *"Variables are possibly the most fundamental element of programming. There really isn't much you can do without variables. ... Nearly everything we do involves manipulating variables in some way. 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."*

4.2 Definitions for Variables and Teaching Strategies

This section will answer the following sub-question: *What definitions and teaching strategies are used for explaining variables?*

4.2.1 Variable Definitions

The instructors presented numerous possible definitions to introduce the concept of a variable. Table 2 contains a list of definitions provided by instructors from the seventeen selected MOOCs.

MOOCs	Variable Definition
P1	*"A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable name."*
P2	*"So now we have two variables basically storing the data that the user entered."*
P3	*"A variable is just a container for some value inside of a computer or inside of your own program."*
P4	*"Alphanumeric (letters and numbers) identifiers that hold values, like integers, strings of characters, and dates."*
P5	No exact definition is given for variables
P6	*"Variables are symbolic names for or references to values or pieces of information. You can use variables to store all kinds of data and functions."*
P7	*"One way to think about variables is to think of them as names that allow us to save values and refer to them later."*
C1	*"Variables, like in math, that let you store values temporarily."*
C2	*"One could think of variables as little memory boxes. These memory boxes contain the things we'd like to remember and are stored in the computer's memory."*
C3	*"Programs track most of their state in variables—you can think of a variable as a box that stores a value."*
C4	No exact definition is given for variables
J1	*"A variable represents a piece of computer memory that can store a value."*

J2	No exact definition is given for variables
J3	*"A place in memory that stores some value. Variables have names (or identifiers) so that you can properly refer to their value and location in your code."*
J4	*"A variable is a name that refers to a value."*
J5	No exact definition is given for variables
J6	No exact definition is given for variables

Table 2: Overview of variable definitions in the MOOCs

We can see that twelve of the seventeen MOOCs provided definitions for variables. We will now review the given definitions and examine the responses to the four questions offered in the previous chapter.

The first question was: *What do variables do?* We recognise the following keywords: "store" (P1, P2, P6, C1, C2, C3, J1 and J3), "hold" (P4), and "save" (P7). In Table 2, these keywords are highlighted in red. The second question was: *What do variables store/hold/save?* We identify the words "value" (P3, P4, P6, P7, C1, C3, J1 and J3) and "data" (P1 and P2). These keywords are highlighted in blue. The third question was: *Where do variables store/hold/save values/data?* The word we are looking for is "memory" (P1, C2, J1 and J3), highlighted in orange. Finally, the last question concerns how we might refer to these variables; the term we search for is "name" (P1, P6, P7 and J3), highlighted in green. We now observe that only two MOOCs (P1 and J3) provided definitions that answered all four questions.

4.2.2 Teaching Strategies

We also examined what teaching strategies instructors employed to teach the concept of a variable. Table 3 shows an overview of the teaching strategies and their presence in the MOOCs.

	P1	P2	P3	P4	P5	P6	P7	C1	C2	C3	C4	J1	J2	J3	J4	J5	J6
Teaching Strategies																	
Visualisation	X				X					X		X		X	X		X
Analogy				X	X				X	X		X		X			X
Link to Mathematics			X	X									X				
None		X				X	X	X			X					X	

Table 3: Overview of teaching strategies and their presence in the MOOCs

As shown in Table 3, eleven of the seventeen MOOCs employed a specific teaching strategy to explain the concept of a variable. In general, three distinct teaching strategies can be identified: only visualisation (P1 and J4), analogies, combined with or without visualisation (P4, P5, C2, C3, J1, J3 and J6), and link to mathematics (P3, P4 and J2). The following sections will describe how these teaching strategies were used in the different MOOCs.

4.2.2.1 Visualisation

The instructor of P1 explained variables by drawing a rectangle that symbolises a spare piece of memory, describing the variable name as the label for this and placing this name on the left side of the rectangle and the contents within the rectangle. See Figure 2.

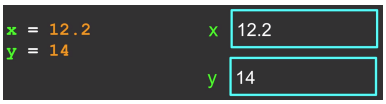


Figure 2: Visualisation of variables (P1)

The instructor of J4, on the other hand, reflected that to comprehend what programs are doing, especially in the beginning, one should write a table called a trace, which shows the variable values after each statement. The instructor explained (see Figure 3), *"So, we start out with none of the variables declared, we say int a equals 1234, then 1234 is associated with a, that's what the table shows. B equals 99, 99 is associated with b, that's what the table says. Then t equals a, takes the 1234 and associates that with t, so after that statement t is 1234. ... So now we say a equals b, so we find the value associated with b that's 99, and we can see that from the trace and then we associate that with a. Then b equals t then we have the 1234 associated with t and then we give that value to b. So that trace of these five statements tells us what that program does."*

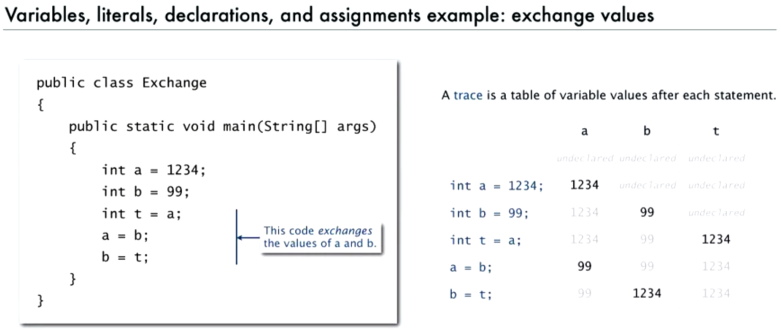


Figure 3: Trace of variables (J4)

4.2.2.2 Analogies and Visualisation

The last teaching strategy we observed was analogies often in combination with visualisation. Like the instructor of P1, the instructor of P5 drew a variable name with a box (see Figure 4). However, the instructor emphasised that the value does not go into that box but lives at a particular memory address. He drew another square with the value 20 and picked an arbitrary memory address, which he marked with an x3 (a memory address) to make it different from other numbers. He explained that the assignment statement takes that x3 and puts it in the box associated with the variable name. Hereafter, the instructor stated the following: *"So base contains x3 and in some sense what that means is that points to memory address x3 where the value 20 lives. Similarly there is a variable height. And Python keeps track of its value in that little box. And its value is a memory address.*

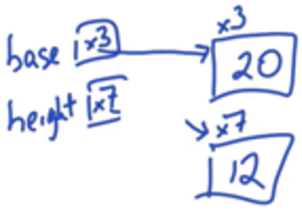


Figure 4: Visualisation of variables (P5)

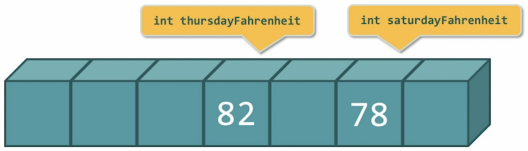


Figure 5: Visualisation of variables (J3)

I'm going to pick x7. These are 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."

