Universiteit Leiden
The Netherlands

Opleiding Informatica

Investigating teachers' perceptions on student programming difficulties related to variables and teachers' strategies to help students

Emilia van der Kooij (s2632403)

Supervisors:
Vivian van der Werf & Efthimia Aivaloglou

BACHELOR THESIS

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

23/06/2022

Abstract

Novices often face difficulties when learning to program. It is important that programming teachers are aware of what difficulties commonly arise, so that they can adapt their teaching practices in order to minimize or prevent these difficulties. This qualitative study investigates teachers' perceptions on difficulties related to variables and their teaching strategies to address these difficulties. The concept of variables is considered one of the most basic and important elements in programming, so it is important that students understand this concept well. Through an analysis of seven semi-structured interviews with programming teachers, this study presents various student difficulties related to variables, their possible causes and the teaching strategies that teachers mentioned to address them. The difficulties that were found include data type errors, problems related to prior mathematical knowledge or language-specific features, and bad variable naming. Teaching strategies to address difficulties include repetition of common pitfalls, trial-and-error, print-debug and drawing memory diagrams to improve students' mental models.

Contents

1 Introduction

The concept of variables is one of the most basic and essential elements of computer programming [Pla15]. Given the importance of variables, it is crucial for students who learn programming to really understand them. In order to make students understand the concept of variables as well as possible, the programming education they receive should be of high quality. Their education should properly help them overcome difficulties they encounter and it should help to prevent students from having any new difficulties.

The current research is meant to gain insight into the practices of programming teachers, and teachers' perceptions of student difficulties related to variables. If we know what difficulties with variables commonly arise and how teachers help students to overcome them, this insight may be useful for other, perhaps less experienced programming teachers. If teachers are aware of what difficulties commonly occur, they can adapt their teaching practices as to prevent these difficulties from the start. This may then improve the quality of the programming education and in turn of the students' understanding of programming concepts. To investigate current educational practices and insights of programming teachers, the following research questions were composed:

1. **According to teachers, what difficulties with variables arise for students who learn to program?**
2. **How do these difficulties arise?**
3. **What methods do programming teachers use to address these difficulties?**

To answer these questions, qualitative data from seven interviews with different programming teachers were analyzed.

Student difficulties with variables have been studied multiple times before (see section 2.1). However, to the best of our knowledge, these difficulties and the teaching strategies to address them have not yet been researched through interviews with programming teachers.

1.1 Thesis overview

This thesis consists of the following sections. Section 2 will discuss some definitions of concepts that are used in this research, and provide an overview of results from previous research on the topic of programming difficulties that relate to variables. This section also discusses the aspects of variables that are relevant to this research. In section 3 the methodology that was carried out is discussed. This section also includes some background on qualitative analysis, to give an idea of the methods used in this type of analysis, as qualitative research methodologies will be applied in this study. Section 4 provides the results of the analysis and section 5 will discuss the obtained results. Here, we will also investigate how the obtained results relate to previous work. Section 6 discusses some limitations to this research. A conclusion with a suggestion for further research will be found in section 7.

2 Background

In this section some background and related work is presented that is relevant to this research. First, in section 2.1 a number of examples of programming difficulties for students (related to variables) will be given that were found in previous studies. Furthermore, some suggestions on how these difficulties arise and how to help students overcome them will be presented, as found in earlier work. The second piece of this background section, section 2.2, will discuss the elements of the topic of variables that are relevant to the current research. Here, we also clarify which aspects of variables this research will focus on and which aspects are not considered.

2.1 Programming difficulties and strategies to address them

Difficulties in programming have been the subject of multiple previous studies. Other researchers in computer science education have used various terms to refer to programming difficulties, such as "misconceptions" [Sor13] or "errors" [SPBK86] [QL17]. The Oxford English dictionary defines a misconception as "a view or opinion that is incorrect because based on faulty thinking or understanding" [Lex22]. Analogously, a programming misconception occurs for students when they hold a certain view of a programming concept that is based on faulty understanding. In this study, we will refer to misconceptions as a specific kind of difficulties. The programming concept that is studied in this research is the concept of variables as discussed in Python programming courses for beginners. This section will discuss a number of student difficulties and causes of misconceptions related to variables that were found in previous research.

As mentioned above, the topic of programming misconceptions for students has been the subject of many studies carried out in the past. These misconceptions vary from misunderstandings of very basic programming concepts to difficulties with more advanced elements of programming. Because the current study is concerned with novices, we will focus on difficulties that were found with basic programming concepts, and in particular surrounding the concept of variables.

2.1.1 Influence of prior mathematical knowledge

One common difficulty, as described by Du Boulay [Bou86], has to do with the confusion of variable assignments and algebraic expressions [QL17]. The application of knowledge of variables in mathematics to computer programming is problematic, because of the differences in how variables are used in these two fields. In computer programming, a variable is a memory location that has a name associated with it and it contains a value. During execution of a computer program, variables can change values. In mathematics on the other hand, variables are often used as placeholders in equations, having a name consisting of a single symbol [Sor08]. A variable only has one possible value or a strict range of possible values that needs to be found. Variables in mathematics are used more as constants and they do not change values, unlike variables used in computer programming.

In variable assignment statements, the left hand side stands for the location and the right hand side refers to a value. This is different from what students learned in mathematics, where equations are symmetrical and statements on either side of the equal-sign can be switched without any problems. From a mathematics point of view the following two statements are equivalent [Bou86]:

```
a = 2
2 = a
```

To a beginning programmer the second statement might seem just as plausible as the first one. However, in programming, the second statement is not a valid way to assign a variable.

Another problem that could arise is that students think that these two statements are equivalent [Bou86] [Sor08]:

```
a = b
b = a
```

Again, in mathematics they are, but when assigning 'b' to 'a', it does not mean that 'a' and 'b' always hold the same value (another value may be assigned to 'b' later on). The prior knowledge from the domain of mathematics [SHS18] can also cause difficulty when trying to understand, for instance, the following statement [Bou86]:

```
a = a + 1
```

In mathematics this would not be solvable, but in programming this statement is evaluated quite differently: the variable 'a' is increased by 1. This type of misconceptions on the sequentiality of code is thus caused by the prior knowledge of mathematics.

The difficulty with understanding assignment statements due to the influence of prior mathematical knowledge is easily formed. Students who begin to learn programming will have some basic mathematical knowledge. Because assignment statements in most programming languages look the same as mathematical statements, it is not surprising that novices transfer their mathematical knowledge to this new concept of assigning variables. It is important that teachers are aware of the misconceptions mentioned above, which are easily formed because of students' mathematical knowledge. Teachers should stress the difference with mathematics from the beginning, in order to prevent or minimize these difficulties for students: once a student has formed a misconception, it can be difficult to get rid of it [Sor13].

2.1.2 Inaccurate mental models

A second type of misconceptions stems from the mental model that students have constructed of the computer. This mental model refers to the student's understanding of the "invisible" elements underlying the code and of what happens inside the computer while processing code [BM83]. A mental model can be incomplete or inaccurate, which can cause misconceptions on how code is executed [QL17]. For example, students may believe the computer to be more intelligent and more like a human than it actually is. This might cause the student to expect the computer to do what he or she means, instead of doing exactly what the student has coded. Pea [Pea86] named this misconception a superbug: "the default strategy that there is a hidden mind somewhere in the programming language that has intelligent interpretive powers". Beginning programmers are often surprised at how detailed the instructions for the computer have to be in order to generate the expected behavior. The misconception of assuming that the computer is more intelligent than it is can be amplified by the fact that programming languages use English keywords [Bou86]. Novices

may think that certain keywords imply something different from how the computer interprets them. For example, the term "and" may be used in the English language to join two sequential actions together, but the Boolean operator "and" as used in programming has a different meaning [Bou86].

Ma et al. [MFRW07] examined different mental models that students have of assignment and found a number of inaccurate or "non-viable" models [KPEH10]. One of the inappropriate mental models they found was the "assign from left to right" model. Some participants thought that the left hand side of an assignment statement corresponds to a value that is being assigned to the variable on the right hand side. Another inappropriate mental model Ma et al. found was where the assignment operator '=' is understood as a compare operator. The participants with this model in mind thought that the statement "a = b" means that 'a' is compared to 'b' and that such a statement is plausible to write if 'a' and 'b' hold the same value [MFRW07]. This misconception seems again to stem from the knowledge of mathematics that students already have: if "a = b" were a mathematical equation, the solution would be that 'b' can hold any value, as long as it is the same value as 'a' and vice versa. In programming, usually the double equal sign is used as a compare operator to check if two variables are equal. It is important to know the distinction between the single equal sign (assignment) and the double equal sign (comparison).

It is important that the mental model that a student has is correct from the beginning, because mental models are usually established early on in the learning process [SPBK86]. Furthermore, flawed mental models can be stubborn: "Students' misconceptions that result from an incomplete or inaccurate mental model are covert and resistant to change" [SPBK86] [QL17].

2.1.3 Problematic teaching strategies

Another cause of misconceptions around the concept of variables concerns the instruction of the programming teacher. In order to explain the concept of variables to programming novices, teachers often use notional machines. Fincher et al. [FJM+20] defined a notional machine as "a pedagogic device to assist the understanding of some aspect of programs or programming". It is used as a way of simplifying a concept or a skill, which reduces the cognitive load of the student in order to help their understanding. Cognitive load refers to the idea that the cognitive capacity of the working memory is limited [dJ10]. If a task requires too much capacity of the working memory, the cognitive load is too high and this will overwhelm students and impede the learning process. A teaching strategy should ideally lower the cognitive load and avoid cognitive overload [dJ10].

A notional machine can come in the form of an analogy. The downside to explaining concepts using analogies is the possible misapplication of an analogy by students. One example is the representation of variables as boxes [FJM+20]. A problem that could arise from this analogy is that students think that a variable can hold more than one value at the same time, because boxes can also hold multiple things at once. It could lead students to believe that assigning a value to a variable overwrites the existing value [Bou86]. Furthermore, consider the following code example:

```
a = 2
b = a
```

Students could deduce from the analogy of variables as boxes that after executing the following code example, variable 'a' is empty, while variable 'b' holds the value 2 because it has taken over

the content of 'a': they think of these operations as removing the content of box 'a' and placing it into box 'b' [Bou86].

2.1.4 Teaching strategies to address difficulties

Multiple teaching strategies and tools to help students overcome difficulties in programming have been mentioned in previous literature. One important aspect of addressing and preventing difficulties for students is that teachers must be aware of the prior knowledge that students have. If teachers know how students' misconceptions relate to their existing knowledge of, for example, mathematics, they can help students reorganize and refine their existing knowledge. In this way, teachers may be able to prevent students from forming misconceptions from the start [QL17].

One of the ways to help students develop a better mental model of what happens in the computer is by using tools, such as Python Tutor [QL17]. Python Tutor allows you to run a step-by-step execution of your code, where at each step the state of the program is visualized. This includes the variables in the global scope, the call stack with local variables and the heap with the objects and lists [FJM+20]. Figure 1 shows the interface of Python Tutor with the visualization it creates of the simple program on the left hand side. Teachers can also draw visualizations by hand to help students get a better understanding of the execution of a program. Visualizations can make the dynamic aspects of a program tangible and can make students more conscious of what they are doing [Sor13]. Teaching with accurate visualizations from the beginning will decrease the level of freedom that students have in constructing their own, perhaps flawed, mental model. "Aiding mental model formation as early as possible is important, as changing an ingrained but flawed mental model is more difficult than helping a model to be constructed in the first place" [Sor13].

When teachers use analogies in notional machines, they should be careful to explain them in such a way that their students do not misapply them [Bou86]. The use of notional machines can then be effective in lowering the cognitive load to help better understand aspects of programming, such as the concept of variables [FJM+20]. However, a notional machine could also increase cognitive load if it presents too much (new) information at once, or if the notional machine is hard to interpret, preventing students from learning its content [Sor12].

2.2 Aspects of variables that are relevant for the current study

The current study focuses on students who are programming novices. There are many aspects to variables, some of which are not given attention in this research, because of the background knowledge or programming experience these aspects require, in order to understand them. We are only interested in the aspects that novices encounter in the first weeks of the beginner's programming course. This will undoubtedly vary among different beginner's courses, but overall we think the basic aspects of variables can be distinguished quite clearly from the more advanced aspects. The basic aspects that we focus on in the current research are the concept of variables (including how they relate to computer memory), variable assignment, naming and data types. This research will not consider aspects of variables that have to do with object oriented programming, such as classes, inheritance and public versus private variables, which we consider to be more advanced. In the following subsections we will give some background on the aspects of variables mentioned

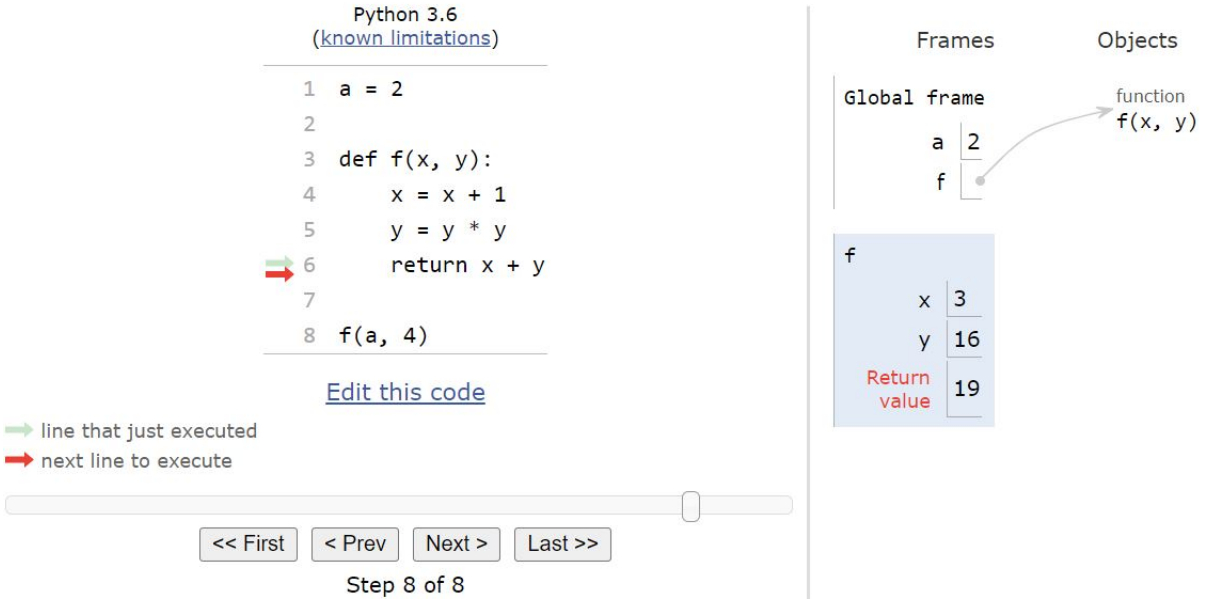

Figure 1: An example of a visualization of a simple program in Python Tutor. [Tut22]

above, that are relevant to this research. First, we briefly discuss variable naming and secondly, we provide some background on the properties of Python regarding the concept of variables, variable assignment and data types.

2.2.1 Variable naming

An essential element of a variable is its name. After all, the name of the variable is what the programmer uses when writing code. The variable name is an important part of abstraction in a program, because the programmer should be able to use the variable without having to think of its value. Therefore, the name should reflect the content of the variable continuously throughout the program. The programmer has a lot of freedom in choosing a variable name, because the name does not modify the execution of a program [GFSM15]. This also allows the programmer to choose bad variable names that do not accurately represent the content. Poor naming can cause difficulty understanding a program and can be misleading. It can make it hard to understand one's own code later on and misleading names can be harmful, especially when others need to understand and build upon the code [ADPAG13]. For novice programmers, bad variable names can be an extra obstacle in the learning process [GFSM15]. Teasly [Tea94] found that for novice programmers, unlike experienced programmers, naming style (nonsense names versus meaningful names) was an important factor in the comprehension of a program. Choosing accurate and readable variable names is thus important when it comes to understanding and using code.

2.2.2 Some properties of Python related to variables

Because the current research involves an analysis of beginner's courses that use the programming language Python, some background on the properties of Python will be given in this section. For

this study it is important to know how variables are treated in Python and how Python behaves in handling data types. An important notion for this research is that Python is a dynamically typed language. This is different from the popular programming languages C, C++ and Java, which are statically typed. In dynamically typed languages, type checking is done at runtime, while in statically typed languages this takes place when the program is being compiled. An important aspect of Python being dynamically typed is that it is not required to specify the type of a variable at declaration. This differs from statically typed languages, which do require an explicit data type declaration. In dynamically typed languages it is also possible to change the type of a variable by re-assigning it a value with a different type [Bae21].