The instructor of J3, on the other hand, presented a row of boxes rather than a single box to demonstrate the concept of variables (see Figure 5). The instructor drew the values 'inside' the boxes and labelled them with the variable names. Similarly, the instructor of J1 used the analogy of mailboxes (see Figure 6) instead of boxes to describe the concept of a variable. The instructor noted, *"Each mailbox is labelled by its owner (or identifier) and different kinds of mails (or values) can be put into the mailbox."* The instructor also presented a trunk of computer memory. See Figure 7. He explained that the current value of a variable could be retrieved by referring to its name. In the illustration, an arrow was drawn from the variable name to this piece of memory.

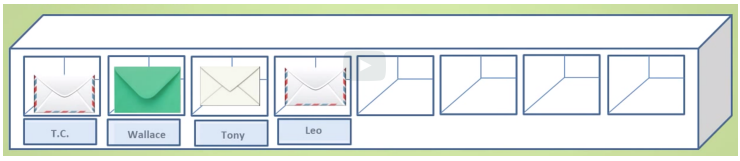


Figure 6: Mailbox analogy for variables (J1)

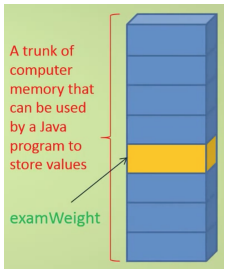
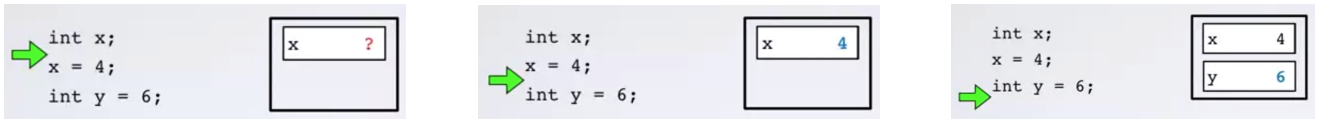


Figure 7: Visualisation of variables (J1)

Moreover, the instructor of C3 emphasised that students of this particular MOOC have to learn how to draw pictures of exactly what is happening according to a set of rules. As the instructor remarked, *"Once you can do these things, writing code will come easier. And when you do have a situation when your code does not do what you expect, you'll have a set of tools you can use to investigate your program to see what it is doing."* After this, the instructor explained variables by describing three different statements: declaration, initialisation and one statement combining the two (see Figure 8). She created a box labelled with the variable name for the first statement. Since the variable is uninitialised, she placed a question mark in the box for its value. The instructor illustrated that executing the second statement will put a value in the previously created box. As a result, the question mark disappears. Lastly, she explained that executing the last statement will create a box for this new variable name and put a value in that box.

Similar to the instructor of C3, the instructor of J6 explained variables by two statements: declaration and one statement combining declaration and initialisation. When the first statement was



(a) Line 1 is executed (b) Line 2 is executed (c) Line 3 is executed

Figure 8: Visualisation of variables (C3)

executed, boxes labelled with the variable names were created, holding the values the programmer eventually decided to put in these variables. For the time being, the instructor drew question marks in these boxes. When the latter statement was executed, a box was created, and the value was immediately assigned to this variable name. The instructor demonstrated this by putting the value in the box for the variable.

Last but not least, C2 did not offer any drawings since the instructors of this course used the CodeCast environment in their course. One could listen to the audio and see a panel with the label 'variables' on the left of the screen, a panel with the source code on the right, and another panel with the terminal underneath these two panels. The instructor described variables as follows: *"Right now, what we'd like to memorise is an integer value. And suppose I want to memorise 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."*

Another noteworthy observation is that instructor of P4 compared variables as questions and values as answers, as he described: *"The variable sticks around, while the value changes. There are a few principles that go along with this though. If you're ever confused about what a variable means, treat it as a question."* To clarify this, he provided the following example: num_cats becomes *Number of cats?* The value, then, is the answer to this question.

4.2.2.3 Link to Mathematics

Other instructors had a different approach. To introduce variables, the instructor from P3 created a link to mathematics, since it is a subject that students may be more familiar with. He stated, *"Odds are almost everyone is familiar with variables from math class, way back when x, y and z. And the like, well programming has that same capability. This ability to create a variable, in this case, in the computer's memory, not just on a piece of paper. And that variable can store a value, a number, some text, even an image or video or more. ... Well, in programming, because I have a whole keyboard in front of me, I can use more descriptive terms to describe what it is I'm writing."*

The instructor of P4 also created a link to mathematics when explaining the concept of a variable. The instructor remarked, *"You're probably familiar with variables from your days learning algebra, and variables aren't really any different here than they were there. A variable is a name that holds a value. The name stays the same while the value can change. In algebra that variable was usually x, and to use that variable you'd give it a value. The equation would then do some stuff to that value, and you'd get a result for y. If you didn't enjoy math, though, don't worry. To be honest, I never fully understood concepts like variables and functions in math, until I learned them in computer*

science. You don't need to know any math to learn about computing either. To me, writing code is more like writing an essay than it is like solving a math problem."

Interestingly, the instructor of J2 took a different route. The instructor introduced programming concepts, including variables, by comparing a calculator used in mathematics to a computer. He commented, *"What would be a good way to introduce what the computer is and what are the concepts of a programming language? Rather than starting from scratch we are going to start with a device that is very familiar to you, namely, the calculator. And we're going to transform gradually the simple calculator into a computer. By doing this we will also move from a sequence of keys you type into a calculator into a computer program. So you will much better understand the concepts behind both the computer and a programming language like Java."*

The instructor considered a simple calculator with memory to explain variables. So there are some keys for 'storing' or 'getting' values from this memory. He continued that memory allows us to store a value for future use. For example, he explained, *"A memory might hold a value, and there might be operations associated to it, like MS to store a value, and MR to restore it or to recall it. Sometimes there's even a third key, MC for memory clear, but we don't need that. And I'd like to call the two keys to 'set' and 'get' like this, 'set' to set or store a value, and 'get' to get or restore it. ... Now this memory just holds one number. So there might be additional memories for storing additional values. We would have, therefore, several, each with its own name, M1, M2, et cetera. For the moment, these names will be pre-defined. But rather than having these awful names M1, M2, et cetera, we would rather, for the moment, call them x y, as we're used to in math."* At this moment, the instructor introduced the concept of variables; he called x and y variables instead of memories. He continued: *"Imagine that we have a 3 on the display, and that we press the set key of variable x. The value 3 would be then stored in variable x. And the display could show something like x equal 3 semicolon, in order to record what we just did. We say that we have assigned 3 to variable x, and record this x equal 3 in the assignment statement. ... We can think of expressions in these calculators as statements by considering that the display can also be seen as a variable, a variable with direct input."* To summarise, the instructor of J2 introduced variables by gradually transforming variables in calculators into variables in computers.