Another feature of Python that is important to mention when discussing variables is that it is a strongly typed language. The opposite of a strongly typed language is a weakly typed language. Strongly typed languages have restrictions defined on intermixing values of different data types [Tec17]. Weakly typed languages apply implicit type conversion when certain operations are used, such as the following examples:

```
3 + '2'; // '32'
3 * '2'; // 6
```

When trying to do an operation with values of different data types, weakly typed languages decide what the type of the result will be. In order to do this, one of the types is implicitly changed. In the first line, the integer 3 is converted to a string and in the second line the string '2' is converted to an integer in order to produce the result. In strongly typed languages, such as Python, such statements as in the example will produce an error, because strongly typed languages do not convert types implicitly [Pyt21].

Regarding the concept of variables, it is also important to know how Python treats variables internally. In other languages, like for example C++, variables have a reserved space in memory of a certain size, where the content of that memory location can change during the execution of a program. In these languages, variables can be seen as containers that store data. In Python, however, a variable is treated as a pointer to an object [Van16]. Two variables that happen to have the same value point to the same object in memory. Figure 2 illustrates how Python sees two variables, 'a' and 'b', that are both assigned the integer value 2.

Figure 2: How Python sees variables with the same value. [Onl15]

When a variable is re-assigned a new value, it will point to a different object. These objects can be of any type, which is why Python is dynamically typed, as mentioned above. There is an important distinction, however, between mutable objects and immutable objects. Immutable objects are for example integers, strings and booleans. After an immutable object is created, it cannot be modified.

Mutable objects on the other hand, such as lists and dictionaries, do allow modification. This modification can be done, for instance, by the append operation, as in the following code example:

```
a = [1,2,3,4]
b = a
a.append(5)
```

The object that variable 'a' points to is modified in the third line. Because 'b' pointed to the same object, 'b' is now also modified into the list [1,2,3,4,5]. If variable 'a' pointed to an immutable object, however, and 'a' were then assigned a new value in the third line, 'b' would still point to the same object after the third line of code, and would still be equal to the same value that 'a' had after the first line of code [Van16].

3 Methods

Here we will elaborate on the methods that were applied in order to answer the research questions. We give an overview of the recruitment and the participants in section 3.1, after which we describe the approach of the analysis in section 3.2. Because we apply qualitative analysis in this research, we also give some background information on the approach to qualitative analysis in general before elaborating on the current approach to this research.

3.1 Participants

The data was collected by means of semi-structured interviews with seven different teachers who all teach a Python programming course for beginners. These interviews were conducted by Vivian van der Werf, PhD candidate at Leiden University and supervisor of the current research. The participating teachers were recruited via personal networks of different researchers from the Programming Education Research Lab at Leiden University, as well as LinkedIn and Twitter. An invite was also distributed in the Dutch network of secondary education ICT-teachers[1].

The participants were required to speak Dutch or English and needed to be involved in teaching a novice Python programming course. The course could be taught at any type or level of education, including secondary education, higher education and adult education. In total, twelve teachers were interviewed, of which seven interviews were analysed in this study to form the basis of this thesis. Some demographics and information on the teaching contexts of these seven programming teachers are presented in Table 1. This table shows that the teachers are predominantly Dutch speaking males, teaching in the Netherlands and Belgium. The teachers are in various age groups and their teaching experience in the subject of programming varies from 1 to 20 years. Their students are mostly computer science bachelor students at university, but there is still a variety of students with different backgrounds.

[1] i&i, vakvereniging informatica & digitale geletterdheid

	gender	age group	mother tongue	teaching experience programming	teaching country	level of education	study program
T1	male	25–34	Italian	2 years	NL	university	computer science BSc
T2	male	45–54	Dutch	20 years	BE	university	computer science BSc and first year engineers
T3	male	55–65	Dutch	1 year	BE	coaching on the job	adults, diverse backgrounds
T4	male	55–64	French	16 years	BE	university	computer science BSc and engineering students
T5	male	35–44	Dutch	9 years	BE	university	computer science BSc and engineering students
T6	male	25–34	Dutch	2 years	NL	secondary education, age 14–17	VMBO3+4, HAVO4+5
T7	female	25–34	Turkish	1 year	NL	university	Technology, Policy and Management bachelor and Engineering Policy Analysis master students

Table 1: Information on the seven programming teachers and their teaching contexts. BSc and MSc stand for Bachelor of Science and Master of Science respectively.

3.2 Approach

The approach for the current research consists of the interview process and the analysis of the interviews. The interview process is described in section 3.2.1, information on the general approach to qualitative analysis is given in section 3.2.2 and section 3.2.3 describes the approach of the analysis part of the current research.

3.2.1 The interviews

The participants were interviewed by Vivian van der Werf through video calls on Microsoft Teams. Every interview lasted about an hour. The interviews were semi-structured, loosely following the interview format that can be found in Appendix A. The main topics discussed in the interviews were the teaching practice of the teacher, student difficulties and the general perceptions of the teacher concerning variables. The questions were open-ended, and because of the semi-structured

nature of the interviews, there was room for discussion and follow-up questions. Although the intention was to ask every question, some interviews were running out of time. In this case, the interviewer intuitively chose the most relevant questions per teacher, to get the most out of the interviews. This type of interview provided the opportunity to thoroughly go into the topics that were discussed and to let the teachers elaborate as much as possible.

All the interviews were transcribed using a combination of oTranscribe and Microsoft Word by the interviewer and the author of this thesis. We made sure to transcribe the interview as precisely as possible, but cleaning up the grammar here and there and without including every filler word or pause. This is called an intelligent transcription [Del22a]. We also anonymized the interviews in the transcriptions. This means that, for example, any mentions of university or teacher names were replaced by "university" or "name" in the transcription to ensure the privacy of the teachers.

3.2.2 Qualitative analysis in general

Qualitative research is research based on non-numerical data, unlike quantitative research. This non-numerical data can be extracted from, for example, interviews, observations or documents. In quantitative research, the common order of execution is to start with one or more hypotheses which are then tested based on a data set, applying statistics to be able to draw conclusions regarding the hypotheses, usually to generalize the findings to a certain population. In qualitative research, on the other hand, it is common to start with a set of data from which conclusions are derived, without testing any hypotheses formulated beforehand. The goal of qualitative analysis is not to generalize, but rather to gain a deep understanding of ideas, experiences or opinions. The data are thoroughly investigated to see what kind of patterns and conclusions can be extracted. Qualitative data can be used, for instance, to develop a new theory or to get an overview of a certain situation by describing it based on the analyzed data [MHS14] [Del22b].

Miles, Huberman and Saldaña [MHS14] describe three main steps of qualitative data analysis: (1) data condensation, (2) data display and (3) conclusion drawing/verification. We followed these three steps in the current qualitative research methodology (see section 3.2.3). Data condensation is the process of selecting and organizing the data in such a way that conclusions can be drawn. This can be done through summarizing, coding and developing themes from the data [MHS14]. Especially coding is an important method in analyzing qualitative research. This is a process of categorizing the qualitative data in a systematic way, which provides structure and makes it easier to derive themes and patterns from the data [Del22a]. Codes are essentially descriptive labels to which excerpts from the qualitative data are assigned. There are two distinct ways of qualitative coding: inductive coding and deductive coding [Del22a]. Inductive coding is a bottom-up and exploratory approach, where you start with the data and group them into themes, without having determined the codes you want to derive beforehand. Deductive coding is a top-down approach where a set of codes is determined before going through the data to assign excerpts to the corresponding codes. You could also combine inductive and deductive coding, starting with a set of codes and then adding new codes while going through the data. In the coding process, it is common practice to do multiple rounds of coding. After the first round, you can organize the codes into categories and subcodes. Then, after more rounds of coding, you eventually turn the codes and categories into a

final narrative [Del22a].

The second main step of qualitative data analysis, according to Miles, Huberman and Saldaña [MHS14], is data display. This is a visual representation of the qualitative data, which helps to provide a deeper understanding of the data. As long as it is a systematic display of the condensed data, any format is possible. Miles, Huberman and Saldaña propose using matrices, where the data are organized into rows and columns, and networks. A network is a collection of nodes and edges that link aspects of the data together and it allows you to visualize relations between data elements. This work follows the advice of Miles, Huberman and Saldaña of using matrices to analyse and present data in an organized manner. The last main step that Miles, Huberman and Saldaña describe is drawing and verifying conclusions. These conclusions can be formed while still collecting the data and become more explicit as the analysis proceeds. During the analysis, the conclusions are also verified [MHS14].

As follows from above, the analysis of qualitative data is not a linear process, but rather an iterative one. Data condensation can lead to ideas on how to display the data. The process of displaying the data may lead to new ideas to further condense the data, or preliminary conclusions may be drawn. Considering these conclusions, it might be necessary to make modifications to the data display [MHS14]. Miles, Huberman and Saldaña displayed this cyclical process as a network, presented in Figure 3. A similar approach was also taken in this study.

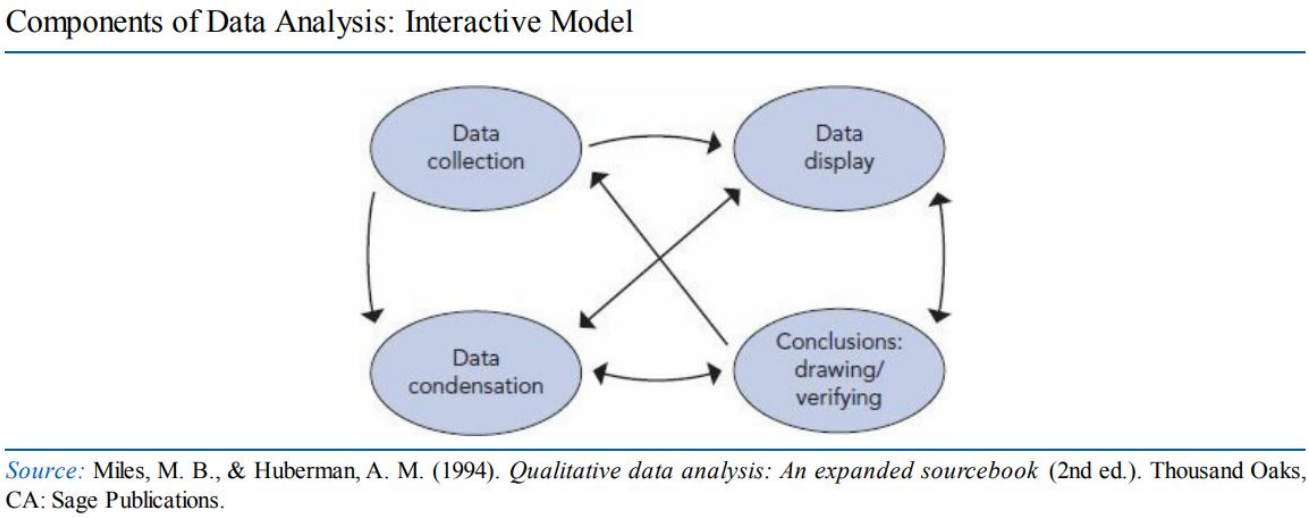

Figure 3: The cyclical process of qualitative data analysis.[MHS14]

3.2.3 The current analysis

For the current research, an inductive coding approach was combined with deductive coding. The tool used to carry out the coding process in an organized way was Microsoft Excel. This spreadsheet software is a convenient coding tool, because the matrix format lends itself to position the transcription and the codes in a readable way. We started the coding process with a separate spreadsheet for every interview. The rows in the spreadsheets represented the paragraphs of the interview. The

transcriptions were formatted in such a way that every question and answer was placed in new paragraph. The columns of the spreadsheets represented the themes that we determined to use for coding beforehand. We will go into detail about the themes we selected based on the research questions later on in this section.

Before starting the actual coding process, we assigned every row a unique identifier that implied the content of the paragraph, based on the question that was asked by the interviewer. Then, the relevant parts of the interview were selected by reading through all parts of the interview and select those parts that included information about how teachers explain variables, student difficulties with variables (including problems related to naming variables) and teachers' approaches to addressing these difficulties. These parts were essential to include because they related directly to the research questions of this thesis. We also included parts of the interview that related to the teachers' general perceptions concerning variables, because they might contain interesting information regarding the current research questions as well. Before starting the first round of coding, we determined three main themes as a basis for the analysis: "misconceptions/difficulties", "teaching: what", and "teaching: how". The latter two themes were included to separate the content of what is being taught from the teaching strategies that are used. After having selected the relevant parts of every interview we started the first round of coding, using the three themes we had determined beforehand as guidelines.

While performing the first round of coding, we added some columns to the sheet for extra themes, because based on the data we found it valuable to have the option to code with more specific themes. We added a sub-theme to the "teaching: how" theme, called "notional machines". This theme was included because we founded it particularly interesting to know if the teachers mentioned using any particular notional machines as a way of helping students understand variables. Furthermore, a fourth main theme "cause of problem" with the sub-theme "new concept" was included to be able to investigate whether any misconceptions that arise have to do with the introduction of a new programming concept. In this first round, we filled the relevant columns with the main interesting points from the paragraphs. During this round of coding, it was also inevitable that some of our earlier selected interview parts were not relevant anymore, because they could not be coded according to the themes and sub-themes and were further irrelevant content-wise. These parts were thus filtered out to further clean the data set.

In the second round, we combined the filtered data from all seven interviews into one Excel file. We made three separate sheets corresponding to the main themes of this research, based on the research questions: teaching (including which topics and strategies), difficulties (including their causes) and helping students overcome difficulties. In these sheets, we placed all the quotes with their identifiers and codes corresponding to these themes. We made a fourth sheet containing all other relevant interview quotes that did not fit one of the main themes. In this round we also adjusted the themes in the columns we were using, to focus more on answering the research questions. Therefore we scaled the columns down to three themes that corresponded to the research questions: "misconceptions/difficulties", "teaching" and "cause of problem". This means that we discarded the two sub-themes we added in the first round ("notional machines" and "new concept"), because only very few quotes contained enough relevant information for it, and they could be coded with some additional information under one of the other themes. Furthermore, we merged the themes "teaching: what" and "teaching: how" into one, because the codes under these themes seemed to

overlap largely and these codes could simply be merged under the theme "teaching (topics and strategies)".

After the coding process, we proceeded with the analysis by organizing the distinct difficulties and corresponding teaching strategies that we found by making matrices, as a way of displaying the data (see section 3.2.2 [MHS14]). These matrices looked like Table 2, but more elaborate. One matrix consisted of every difficulty that was mentioned in the interviews and the corresponding cause(s) that were found. Another contained the teaching strategies that were mentioned for every difficulty. When the matrices were complete, we could describe the results in an organized way. This task involved making lots of modifications in order to come to a satisfactory way of ordering the results. After this, we were able to draw conclusions with respect to the research questions.

4 Results

In this section, we will present the results from our analysis with respect to the three research questions of this study. We found that certain difficulties that teachers mentioned could be grouped into the same category. These categories are each presented in a different subsection. Furthermore, every subsection is divided into two parts. The first part presents our results regarding the first two research questions, "According to teachers, what difficulties with variables arise for students who learn to program?" and "How do these difficulties arise?". The second part of every subsection presents the results that we found regarding the third research question, "What methods do programming teachers use to address student difficulties?". In Table 2 the different categories with their corresponding difficulties are presented. Some difficulties that we found were not really related to other difficulties. These were classified into a remaining category "other". The categories will be discussed in the order of presentation in Table 2, which presents them in a random order. We assigned every difficulty a unique ID, to be able to refer to them individually. The teachers mentioned in the following subsections are referred to by their unique ID as well, which can be found in Table 1. Table 3 presents an overview of which teachers mentioned which difficulties.

4.1 Data type errors

4.1.1 Difficulties and mentioned causes

The first category of difficulties is data type errors. The two difficulties that fall into this category are the confusion when numbers should be used as a string value (D1) and that students forget to write quotation marks around a string value (D2). Confusion about data types can often lead to type errors: sometimes students think that a variable has a certain type and they use it according to the type they have in mind, but then this does not work, because the type they have in mind is incorrect. Students can also be unaware of when to use which data type, for example what data type a certain function argument requires. Multiple teachers mentioned that type errors occur often. In fact, T1 said that most problems with variables are related to type errors: *"[...] because most of the problems that you might have with variables are related to a type error in the end of the game, almost always"* (T1).