4.3 Variable Role - Variable Naming - Variable Assignment and Time Duration

This section will answer the following sub-question: *What topics do instructors introduce concerning variables, and how much time are they given?*

During our analysis, we identified three essential topics: variable role, variable naming, and variable assignment. An overview of the topics that instructors introduced concerning variables and their presence in the MOOCs is shown in Table 4. In the following sections, we will describe what was explained regarding these topics in the various MOOCs.

	P1	P2	P3	P4	P5	P6	P7	C1	C2	C3	C4	J1	J2	J3	J4	J5	J6
Variable Role																	
Storing Values	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Flexibility									X								
Variable Naming																	
Legal Characters	X			X	X				X	X	X	X	X	X			
No Reserved Keywords				X								X	X	X			
Case Sensitivity	X				X	X			X	X		X					
Mnemonic Name	X	X		X			X	X	X		X	X	X				
Naming Convention	X	X		X	X			X	X		X	X	X	X		X	
Constant Variable							X	X					X				
Variable Assignment																	
Assignment Statement	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
'Equality' Operator	X		X	X	X			X	X			X			X		
RHS before LHS	X	X	X	X	X			X				X	X	X			
Same Variable on LHS and RHS	X					X						X	X	X			
Variable Without a Value				X					X	X	X		X				X

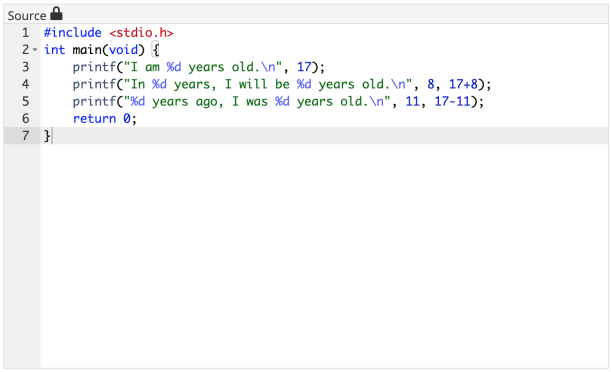
Table 4: Overview of topics concerning variables and their presence in the MOOCs

4.3.1 Variable Role

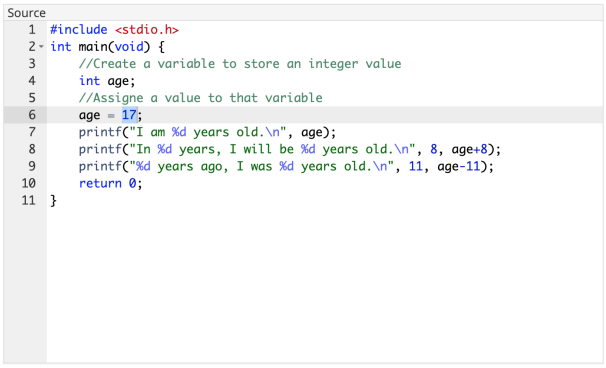
As seen in Table 4, all seventeen MOOCs discussed the main use of variables, which is variables store values. For instance, the instructor of P7 stated, *"So far we have learned how to use numbers, write some arithmetic expressions, calculate other numbers doing that. But you may have noticed that we can't actually save the results of anything we do, all we can do is print it. Well we need to fix that, okay? In order to actually write any kind of program, we need to be able to actually save the results and use them again and do something with them, right? ... Variables are a way to save your values, so that you can save them later."*

Furthermore, C2 was the only MOOC with a simple demonstration of why variables are so helpful (see Figure 9). The instructor first provided a simple program without the use of a variable. She ran the program to see what the output of this program was and expressed that she wanted to memorise the number seventeen, so she preceded by creating a variable for this number and named this variable "age". The instructor then replaced the number seventeen everywhere in her program with this new variable and ran the program again, and the same output appeared on the terminal. She then explained why this was so valuable; the instructor pretended she was no longer seventeen, as shown in line six of the figure, but twenty-nine years old. Thus, the assignment value had to be changed. She changed the seventeen to twenty-nine in the assignment statement on line six. That was all she had to do, she explained. She emphasised that she only needed to alter the variable in

one location in the program and that when she ran it, the variable was replaced with the number twenty-nine everywhere.



(a) Program without the use of a variable



(b) Program with the use of a variable

Figure 9: Explanation role of variables (C2)

4.3.2 Variable Naming

We observed six primary aspects the instructors paid attention to when describing the topic variable naming. The first two aspects concern the legal naming rules for variable names, such as" legal characters" and" no reserved keywords", as illustrated under the table heading "Variable Naming". The following aspects were whether the instructors clarified that variable names are case sensitive, explained that variable names should be mnemonic, discussed naming conventions and whether naming rules for constant variables were provided. Remarkably, eight out of seventeen MOOCs discussed two or fewer aspects, and only five MOOCs discussed four or more aspects.

4.3.2.1 Legal Characters

The first naming rule is about what characters variables may contain. For Python and C, variable names must start with a letter or an underscore. Moreover, variable names can only contain alpha-numeric characters and underscores. The same rules apply to Java; however, starting variable names with a dollar sign is here also possible. We can see from the table that only nine out of seventeen MOOCs discussed this topic. P4 provides one example of such a discussion; the instructors from the MOOC stated, *"When we're choosing variable names, there are two things we need to keep in mind– a set of rules, and a set of conventions. The rules govern what we actually can name our variables. The first rule is that variable names can only contain letters, numbers, and in Python, underscores. They can't contain spaces or special characters. Secondly, variable names must start with letters. Technically underscores are also allowed, but we generally only use those in certain situations."*

4.3.2.2 No Reserved Keywords

Another naming rule for legal variable names is that no reserved keywords can be used for variables. For instance, a few reserved keywords in all three programming languages are "if", "for", "break", and "return". Only four out of seventeen MOOCs discussed this topic. For example, the instructor

of J3 stated, *"Finally, 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."*