D1: confusion when numbers are used as a string value
A specific example that T1 and T6 mentioned was that students have difficulties with types when

category	ID	description of difficulty
data type errors	D1	confusion when numbers are used as a string value
	D2	forgetting to put quotation marks around a string value
Python-specific features	D3	confusion caused by concatenating strings using the plus operator
	D4	variables with the same name referring to different parts in memory (scoping)
	D5	understanding that variables with the same value are references to the same object in memory
	D6	string formatting with the percent sign
the relation of variables to memory	D7	the idea that a variable is actually a name that addresses a memory location
	D8	confusing the order of evaluating assignment statements
	D9	the difference between reading and writing a variable from/to memory
naming	D10	confusion about the purpose of a variable because of its name
	D11	not understanding that arguments can have different names from their corresponding parameters in the function definition
other	D12	using a single equal sign when comparing variables
	D13	the difference between returning and printing a value

Table 2: Difficulties classified into categories, presented in no particular order.

working with numbers in a string. Students often think that numbers should automatically have an integer type, also when they are included in a string. T1 attributes this difficulty to a simple distraction, because it is a small mistake that is easily made. Usually, the student that makes this mistake does understand the concept: *"It's so small, and so irrelevant, that you think "oh that's 5". But it's not 5, it's the character of five"* (T1). T2 mentioned that type errors might also be easier to run into because of the fact that Python is a dynamically typed language (see section 2.2.2): a variable can change types when it is assigned a new value during the same program.

D2: forgetting to put quotation marks around a string value

The difficulty of type errors when numbers are used as strings can be related to forgetting to put quotation marks around a string value (D2). As T1 mentioned, this may also be caused by distraction and by the fact that quotation marks are easily overlooked. It is especially a common mistake when the string variable is a number, because numbers are usually stored as integers. However, forgetting quotation marks does not just occur when the string is a number, but also in

	T1	T2	T3	T4	T5	T6	T7
D1	X					X	
D2	X	X					
D3	X		X				
D4				X	X		
D5				X	X		
D6			X				
D7		X	X			X	
D8		X					
D9		X					
D10	X				X		X
D11							X
D12	X	X					
D13				X	X		

Table 3: What difficulties were mentioned by which teachers: the X indicates that the teacher mentioned this difficulty in the interview.

general. T2 said that this happens especially when students have to write code with pen and paper, perhaps because then they cannot test their code.

4.1.2 Teaching methods and mentioned solutions

According to T1, type errors, such as D1 and D2, are oftentimes just caused by a lack of attention, and as soon as you point out the mistake, students immediately see the problem: *"... it seems the majority of the students understand it and when there are these problems, they're often distractions. As soon as you point it out they're like "Oh yeah, of course. Of course I have to convert" "* (T1). However, T1 also mentioned that it is important to make sure that students pay attention to data types when they are writing code, to avoid these type errors. He tries to make students aware of possible type errors by paying lots of attention to data types during the lectures. T1 stresses throughout the whole course that checking data types of variables is important, by often asking students questions in class, such as which type a certain variable has. This helps students to form the habit of thinking of the data type every time they assign a value to a variable.

T6 mentioned that he used to give elaborate explanations on data types, but that he now only gives a brief explanation, before letting students practice with exercises where they have to use different data types. He found that it works best to let his students practice a lot and learn from their mistakes through trial-and-error. This way, they figure out themselves when to use which data types and that they have to be careful to use the right data type in order to prevent or solve type errors.

Moreover, T1 mentioned that he assesses data types explicitly by including exercises in the exam related to type errors, where students have to identify a problem presented in an example. T1 and T4 said they give students tools that allow them to check variable types themselves when they are

programming, for example applying print-debug regularly: *"I teach a lot with print-debug: I say "okay, [at] every single step, if you are not 100% sure of the value of these variables, print it, just print it" "* (T1). As another way to avoid type errors or confusion of which type a variable has, T3 and T7 suggested including the data type of the variable in the name to make it more obvious: *"[...] if I'm talking about data structures, data types, then I tend to use this structure name within the variable, [so] if it's a list, then I tend to write like number_list [...]"* (T7).

4.2 Python-specific features

4.2.1 Difficulties and mentioned causes

The second category are difficulties that have to do with various language-specific features of Python. These features are not necessarily exclusive to Python, as some of them are also found in other programming languages. It is important to understand these features when programming in Python in order to avoid certain problems, as we will see in this section. The four difficulties in this category that we will discuss are the confusion that strings can be concatenated using the plus operator (D3), that variables with the same name can refer to different parts in memory (D4), not understanding that variables with the same value are references to the same object in memory (D5) and the difficulty with string formatting using the percent sign (D6).

D3: concatenating strings by using the plus operator
The first difficulty that relates to a Python-specific feature is that students are confused that in Python you can concatenate strings by using the plus sign. T3 said that this can cause confusion for students, because the plus sign is normally used in numerical statements to add numbers, so it is counter-intuitive to use a plus sign to concatenate strings. The fact that you can also use a plus sign to concatenate strings comes from the way the plus operator was defined by the developers of the language. It is a specific feature of which you have to know how it works in Python. A difficulty that relates to this, mentioned by T1, is that students sometimes try to use a plus sign to combine a string with an integer. This produces a type error in Python, because it does not allow adding integers to strings: Python is a strongly typed language (see section 2.2.2).

D4: variables with the same name referring to different parts in memory (scoping)
Another difficulty that is related to Python specifically, according to T4 and T5, is that students can find it hard to understand that variables with the same name can refer to different parts in memory. Teachers note that this has to do with the scoping rules in Python and when a variable is global or local. T4 explained that in Python, a variable becomes local once it is assigned inside a function and otherwise it is global. Students have difficulty understanding that there can be two variables with the same name inside and outside of a function that refer to different parts in memory, even though they are used in the same program: *"[...] what are local variables within a function, as opposed to global variables, and Python, unfortunately, is [...] not very intuitive in that respect, since a variable becomes local once it's assigned in the body of a function and otherwise it's global. So this is a bit tricky and also to understand that you can have variables with the same name outside of a function and inside of a function [...]"* (T4).

In order to clarify why Python can be tricky when it comes to global versus local variables, we

included the following example of Python code.

```
1    example = 7
2
3    def f():
4        example = "hi"
5        print(example)
6
7    print(example)
8    f()
9    print(example)
```

Line 7 prints 7, line 8 prints "hi" and line 9 prints 7. In line 1, the global variable "example" is initialized and function f() prints a variable "example". However, the variable "example" that f() prints is a different variable from the global variable "example", because in line 4, a variable "example" was assigned the string "hi". Python sees this variable as a new local variable in the scope of the function. If you wish to manipulate the global variable "example" inside f() (so that line 9 also prints "hi"), then you have to use the "global" keyword when you assign "example" a new value. This can be tricky, because it is easy to forget this keyword and to think you are still manipulating the global variable. The syntax of assigning a new value to an existing variable looks exactly the same as initializing a new local variable with a name that is already in use. This is because Python is dynamically typed, so it is not necessary to specify the data type when initializing a new variable. Thus, if you want to manipulate a global variable inside a function, you have to be careful to use the "global" keyword in front of the variable name, because if you forget this you are unintentionally creating a new local variable.

D5: not understanding that variables with the same value are references to the same object in memory

The third difficulty in this category is that students have trouble understanding that variables with the same value are references to the same object in memory. This relates to the specific property of Python that variables are treated as pointers to objects (see section 2.2.2). T4 mentioned that in the beginning, when students only have to deal with simple types that are immutable, such as integers and strings, it is not crucial yet to really understand this property of Python. These simple types of variables do not have operations that can change the object that (multiple) variables refer to. However, T4 and T5 also mentioned that when, for instance, lists are introduced, which are mutable types, it does become important to know that Python does not treat variables as containing a value, but rather as containing a reference to an object. In this regard, the teachers found that students struggle to understand the following example of code (also given in section 2.2.2):

```
a = [1,2,3,4]
b = a
a.append(5)
```

Some students think that after the third line has been executed, 'b' is still equal to the list [1,2,3,4]. However, because lists are mutable objects and 'a' and 'b' refer to the same object, 'b' is also changed in the third line of code and is then equal to the list [1,2,3,4,5]. This shows

that it important to be aware of this specific feature of Python in order to avoid this type of difficulty.