4.3.2.3 Case Sensitivity

All three programming languages are case-sensitive, meaning variable names must always be typed with a consistent capitalisation of letters. Six out of seventeen MOOCs mentioned this when explaining variable naming rules. For instance, the instructor of P1 expressed, *"And it's case sensitive, but we don't want you to depend on that. So spam, Spam with one upper case and SPAM. These all are different variable names, but you're not doing anybody any favour if you think that's being clever."*

4.3.2.4 Mnemonic Name

Choosing a variable name that makes sense for what a programmer uses it for is called mnemonic. This is important because programmers get to choose the names of their variables. For example, the instructor of P1 stated, *"Now, I emphasise that one of the key things about variable names is that you get to name them. And 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. And Python doesn't care whether you choose mnemonic variable names or not. And the name that you choose for a variable does not communicate any additional information to Python. ... So mnemonic variables are only for humans."*

Eight other MOOCs also mentioned that variable names should be meaningful. For instance, the instructor of J1 expressed, *"It is important that meaningful names should be used, the use of meaningless names such as x, y, z should be replaced by more meaningful names such as radius, area, scores if possible." Furthermore, complementing this, the instructor of P7 stated, "Let's go back to the example of i and j. So if you think about this a little bit, if we have indices into a two dimensional structure, maybe one choice could be row and column. And 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. So 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."*

However, not all instructors seem to be on one line with this. For example, the instructor of C1 remarked, *"Let's just talk about style for just a moment. So x and y, at least in this case, are pretty reasonable variable names. Why? Because that's the go-to variable names in math when you're adding two things together. So x and y seem pretty reasonable. I could have done something like, well, maybe my first variable should be called first number and my next variable should be called second number. And then down here, I would have to change this to first number plus second number. Like, eh, this isn't really adding anything semantically to help my comprehension. But that would be one other direction we could have taken things. So if you have very simple ideas that are conventionally expressed with common variable names like x and y, totally fine here."*

The term context may be the missing piece in this alleged discussion, which C4 clearly explains. The instructor of C4 first provided examples of variable names considered 'bad'—for example,

'grx33', '_pp_25' and 'i_am_FourWords'. The first two variable names, according to him, gave no clue of what they might be. Moreover, the last variable name mixes two conventions. He then continued, *"Other things we are not going to necessarily know if they're good or bad, let me put some up. 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. Mxyzptlk, I'm not sure if I'm pronouncing that right. Maybe that's an okay identifier, maybe not. Let me exit this. We look in our browser, and we look up these E-cartoons. Here's Mister Mxyzptlk is a typical foe of superman. Superman always has to get him to say his name backwards, which again is very hard to pronounce. I'm not sure if gets to zoominic, whatever. Again, if you were writing some movie with Superman and Mxyzptlk, it might be an appropriate identifier. X might be an appropriate identifier especially if it's something like a coordinate in the x,y plane, then it would make perfect sense. Again, we have said if we use j for integer is fine. Q may be good or bad, maybe it's obscure, maybe we're doing something like logic program, and in the logic community, p and q are typically chosen to mean Boolean variables. If we were doing logical evaluation, p and q might be quite appropriate. Again, that's contextual, so they might be either good or bad."*

To conclude, all five instructors of the MOOCs (P1, J1, P7, C1 and C4) made valid points regarding meaningful variable names. What is considered meaningful is hard to define and, as demonstrated, differs per instructor. The instructor of C4 acknowledged this and sought to clarify this by presenting a few examples of variable names and their context. The word 'context' seems to play a crucial role in determining whether variable names are necessarily 'good' or 'bad'.

4.3.2.5 Naming Convention

A naming convention provides a set of rules for choosing the sequence of characters to identify variables. The naming convention for Python and C uses a single lowercase letter, word, or multiple words. Moreover, words should be separated with underscores to improve readability. The naming convention for Java is to use lower camel case, this means that variables should start with a lowercase letter, and then the first letter of every subsequent word is capitalised (e.g. camelCase). From our analysis, eleven out of seventeen mentioned naming conventions when explaining variable naming. For instance, the instructor of P5 stated, *"Choosing good names is important, because programs can easily be used, read, and improved on for years. Every programming language has a set of conventions for how to choose a name, much like web sites 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."* Furthermore, other instructors have communicated this convention (using underscores) as a particular name. For instance, the instructor of P5 referred to this naming convention as the pothole case.

4.3.2.6 Constant Variable

The last aspect regarding naming concerns constant variables, which are values that remain unchanged throughout program execution. A standard convention in the programming world is to use all uppercase letters when assigning names to variables. Three out of seventeen MOOCs mentioned this. The instructor of C1 stated, *"And another convention in C and other languages, when you have a constant, it's often common to just capitalise the variable. Kind of like you're*

yelling, but it really just visually makes it stand out. So it's kind of like a nice rule of thumb that helps you realise, oh, that must be a constant. Capitalisation alone does not make it constant. The word const does. But the capitalisation is just a visual reminder that this is somewhere, somehow a constant."

4.3.3 Variable Assignment

We identified five crucial points that the teachers addressed when describing the topic of variable assignment. These points were 1) the assignment statement, 2) the 'equality' operator', 3) reading the right-hand side before the left-hand side of the assignment statement, 4) the same variable on both the left-hand side and right-hand side, and 5) variables without a value. Notably, eight of the seventeen MOOCs addressed two or less of these points, while only four MOOCs addressed four or more points.

4.3.3.1 Assignment Statement

Assignment statements assign a value or expression to a variable. For example, x = 10 assigns the value 10 to the variable x. Table 4 showed that all MOOCs explained the assignment statement when discussing a variable assignment. The instructors handled this point by demonstrating an assignment statement on the slides or in a programming environment. For example, the instructor of J1 noted, *"We have seen many statements with an equal sign in the previous examples. These are called assignment statements. The syntax of an assignment statement is to place a variable name on the left hand side of the equal sign, an expression on its right hand side and a semicolon at the end. The meaning or semantics of an assignment statement is to assign the value evaluated by an expression on the right hand side to the variable on the left hand side and the original value stored in the variable will be replaced."*

4.3.3.2 'Equality' Operator

The 'equality' operator is used in assignment statements to assign values to variables. However, this equal sign in mathematics has a different meaning (equality) than in programming (assigning). Eight out of seventeen MOOCs have addressed this common misconception. For instance, the instructor of P1 addressed, *"Remember that assignment statements always have a direction, right? Just think of these equal signs as having an arrow along them. In some languages, I saw a language that uses kind of an arrow, that uses a less than and a dash as the assignment, I'm like that's how it should be. Cause equal confuses us, because equal means something different in mathematics than it does in Python or other programming languages."* The programming language that the instructor of P1 hints at is R.

4.3.3.3 Right-Hand Side before Left-Hand Side

Variables should be read from right to left, with the receiving variable on the left-hand side. Instructors addressed this notion in nine out of the seventeen MOOCs. For instance, the instructor of P2 explained, *"Actually, when we're executing an assignment expression, $x = 12$ or $y = x + 5$. This expression has basically two steps when executing it. First, we evaluate the right hand side the of the assignment, in this case $x + 5$. And once we get the result of that expression, we assign it*

to the left hand side, the variable in the left hand side. So in this case we evaluate x + 5. ... So y would be equal to 17."

4.3.3.4 Same Variable on Left-Hand Side and Right-Hand Side

In programming, the same variable can be on both sides of the 'equality' operator. This notion was mentioned in five out of seventeen MOOCs. For instance, the instructor of P1 explained, *"Assignment statements means arrow and the key thing is you can almost think of this as like there is a little wall there and it completely computes this expression. This is an expression on the right hand side, gets that down to a single variable and then writes it into the memory location. And that's why it is possible to have the same variable on both sides. Because this side happens first, ignoring the left hand side and then once this side (RHS) is done, then it actually puts it into that other side. And so sometimes we'll say something like x = x + 1 and that's our way of adding 1 to x. Now in mathematics x = x + 1 makes absolutely no sense, but in programming it is one of the more common things that we do."*

4.3.3.5 Variable Without a Value

The last crucial point is the scenario when no value is assigned to a variable; what would happen in this case? Six out of seventeen MOOCs clarified this when explaining the topic of variable assignment. For instance, the instructor of P4 expressed: *"Secondly, to use variables they generally have to have values. When we use a variable without actually giving it a value, we usually trigger something called a null pointer exception. We'll talk about exceptions later. They're kind of like errors. But generally, exceptions occur when we break the rules of programming. One way to break the rules, is by trying to use a variable that doesn't have a value yet. Imagine if I asked you what colour shoes am I wearing. The variable is shoe colour, and you know the value would probably be some colour like black or brown, but you don't know the answer, so you can't answer the question. Now imagine if I were to tell you, please paint this wall the same colour as my shoes. If you don't know what colour my shoes are, you're unable to complete my instruction. That's what happens when we use a variable in programming that doesn't have a value. The program can't proceed, so it's throws up an exception and it just stops."*

4.3.4 Time Duration

We also noted how much time instructors spent teaching the two programming concepts, variables, and data types. Table 5 shows how much time (in minutes) was spent on these concepts for each MOOC. Four averages are calculated: the first for the programming language Python, the second for the programming language C, and the third for the programming language Java. Finally, a global average and standard deviation were calculated across the three programming languages.

	Time Duration of Variables (min)	**Time Duration of Data Types (min)**
P1	11:42	11:33
P2	Hard to indicate	Hard to indicate
P3	4:36	43:32

P4	15:34	35:13
P5	10:24	34:17
P6	8:46	9:42
P7	8:13	6:45
P1-P7 Average	11:31	23.51
C1	7:41	14:10
C2	28:30	20:15
C3	2:19 + 20 min. reading	4:11 + 40 min. reading
C4	17:25	39:25
C1-C4 Average	18.97	29.51
J1	19:45	3:12
J2	14:33	6:12
J3	7:11	10:51
J4	3:26	27:34
J5	1:24	1:50
J6	2:46	5:59
J1-J6 Average	8.18	9.27
Overall Average	11:31	19:40
Overall Standard Deviation	7:39	15.14

Table 5: Time duration of variables and data types

Variables and data types are two programming concepts that are frequently introduced together. However, we can observe that more time is spent describing data types than variables, precisely, eight minutes and nine seconds more in the MOOCs we analysed. Moreover, we observe that the instructors of the C programming course spent the most time teaching variables, followed by Python and in last place Java. Remarkably, three MOOCs (J4, J5, and J6) spent less than 3:30 minutes explaining variables. In Table 4, we observe that in these three MOOCs, the number of discussed points over all three subjects (variable role, variable naming, and variable assignment) is three. Moreover, the subject variable naming was not discussed at all in two MOOCs (J4 and J6). In comparison, two MOOCs (C2 and J1) that spent the most time explaining variables discussed nine and ten points over all three subjects. Moreover, the explanation of the instructors in these MOOCs was much more elaborate.

4.4 Use of Variables by Instructors

This section will answer the following sub-question: *How do the instructors use variables in example programs?*

In the previous section, we covered three main aspects that instructors focused on while describing the topic of variable naming; we will now scrutinise two of these aspects: naming convention and mnemonic name, and investigate whether instructors apply what they teach to their students.

4.4.1 Naming Convention

We have stated earlier that the naming convention of Python and C uses a single lowercase letter, word, or multiple words which are separated by underscores. The naming convention of Java uses a lower camel case, so variables start with a lowercase letter, and then the first letter of every subsequent word is capitalised.

In our analysis, we observed that almost all instructors (fourteen out of seventeen) adopted the naming convention of the designated programming language. The alternative naming convention was used in the remaining three MOOCs (P4, C2, and C3); hence in P4, C2 and C3, the lower camel case convention was utilised instead of underscores. From these three MOOCs, the instructor from P4 also shared his reasoning. He commented, *"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!"*

4.4.2 Mnemonic Name

Previously, we observed that nine out of seventeen MOOCs addressed the importance of meaningful variable names. P1 was one of these MOOCs. The instructor of P1, however, also remarked, *"Okay, so when you are beginning students sometimes if you use variable names that are too good, it's confusing. So you'll notice as I write especially in these first two chapters. Some of my code uses really dumb variable names and some of them uses really clever ones. So I go back and forth to emphasise 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."*

Table 6 shows some of the 'dumb' variable names from the beginning slides and worked exercises videos from the few first weeks. The instructor explained that he would choose more effective variables later in the course, and the instructor made the variable names 'silly' for now. The second row of the table displays the chosen variable names from later slides and worked exercises videos (towards the end of the course).

	Slides	**Worked exercises videos**

Beginning	`ddd = 1 + 4`
	`eee = 'hello' + 'there'`
	`xx = 1`
	`sval = '123'`
	`nsv = 'hello bob'`
		`xh = input("Enter hours: ")`
		`xr = input("Enter rate: ")`
		`xp = (float)xh * (float)xr`

End		
	largest_so_far = -1 count = 0 sum = 0 smallest = None found = False	num = 0 tot = 0.0 sval = input('Enter a number: ') fval = float(sval)

Table 6: Chosen variable names by the instructor of P1

Other MOOCs that have also emphasised using meaningful names instead gave meaningful names to their variables from the start. Table 7 shows some variables names from the beginning and final slides from the instructor of C2.