D6: string formatting using the percent sign
The last difficulty, mentioned by T3, in the category of Python-specific features is that students find it hard to understand the string formatting syntax where the percent operator is used. In Python you can use the percent operator in a string as a placeholder for a variable that you want to insert in a certain part of the string. The letter after the percent sign specifies the data type of the variable to be inserted. Consider the following example:

```
a = 7
b = "two"
c = 3.5
print("%d divided by %s equals %f" % (a, b, c))

// output: 7 divided by two equals 3.5
```

The 'd' is used for an integer, the 's' for a string value and the 'f' denotes a floating point value. This structure originates from the C programming language. T3 indicates that students do not understand why they have to specify these different letters in order for it to work. He also thinks that if the students had experience with programming in C, they would have immediately understood this formatting structure. When students see this structure for the first time, the syntax can be overwhelming because it looks complicated: *"If you start to write a string and then add %s and %d and another % sign, and then brackets with variables between them et cetera ... That is too much information. So that can make it quite complicated"* (T3, translated).

4.2.2 Teaching methods and mentioned solutions

For difficulties that are related to Python-specific features, we found a pattern where teachers do not want to go into much detail about the underlying workings of Python, because their students are only beginners. Instead, they just let the students practice with exercises so that they learn how to handle and use these specific features. Teachers find it more important that students know how to use these specific features in their code than that they try to understand how Python exactly works, as we will see in this section.

D3: concatenating strings by using the plus operator
T1 mentioned that he pays lots of attention to avoiding this difficulty in class. He found that some students think they can use the plus operator in more cases than Python allows. He uses a separate PowerPoint presentation to explain the print function, where he stresses when you can and cannot use the plus sign, so for example "print(integer + integer)" or "print(string + string)" works, but "print(string + integer)" does not work. This way, by repeating what is and what is not allowed in Python, he hopes to stress it enough so that students will not make mistakes with this. However, he also mentioned that even after having showed students this PowerPoint presentation, they still make mistakes with which types you can and cannot use the plus operator. T1 attributes this to the students being distracted or inattentive.

D4: variables with the same name referring to different parts in memory
Students sometimes struggle to understand that variables with the same name can refer to different parts in memory. T4 tries to help students overcome this difficulty by discussing scopes and the difference between global and local variables in more detail in the course, and by allowing students to practice with it in exercises. Students are also evaluated on their understanding of scoping and global versus local variables in exam questions: *"[...] that's typically one of the things that we try and question them over, so we'll have a little example program where they're asked, "what's the value of this, and that variable after this and that step of the program", to see whether they understand that well"* (T4).

D5: not understanding that variables with the same value are references to the same object in memory
Some students do not realize that in Python, variables with the same value point to the same object. In the beginning, when discussing simple immutable types, teachers usually present variables as boxes that contain a value, even though this is not exactly how Python works: *"There is one element we kind of leave on the side, but which is important in dealing with variables, in particular in Python. It's the fact that a variable can contain not a value, but a reference to an object and that as it turns out in Python, everything is an object, so in the end all the variables contain not integer numbers, but actual references to integer objects. But in the case of integers, those are immutable objects, they can be handled essentially as the values themselves"* (T4).

T5 mentioned that he tries to avoid saying that values are stored *in* a variable, because in Python the part of memory that a variable refers to can change, instead of what is inside a fixed part of memory. However, T5 said that he is not sure into how much detail he should go about how Python works precisely, because it could complicate things too much for students who have just started to learn programming: *"And to present variables in such a way that students properly understand how variables work in Python is also important, but a very difficult discussion I find sometimes is how much attention you should give to what are these kind of small differences between how variables are treated in one language versus the other. That could make things too complex when you're just learning to program in the beginning"* (T5). T5 did mention, however, that he tries to prevent students from forming misconceptions about how Python treats variables by always being consistent with in explanations. He makes sure to treat simple (immutable) types the same way as the more complex, mutable types by strictly separating the values and the variable names in memory drawings. With such a uniform notation, he tries to prevent that students see variables as boxes of which the content can change. Still, it was mentioned that students have problems later on, when they have to understand how Python deals with mutable objects such as lists, as in the code example given in section 4.2.1. T5 says that he tries to help students understand this better by asking them questions during the lecture and discussing the matter in more detail if students give the wrong answer.

D6: string formatting using the percent sign
T3 mentioned that students ask questions about why this string formatting with percent signs is needed and that students find the syntax hard to read. He says that he finds it hard to explain why this structure has to be formatted like this, because then he has to go into detail about the underlying workings of Python: *"And with those formats, using %this and %f and %d and whatever.*

That can make things very difficult. The way I explain this, yeah, I do not really explain that because then it gets very complicated. So yeah, then I say "if you want to know more about that, just read a book that explains this. But for now, just assume that this is an easy way to concatenate strings" " (T3, translated). Again, the strategy here is to tell the students how to use the structure instead of going into detail about how Python exactly works.

4.3 The relation of variables to computer memory

4.3.1 Difficulties and mentioned causes

Another category of difficulties is that students struggle to understand the relation of variables to computer memory. Three difficulties were found that fall into this category: the idea that a variable is actually a name that addresses a memory location (D7), confusing the order of evaluating assignment statements (D8) and the difference between reading and writing and reading a variable from and to memory (D9).

D7: the idea that a variable is actually a name that addresses a memory location
According to T2, students who start learning to program seem to struggle to understand that a variable is a name that addresses a memory location. *"I think the students in their first year maybe struggle still a bit too much with the concept of what is a variable, and the fact that it's a thing, a name that refers to something in memory"* (T2). T2 suggested that this difficulty could be caused by the students' prior knowledge of mathematics. After all, variables in mathematics are used in a different way from variables in programming and they have nothing to do with computer memory. T6 noticed another error that could be related to students not understanding that a variable addresses a memory location: he mentioned that some students use the current value of a variable in their code, instead of the variable name. For example, when they have a variable called "age" with a current value of 17, they write in their code "print(17)" instead of "print(age)". T6 explained that students who make this mistake do not realize that the use of variables makes a program flexible.

D8: confusing the order of evaluating assignment statements
Another difficulty in this category, mentioned by T2, is that students confuse the order of evaluating assignment statements. The problem belongs in this category, because if students are not aware that a variable refers to a memory location that contains the value it is given in an assignment statement, it might not be obvious what the order of evaluation should be. T2 also attributes this problem to mathematical knowledge because in mathematics the equal sign has a different function from the equal sign used in programming. In mathematics variables are not assigned values and they are treated more as constants, without changing to different values (see section 2.1.1). T2 suggested that the confusion with mathematics might even be strengthened by the fact that simple programming exercises often involve mathematical functions, such as the Fibonacci-function or the factorial function. This might give students the impression that they can use their prior mathematical knowledge when programming in more aspects than what is actually the case.

D9: the difference between reading a variable from and writing it to memory
Another difficulty in this category is that students struggle with the difference between reading a variable from memory and writing a variable to memory. T2 mentioned that students have difficulty

understanding that reading a variable is getting it from memory and that writing is putting the variable into memory. This is again related to not having a clear understanding of what goes on inside the computer memory. Students can struggle to form a picture of the difference between the processes of reading from and writing to memory.

4.3.2 Teaching strategies and mentioned solutions

We will address the teaching strategies that teachers use for the difficulties in this category altogether, instead of discussing them for every difficulty individually. We do this because the difficulties in this category are very much related to each other, in that they clearly have a common underlying cause.

It was found that an influential factor to the struggle of understanding what happens in computer memory is the students' prior knowledge of mathematics. Teachers try and help students to shift their attention away from variables in mathematics and more to the concept of variables as it is used in programming. T2 does this by explaining that a variable is a name that addresses a memory location and use memory drawings to draw students' attention to this. He tries to prevent this difficulty for students by thoroughly explaining the concept of variables in programming from the beginning of the course: *"And then to introduce it, we use this picture as it should be with a nice memory model. We use the metaphor that basically, a variable is a name which references some address in the memory of your computer and on that address you store a certain value. So the variable is not a value. You have to be careful [...] that the equal sign is not a comparison, it's really an assignment, which means you're going to store something at a certain address in the memory of the computer. To be able to address that memory, rather than using a memory address, which is a very obscure thing, we name it, and so that's essentially what the variable is"* (T2).

T2, T3, T4 and T5 all mentioned that they extensively use memory drawings to provide students with a clear picture of what happens in the computer memory when variables are dealt with. If students can visualize what goes on inside the computer memory in their head or on paper, this will also help them to understand more complex memory manipulations later on in the course, for instance when classes objects are introduced: *"I have been updating this year, the entire third module with lots of memory diagrams. Because I think they become even more important when you talk about classes and objects. [...] It gets very complex, and if you don't make these drawings, you get completely lost"* (T2). T3 said that he explains that the memory of the computer looks like a matrix that contains addresses where you can store variables. This relates to the way T4 illustrates the concepts of variables in memory, by drawing boxes: *"[...] typically we'll illustrate that by drawing a box that represents the memory location that goes with the variable, with the name across it and then illustrate how this content of that box changes as the program is executed. And that's something that we try to invite everyone to do in a systematic way to understand how a program works"* (T4).

T2 and T4 also recommend students to use tools that can make memory diagrams of a specific program for you, such as Python Tutor (see 2.1.4). *"It's all because once you have that mental model in place, even if they maybe don't draw it anymore, they might still visualize it in their head. This is really what is happening inside the computer"* (T2).

4.4 Variable naming

4.4.1 Difficulties and mentioned causes

This category includes difficulties that have to do with the naming of variables. Two difficulties were found that belong in this category: confusion about the purpose of a variable because of its name (D10) and not understanding that function arguments can have different names from their corresponding parameters in the function definition (D11).

D10: confusion about the purpose of a variable because of its name
T1, T5 and T7 mentioned that students sometimes struggle to deduce the purpose or the meaning of a variable from its name. One example that was given is that in for-loops, the index that is used to iterate over the elements is usually given the name 'i'. However, students who are new to programming do not have enough experience to know that this is a common practice and are confused as to why the iterator is called 'i': *"The typical variables that I use for for-loops, so "for i in something", I use 'i' and 'j'. But I do remember this was a bit confusing for the students because they were thinking "why is it called 'i' ", is it just a random arbitrary variable name that I choose or does it have any significance, does it have to be 'i' "* (T7).

Another problem where students struggle with variable naming is that sometimes they use certain variable names in their program, and then they refactor parts of their code, after which some of the variable names do not make sense anymore. Some names will not give an accurate representation anymore of what the variables contain or how they are used in the program. As T1 puts it, *"Students start maybe giving names that makes sense, and then the code maybe was wrong after we fixed something, and then they change it a little bit and change it again. And at a certain point, those names don't make sense anymore for what they contain, then yes, at that point they get very confused [...] "* (T1). Another example of when students do not understand the purpose of a variable because of its name is when they are lazy and give their variable a one-letter name: *"Some people they're really lazy: instead of, what is clearly an index, and you could write just index. They just use 'a' because it's faster. And it's like okay, yes, but then later on, you don't know what 'a' is anymore because it's called 'a' "* (T1). T5 also noticed that students confuse what the purpose of a variable is when one-letter variable names are used.

D11: not understanding that arguments can have different names from their corresponding parameters in the function definition
A difficulty that T7 mentioned was that students find it hard to understand that when calling a function, the argument names do not have to be the same as the names of their corresponding parameters in the function definition. *"I think how you name a function argument usually creates confusion for students. How you name a function and then arguments and how, when you call that function, you can use different names, they don't have to be the same name. That requires some explanation or requires some type of effort, yeah, it takes a while for them to get that"* (T7). T7 attributes this problem to the students lacking enough experience and practice in programming.

4.4.2 Teaching methods and mentioned solutions

D10: confusion about the purpose of a variable because of its name
In the interviews, the teachers were specifically asked about how they recommend students to name their variables. All teachers mentioned that variable names should be readable, intuitive and representative of the content of the variable. In some cases, however, teachers said to use shorter, one-letter variable names in the context of mathematical problems or when writing a for-loop. As mentioned before, students sometimes struggle when short names are used. About helping students who are confused about the name 'i' for the iterator in for-loops, T7 said: *"[...] after receiving this question, I said a few times, "this is just something random, it could be anything, it could be something else." So I think with for-loops it's important to tell them, or also perhaps with while loops, you know, "it's just something random, it doesn't have to be 'i', it could be anything, it doesn't matter. It goes through all the things that you put after 'i' ""* (T7).

All teachers mentioned that they generally prefer to see longer, more meaningful variable names. However, T1 mentioned that he also want to stress that it does not matter what name you give your variable, because the code can still work. *"So I really put a stress, especially in the forums. It happens a lot that they're like: "Why is it 'i'?" [and me:] "Like I don't know. It could be "banana". I don't care", so I tell them that. On the other hand, I also tell them it has to make sense for somebody who reads. [...] So, I tend to stress on that, or like make it such that it makes sense for somebody who reads it"* (T1). T1 said he applies the same practice when students have variable names that do not make sense anymore regarding their purpose, after they refactored their code: *"In that point, I often tell them you might want to rename it. I don't oblige them to rename it because on the other hand I want to also stress that the code could work anyway. You could, again, call it "banana" and it works. You just have to know where to put "banana". If you talk about apples and bananas, and at a certain point "apple" becomes "banana", then at this point everything is flipped. I usually try to recommend that, but then at a certain point, I don't push it too hard"* (T1).

T2, T4 and T7 mentioned that, even though they prefer readable and meaningful names, they do not explicitly tell their students how to name their variables or to stick to any naming conventions. They only pay attention to variable naming implicitly, by setting the right example in code examples or through feedback on variable names in students' code during practical sessions. *"We don't specifically insist very much on how variables should be named, you know, as you should use short names or long names, that comes more naturally by example and I don't think that that's a major source of concern for the students. So they learn essentially from the examples they are confronted with [...]"* (T4).

Even though in for-loops it is a common practice to use one-letter names for iterators, T3 and T5 mentioned that here, they also prefer longer and meaningful names. This would make it easier for students to understand the purpose of the variable. Teachers said they try and set the right example for their students, by using longer and readable names in their code examples on the lecture slides, or when writing solutions to an assignment. However, T2 and T5 did also admit that they sometimes tend to use shorter variable names, which fit better on the slides, even though they prefer longer names: *"[...] so it's not because I don't know how to do it, but often as a teacher you have a limited space on the slide. So sometimes, even though you know to name your variables well, you will use x, y, z, because that fits on the slide. [...] Currently, whenever I see something like that,*

I'm changing it on my slides to use the longer names. I think we should be consistent in whatever you do, even if it's on a slide or a small example you put on the blackboard. [...] You tend to go for the most simple things, but don't expect ... They won't do it if you don't do it yourself. So that is something that I try to pay attention to" (T2).

D11: not understanding that arguments can have different names from their corresponding parameters in the function definition.
In order to make students understand that the arguments with which you call a function do not need to have the same name as their corresponding parameters in the definition, T7 gives students more time to practice. She also mentioned to give students more examples of how arguments in a function call can be used. She explains that when you give your function arguments different names from their corresponding parameters in the function definition, this will not produce an error and your code will still work.

4.5 Other difficulties

4.5.1 Difficulties and mentioned causes

There were two difficulties that could not be classified into one of the categories above. These were the mistake where students use a single equal sign when trying to compare variables (D12) and the difficulty of understanding the difference between returning and printing a value (D13).

D12: using a single equal sign when trying to compare variables
T1 and T2 mentioned that beginners often confuse the single equal sign with the double equal sign. There is an important difference, however, as was discussed in section 2.1.2. Teachers did not mention why so many students find this difficult.

D13: the difference between returning and printing a value
Another difficulty, mentioned by a T4 and T5, was that some students do not understand the difference between returning and printing a value. T4 said that he understands that this difference is not very simple and that it takes a while to get used to for the students. After all, the first example that students see in the course is usually a program that produces the result by printing it. Later on in the course, the result that is computed in a function will usually be returned. This result will usually not be printed, but will be used in another part of the program. *"So the very first thing they see is, the result is something that you print, and so when the result that you compute is now something that you have to return, well, it's a different way of proceeding. And yeah, if you have a function that returns something and then when you test your program, well you don't see the result unless you specifically print it yourself. So I think it's just a concept that is new with respect to the previous concept and so it just takes time to get used to that"* (T4).

4.5.2 Teaching strategies and mentioned solutions

D12: using a single equal sign when trying to compare variables
Because students so often make the mistake of using a single equal where a double equal sign should be used, T1 and T2 mentioned that they now try to prevent this difficulty early on in the course. T1 said that he stresses the difference between the single and the double equal sign from

the beginning, right when introducing variables in the course: *"We attract the attention on saying there's the equal sign which is assignment, but of course we also want to compare things. There is an equal equal sign so don't mistake that because that's indeed a very common mistake in the beginning"* (T1). T2 said that he tells his students to read the double equal sign as "is indeed", instead of "is". He gave his students this tip from the beginning, which seemed to have prevented the students from having this difficulty of confusing the single and the double equal sign: *"[...] I think, paradoxically, most of them don't have that much of a problem with the conditions, which usually often I expect people to because of the equal equal, for example. But they don't, [...] I don't know if it helped. Maybe they just find it easy and they got it. What I told them is to read it every time as "is indeed". Is indeed, so equal equal. It's not just "is", it's "is indeed". So that often doesn't happen as a problem in general. Both writing and reading, they're like, "oh equal equal. That makes sense. It's a condition""* (T2).