Beginning slides	**End slides**

int age;
int balance = 50;
int numberOfHazelnuts = 0;
int distanceTraveled = 0;
	int twenties = 166/20;
	int rest = 166%20;
	double dOne, dTwo;
	int iOne, iTwo;

Table 7: Chosen variable names by the instructor of C2

However, there were also instances where variable names with no particular meaning were chosen to explain a specific concept. For instance, when the instructor of C1 explained the concept of variable scope, the following code was given (see Figure 10).

```
int main(void)
{
    int foo = 4;
    foo = triple(foo);
}

int triple(int x)
{
    return x *= 3;
}
```

Figure 10: Example code from the instructor of C1

The variable name "foo" is used by programmers as a placeholder and serves only to demonstrate a specific concept.

The last observation involves variable naming for numbers. Almost all MOOCs used single letters, such as a, b, c, x, y, and z, to name their variables with numbers as values. This practice most likely comes from mathematics, where using single-letter variables is the norm. Two MOOCs (P4 and J1), on the other hand, gave more descriptive names (see Table 8).

Variable names (P4)	**Variable names (J1)**

`aNumber = -2`
`aDecimal = 7.1`
	`int aInt = 10;`
	`float aFloat = 10.0f;`
	`double aDouble = 10.0;`

Table 8: Variable names for numbers as values (P4 and J1)

To conclude, the use of variables by instructors varies greatly. Each instructor has their own naming style and motives for doing so.

4.5 Differences in Teaching of Variables between Python, Java and C

This section will answer the following sub-question: *Is there a difference in how variables are taught between Python, Java and C?*

No apparent differences were seen regarding when variables were introduced in the courses, nor were there very different definitions given or teaching strategies used between the different programming languages. Nevertheless, a few things can be said regarding taught topics concerning variables. As stated earlier, C and Java are compiled programming languages, whereas Python is an interpreted programming language. Moreover, C and Java are statically typed languages, so variables first need to be declared before using the variable. The variable can only store values of that type. This is not the case with Python, where variables are dynamically typed, so no declaration of the type is needed when assigning a value to a variable in Python. We observed that the statically typed languages paid much more attention in their explanation to assigning a value to a variable before using this variable, for example, in a print command. For instance, the instructor of J6 explained, *"But in this example, we have not explicitly provided any initial values. In some languages, such as C, no default value is ever provided when you declare a variable. Meaning, you get undefined behaviour if you use an uninitialised variable. This is such a common and significant problem that Java provides two solutions: an initial default value of zero is given to instance variables, or an explicit error is given for using a local variable before initialising it."*

Moreover, the instructor of C2 even made a separate video on correcting common syntax errors involving variables. The instructor commented, *"Let me show you the most common errors when you use variables with a compiler."* The first error the instructor described was forgetting to provide the type definition. The second error he demonstrated was forgetting the definition, so he did not assign a value to the variable, leaving the program with only a variable declaration. The instructor then tried to use this variable in the print command, which resulted in an error.

Lastly, as stated earlier, we observed that the instructors of the C programming course spent the most time teaching variables, followed by Python and in last place Java (see Figure 5). Since Python is a statically typed language, we anticipated instructors to spend more time on variable naming, particularly choosing meaningful names. We expected this because Python lets programmers assign any value to any variable, and the variable gets its type from the value. When the variable is used later in the code, determining the value of a variable might become more challenging if the variable name does not reflect what it represents. Our expectation, however, was not entirely confirmed in the analysed MOOCs since three out of six MOOCs did not teach their students to choose meaningful names. P3, for instance, did not explain the topic variable naming at all.

5 Discussion

This chapter discusses the presented results concerning our analysis and describes the interpretations and limitations of our findings.

5.1 Summary and Interpretations

5.1.1 Introduction of Variables in the Courses

According to our results, the programming concept of the variable is introduced in the first two weeks. Nine MOOCs introduced variables in the first week of the course, and eight introduced this concept in the second week. This result does not surprise us as variables possibly represent the most fundamental concept in programming, and many programming concepts, including control flow and loops, expand on this concept. Moreover, we detected a pattern in which variables are presented structurally. The concept of a variable is often introduced together with data types (five out of seventeen MOOCs), shortly before (eight MOOCs), or shortly after (four MOOCs). Other programming concepts that were often introduced with variables were arithmetic expressions and functions.

Remarkably, two MOOCs, presented programming concepts, including control flow, functions, and loops, before discussing variables. Since variables may be viewed as a significant building block, it may have been more reasonable to describe them first, as most MOOCs (fourteen out of seventeen) did. However, it may not be entirely unusual for instructors to demonstrate where variables can be used in programs as if showing the bigger picture first.

5.1.2 Definitions for Variables and Teaching Strategies

Regarding provided definitions for variables, we observed that twelve out of seventeen MOOCs provided definitions for variables. We have noticed that instructors try their best to simplify explaining variables, for example, by not going in-depth about where variables are stored and what happens inside the computer when a program is executed. In the provided definitions, the word "memory" was only mentioned in four of these definitions. Another essential word, "name", was only mentioned in four definitions. In our research, we posed four questions that we considered essential to have in the definition of variables:

1. What do variables do?
2. What do variables store/hold/save?
3. What do variables store/hold/save value/data?
4. How can we refer to variables?

However, only two MOOCs answered all four questions in their definition of variables. Interestingly, only one MOOC, P3, used the analogy of a container, as explained by Waguespack [WJ89] in his system of visual metaphors, to explain the concept of a variable.

We also examined the instructors' teaching strategies to explain the abstract world of variables. In general, three distinct teaching strategies were identified: two MOOCs used visualisation, three MOOCs linked variables to mathematics, and five MOOCs used analogies combined with visualisation. Of the MOOCs that used analogies, four used the analogy of a "box", and one used the analogy of a "mailbox". Doukakis et al. [DGT07] already found that many instructors use the "box" analogy and warns that it might lead to misconceptions. These analogies can be helpful if explained clearly; however, instructors must ensure that not everything about boxes applies. For instance, they have to teach students that variables can only contain one value at a time [HSAS18], so if two consecutive assignment statements are given, as the following example shows:

```
x = 1
x = 2
```

Instructors should then ensure that the box with the variable name x now has the value two, so the value one is replaced. This is one of the misconceptions that Doukakis et al. [DGT07] hinted at. Moreover, another point the instructor should clarify to their students when using the "box" analogy is that values are copied. Consider, for instance, the following example code:

```
x = 1
y = x
```

In this case, the value one is copied to the variable y. So, now both variables contain boxes with the value 1. Students might confuse this with moving a value from one box to another, leaving the box drawn for variable x empty. Some instructors from our analysis, for instance, the instructors of P5 and J1, have attempted to explain this, but unfortunately, not all MOOCs have noted this.

In our analysis, we also discovered a new teaching strategy that had not previously been covered: instructors create a link to mathematics when introducing variables. Moreover, the instructor of J2 introduced variables by gradually transforming variables in calculators into variables in computers. Students may be more familiar with a calculator from their daily lives. Fincher et al. [FJM+20] addressed pedagogic augmentation in their work, which is the mapping of certain aspects of a real-world system into the (pedagogic) domain of the notional machine. The authors demonstrated this by explaining variables as an analogy to parking spaces. Interestingly, the instructor of J2 explained variables using a simple calculator. Perhaps, an example that more closely resembles that of a computer, since calculators also have memory (M1, M2, M3, etc.) and buttons to save (MR) and recall (MR) values.

5.1.3 Variable Role - Variable Naming - Variable Assignment and Time Duration

All MOOCs covered the main use of a variable (storing values). One MOOC, C2, also gave a helpful example of why using variables is so beneficial. Reasonably, it would be useful if more instructors provided a similar example to their students to explain the flexibility of variables. This could serve as a good motivation for students to start using variables when writing code. Sajaniemi's role theory [Saj02] defined stereotypical uses of variables that appear regularly in programs. In our analysis, we observed that no instructors mentioned this theory when explaining variables. We noticed that some of the roles, such as 'stepper,' were mentioned implicitly. This role was usually denoted by

the variable name 'count.' Alternatively, the roles 'temporary' and 'constant' were mentioned, with the former being shortened as temp in variable names. Even if some of the roles were mentioned implicitly, students might benefit from being taught to the other roles as well. All the more so because only ten of these roles are essential in covering 99 per cent of all variables. Experiments [SK05] have demonstrated that using roles of variables offers students a new conceptual framework, allowing them to mentally handle programs in a manner comparable to skilled code comprehenders.

Moreover, we observed that the topic variable naming was not given enough attention in at least eight MOOCs, where only two or fewer aspects of variable naming were discussed. Keller [Kel90] claimed in his study that the topic of choosing meaningful names for variables is rarely covered in programming textbooks. In our analysis, we observed that nine out of seventeen MOOCs had paid attention to this. So one can say that some improvements have been made since his paper (1990), and more instructors are educating students on this topic, but improvements can still be made. Just saying that students should choose meaningful names is not enough. Many instructors of the MOOCs gave examples of 'good' and 'bad' variable names; however, this was usually regarding the legal naming rules for variables so that the program can be executed without errors. However, no examples of 'good' or 'bad' variables were given concerning meaningful names. Only a handful of instructors explained what they meant by meaningful variable names. The instructor of C4, for example, emphasised that context also plays a crucial role in choosing valuable variable names. The term context was also acknowledged by Glassman et al. [GFSM15] in their developed user interface Foobaz.

Another topic we explored was the variable assignment. We noticed that all instructors explained the assignment statement itself. However, only five MOOCs also paid special attention to common misconceptions and difficulties regarding the assignment statement. For example, the equality operator in programming is not the same as in mathematics (eight MOOCs); the right-hand side of the 'equality operator' is assigned first, followed by the left-hand side (nine MOOCs). Furthermore, the same variable may appear on both sides of the assignment statement (five MOOCs), and what happens if a variable is not assigned a value (six MOOCs). Previous research ([Koh17, ŽMK22]) has revealed that many novice programmers develop misconceptions, particular around the topic variable assignment. Therefore, it would be advantageous if instructors addressed common misconceptions when explaining the assignment statement.

We also noted how much time instructors spent teaching the two programming concepts, variables, and data types. We observed that more time is spent describing data types than variables. One could question if this should not be switched, given that novices still struggle with the concept of a variable. Perhaps greater emphasis should be placed on correcting any misunderstandings students might develop or have. Nevertheless, because our study only looked at variables and not data types, this cannot be said clearly. However, this may be an interesting topic for future research.

5.1.4 Use of Variables by Instructors

Another sub-question we tried to answer in this study is how instructors used variables. Here, we looked mainly at how they named their variables. We observed a few interesting cases. For instance, the instructor of P1 deliberately chose silly names for his variables to clarify that Python does

not care about how one names their variables but that this is mainly for other programmers. The instructor explained that he would use more effective variables later in the course. This was indeed observed in the slides; however, when he was programming on the spot, working out of one of the exercise videos, the names he chose for his variables might not be as meaningful as he emphasised in his lecture about variables. This can be confusing or even troubling for students since the instructors have an exemplary role: instructors should set the example to choose meaningful variable names if they wish their students to do so as well. Indeed, for more experienced programmers, the names "sval" and "fval" might be obvious; however, for novices, this cannot be just assumed. Moreover, we also saw instances where instructors used not-so-meaningful names, such as "foo", to explain a particular programming concept.

Another observation was that many instructors used single letters, such as a, b, c, x, y, and z, to name their variables with numbers as values. Other instructors (P4 and J1), on the other hand, also try to give more descriptive names for these values, such as "aNumber", "aInt", and "aFloat." These variable names might be more straightforward regarding what they stand for. However, other instructors might, for instance, the instructor of C1, found it unnecessary since using certain single letters, such as i and j for columns and rows, or n for counting, is widely accepted in the programming community. In fact, an online survey carried out by Beniamini et al. [BGOF17] showed that certain letters, including the letter i, are closely connected with specific types and meanings and may, in fact, communicate meaning.

5.1.5 Differences in Teaching of Variables between Python, Java and C

Lastly, we also were interested if there were differences in how variables are taught between the three programming languages, Python, Java and C. The different nature of the languages makes this question appealing. We observed that the statically typed languages (C and Java) paid much more attention in their explanation to assigning a value to a variable before using this variable than Python as a dynamically typed language. The instructor of C2 even made a separate video on correcting common syntax errors using variables with a compiler. This can be explained by the fact that variables in Python require no declaration of the type, so when one is less likely to forget to assign a value to a variable in Python.

5.2 Limitations

This study had some limitations. First, our study only focused on free audition or entirely free MOOCs. As a result, we did not access premium content features such as assignments, practice exercises, and videos. Therefore, we may not have experienced the full experience of what the programming MOOCs could provide. This could have had (extra) influence on the fourth sub-question, which examined how instructors used variables in example programs. It would have been interesting to see how instructors named their variables on the spot; fortunately, some instructors also programmed on the spot in the lecture videos when introducing variables.

Another limitation is that we only looked at seventeen MOOCs; the majority of MOOCs (fourteen out of seventeen) are from the United States, one from Japan, one from Hong Kong, and one from Spain. Our MOOC list might not entirely represent all programming MOOCs online. We

only focused on the MOOC platforms, edX and Coursera, and these platforms are particularly well-known in the United States. However, other MOOCs are available—for example, MOOC platforms targeted for Europe, Asia, or even country-specific MOOC platforms. Perhaps, selecting a list of programming MOOCs from different MOOC platforms may have yielded different results.

6 Conclusions and Further Research

Programming is a rather complicated subject that demands special effort and particular strategies to understand and teach. Thus, studying how programming concepts are taught is essential and begins with the fundamental concept of the variable. To gain a picture of the current state of teaching variables in online education, we conducted a study to analyse how variables are taught in introductory programming MOOCs. We looked at when variables were introduced in the courses, what definitions and teaching strategies were used for explaining variables, what topics instructors introduced concerning variables, and how much time they were given. We also looked at how instructors used variables in example programs, including whether there were any differences in how variables are taught between three programming languages Python, Java and C.

Our study has shown that the basic programming concept of a variable indeed deserves special attention in programming MOOCs. We found that the definitions provided by instructors overlook essential keywords, such as 'name' and 'memory'. Moreover, we observed that many instructors use specific teaching strategies (visualisation, analogy, and link to mathematics) to explain variables. However, we found that instructors often fail to inform their students that these methods may not tell the whole picture. An additional explanation to clarify this would be beneficial. Furthermore, we discovered that two variable topics: variable naming and variable assignment require much more attention. In both topics, about half of the MOOCs did not explain naming rules and practices, nor did they address misconceptions. Lastly, instructors should realise that they serve as role models for choosing meaningful variable names. This already begins with choosing proper variable names in the lecture material.

Further research could relax some of the selection criteria we have used to select our list of MOOCs. For instance, our study only focused on MOOCs created by universities; it might be interesting to look at MOOCs developed by companies, such as Google or IBM, to see if there is a similar pattern there. Further research could so look at other programming languages, such as Scratch, R, or C++. Moreover, other MOOC platforms can be considered, and a wider variety of MOOCs from different platforms can be chosen. In this thesis, we looked at the programming concept of a variable. Further research could also focus on other programming concepts, such as data types, control flow and loops.

References

[AF17] Eran Avidan and Dror G Feitelson. Effects of variable names on comprehension: An empirical study. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 55–65. IEEE, 2017.

[BB15] Nikolina Bubica and Ivica Boljat. Programming novices'mental models. In *7th International Conference on Education and New Learning Technologies*, 2015.

[BGOF17] Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach, and Dror G Feitelson. Meaningful identifier names: The case of single-letter variables. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, pages 45–54. IEEE, 2017.

[BM83] Piraye Bayman and Richard E Mayer. A diagnosis of beginning programmers' misconceptions of basic programming statements. *Communications of the ACM*, 26(9):677–679, 1983.

[Bow21] Pat Bowden. Beginners guide to massive open online courses (moocs). https://www.classcentral.com/help/moocs, Jun 2021.

[DGT07] Dimitrios Doukakis, Maria Grigoriadou, and Grammatiki Tsaganou. Understanding the programming variable concept with animated interactive analogies. In *Proceedings of the The 8th Hellenic European Research on Computer Mathematics & Its Applications Conference (HERCMA'07)*, 2007.

[DP06] Florian Deissenboeck and Markus Pizka. Concise and consistent naming. *Software Quality Journal*, 14(3):261–282, 2006.

[DR08] Michael De Raadt. *Teaching programming strategies explicitly to novice programmers*. PhD thesis, University of Southern Queensland, 2008.

[FJM+20] Sally Fincher, Johan Jeuring, Craig S Miller, Peter Donaldson, Benedict Du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, et al. Notional machines in computing education: The education of attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, pages 21–50. 2020.

[GFSM15] Elena L Glassman, Lyla Fischer, Jeremy Scott, and Robert C Miller. Foobaz: Variable name feedback for student code at scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, pages 609–617, 2015.

[Gie17] Michelle Gienow. Code n00b: The (variable) naming is the hardest part. https://thenewstack.io/code-n00b-naming-hardest-part/, Oct 2017.

[HSAS18] Felienne Hermans, Alaaeddin Swidan, Efthimia Aivaloglou, and Marileen Smit. Thinking out of the box: comparing metaphors for variables in programming education. In *Proceedings of the 13th workshop in primary and secondary computing education*, pages 1–8, 2018.

[Kel90] Daniel Keller. A guide to natural naming. *ACM Sigplan Notices*, 25(5):95–102, 1990.

[KM06] Päivi Kinnunen and Lauri Malmi. Why students drop out cs1 course? In *Proceedings of the second international workshop on Computing education research*, pages 97–108, 2006.

[Koh17] Tobias Kohn. Variable evaluation: An exploration of novice programmers' understanding and common misconceptions. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 345–350, 2017.

[Kok20] Ilker Koksal. The rise of online learning. https://www.forbes.com/sites/ilkerkoksal/2020/05/02/the-rise-of-online-learning/?sh=28f26e472f3c, May 2020.

[LAMJ05] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *Acm sigcse bulletin*, 37(3):14–18, 2005.

[Pop22] Alexandru Pop. Moocs – what exactly are they? https://www.distancelearningportal.com/articles/401/moocs-what-exactly-are-they.html, Jan 2022.

[Saj02] J. Sajaniemi. An empirical analysis of roles of variables in novice-level procedural programs. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, pages 37–39, 2002.

[SK03] Jorma Sajaniemi and Marja Kuittinen. Program animation based on the roles of variables. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 7–ff, 2003.

[SK05] Jorma Sajaniemi and Marja Kuittinen. An experiment on using roles of variables in teaching introductory programming. *Computer Science Education*, 15(1):59–82, 2005.

[Wei22] Joshua Weinstein. How long does it take to learn coding? learn the fastest way to learn code. https://careerkarma.com/blog/how-long-does-it-take-to-learn-coding/, Apr 2022.

[WJ89] Leslie J Waguespack Jr. Visual metaphors for teaching programming concepts. *ACM SIGCSE Bulletin*, 21(1):141–145, 1989.

[ŽMK22] Žana Žanko, Monika Mladenović, and Divna Krpan. Analysis of school students' misconceptions about basic programming concepts. *Journal of Computer Assisted Learning*, 38(3):719–730, 2022.

A Appendix