D13: the difference between returning and printing a value
T4 and T5 said that many students have difficulty understanding the difference between return and print, so they both try to prevent them from taking the habit of confusing the two, by repeating the distinction a lot. *"An ever recurring topic, in almost all of our lectures, at least for my part, is to try to repeat and repeat that print and return, by the way, are not the same thing. In Python that seems so many students confuse the two. So also here when you discuss about prints, I feel you always have to be careful that they don't take the habit of using print instead of returns or other things they should be using"* (T5).

5 Discussion

Through the analysis of data from interviews with seven programming teachers, we have gathered results with which we can now provide answers to our research questions. We will discuss the results that we found and compare them to findings from previous studies on student difficulties relating to variables. They are discussed in this section per research question.

RQ1: According to teachers, what difficulties with variables arise for students who learn to program?
We found that the student difficulties could be grouped into different categories, based on their similarity or a common underlying cause. The categories we found to be fit for classifying the difficulties were type errors, difficulties having to do with Python-specific features, difficulty understanding the relation of variables to memory, problems with variable naming and, lastly, a category for two unclassified difficulties.

A number of the difficulties that we found in the current research were also found in previous studies. Firstly, Du Boulay [Bou86] noticed that novices tend to confuse the data type of a variable when a number is used as a string (D1). So and Kim [SK18] also found that students make the syntax error of forgetting to put quotation marks around string values (D2). The error of evaluating assignment statements in the wrong direction (D8) was also found by Du Boulay [Bou86] and Ma [MFRW07] [Sor12] (see section 2.1). Furthermore, regarding confusion about the purpose of a variable because of the name (D10), it has been found in previous research that bad variable

names can contribute to poor readability and making wrong assumptions about code [ADPAG13] (see section 2.2.1). The mistake of using a single equal sign trying to compare variables (D12) was previously found by Ma [MFRW07] (see section 2.1) and by Miller et al. [MSL15]. Miller et al. [MSL15] also found students to struggle with the difference between returning and printing a value (D13).

RQ2: How do these difficulties arise?
Teachers mentioned multiple possible causes for the difficulties that they observe. These include (i) distraction, (ii) not being familiar with Python-specific features, (iii) having a flawed mental model (caused by a lack of Python-experience or influenced by prior mathematical knowledge), (iv) variable naming and (v) a lack of programming experience in general.

Type errors (such as D1 and D2), for example caused by forgetting quotation marks around a string value, related mainly to students being inattentive or distracted, according to the participant. Distraction might also be the cause of students using the single equal sign instead of the double equal sign in a comparison (D12). In prior research, it has been suggested that students might be more prone to forgetting syntactical parts of a program, such as quotation marks (D2) or an extra equal sign (D12), when a task becomes more complex, because of an increased cognitive load [QL17].

Students being unfamiliar with Python-specific features also causes confusion and difficulties. D3 and D4 for example, were caused by students not being aware of some Python-specific features. Additionally, some of these features, for example being able to concatenate strings using the plus operator, are not very intuitive. As the students are beginners, they have not had much practice with Python to get used to these specific features, which makes them even more of an obstacle.

A deeper cause of difficulty, however, is when students have an inaccurate mental model of how Python handles variables. This became clear especially from the fact that students often misunderstand the effect of operations on mutable objects. It appears that students think of variables as referring to one memory location that can hold different values, instead of as a pointer that can refer to different objects (D5). It is not surprising that students have this inaccurate mental model, because teachers also mentioned they explained variables using the analogy of a box that contains a value, with the content of that box changing throughout program execution. This analogy is not accurate for how Python handles variables. While using this analogy may not do much harm for simple, immutable objects in the beginning, it can cause problems when students apply it to mutable objects later on in the course. Du Boulay noted this about analogies in general: "Very often an analogy introduced at one point does not fit later on, so producing extra confusion in addition to any misapplication of the analogy at the point where it was appropriate" [Bou86]. It is thus important that teachers consider the possible negative consequences of certain analogies, before using them.

Students were also found to have flawed mental models caused by influence of their prior mathematical knowledge. They were found to think of variables in programming as similar to variables in mathematics. This causes students to think of variable assignment statements as algebraic expressions and it causes them to fail to form the link between variables and the computer memory. The influence of mathematical knowledge can be related to difficulties with understanding the relation of variables to memory (D7, D8, D9) and also to the mistake of using the single equal

sign when trying to compare variables (D12). Ma [MFRW07] found that confusing the order of evaluating assignment statements (D8) is caused by the inaccurate mental model "assign from left to right" (see section 2.1). Teachers did not mention a possible cause for students using a single equal sign in a comparison statement (D12). Ma [MFRW07] referred to this difficulty (D12) as another inaccurate mental model. As mentioned in section 2.1, this flawed model could also be caused by prior mathematical knowledge. An example showing that students have trouble understanding that a variable refers to a memory location (D7), was that some students use the current value of a variable in their code, instead of its name. This could also be caused by their prior mathematical knowledge: students who make this mistake may think that variables are merely a placeholder for a certain constant value, which is how variables are treated in mathematics.

Another cause of difficulties can be the names that variables are given. Some names do not reflect the purpose or the content of the variable accurately, which can lead to confusion or misunderstandings (D10). Previous research has also shown that the use of poor code lexicon (e.g. variable names) can lead to a decreased understandability and can lead readers to make the wrong assumptions about the behaviour of the code [ADPAG13]. Arnaoudova et al. [ADPAG13] use the term Linguistic Antipatterns to refer to recurring poor practices in the lexicon of software systems. They give multiple examples of misleading names, such as a variable with a name that suggests it is a Boolean, but with a completely different type. This would be, for instance, a variable with the name "is_number" (suggesting it is a Boolean), but with a string data type.

Teachers also mentioned a lack of experience with coding in general or with coding in another language as a cause of students' difficulties. This cause was mentioned for type errors in general and also for the difficulty with variables of the same name that refer to different parts in memory (D4), string formatting with the percent sign (D6), not understanding that function arguments can have different names from their corresponding parameters (D11) and the difference between returning and printing a value (D13). A lot of these difficulties are related to the introduction new concepts that students who have never programmed before have to get used to. This can inevitably cause problems in the beginning, as programming novices face a high cognitive load [QL17]. Learning to program involves simultaneously solving problems that require abstract thinking and learning a new syntax. When new concepts are introduced, the required cognitive load can increase even more.

RQ3: What methods do programming teachers use to address these difficulties?
A lot of the teaching strategies that teachers mentioned for helping students overcome difficulties involved frequently repeating and stressing elements that often cause problems, by giving examples and doing exercises in class. Despite endless repetitions of pitfalls, however, students still often make the same mistakes. Therefore, teachers also found that giving students lots of practice time helps them to be able to solve and prevent their own mistakes overtime, as a kind of trial-and-error approach. They encourage students to apply print-debug regularly to make this easier.

Still, there are certain stubborn misconceptions that teachers need to address and preferably prevent, that do not simply disappear through trial-and-error. These are misconceptions that arise from flawed mental models and that need more attention. Teachers try to help students get an accurate mental model of what happens inside the computer by drawing memory diagrams, or by using tools that generate visualizations, such as Python Tutor. By often showing what happens in memory

during program execution, it is assumed that students will automatically picture these diagrams in their head or draw them on paper when trying to understand a program. This teaching strategy was also found in previous research [QL17]. As mentioned earlier, an important example of a flawed mental model is that students think a variable refers to one memory location that can hold different values, instead of as a pointer that can refer to different objects. Some teachers mentioned that they find it difficult how to explain the concept of variables to novices appropriately. They question whether they should explain it in a simplified way (e.g. by using the box-analogy) that is not entirely accurate, or to explain how Python actually deals with variables. The former is easy to understand and does not result in problems in the beginning, but may cause students to struggle with understanding mutable objects later on. The latter may be too complicated for beginners, but presents the concept accurately. It might be preferable to give the full, accurate explanation of how Python handles variables from the beginning. Even though it could make the concept more complicated than needed in the beginning, this approach may prevent difficulties later on, as was also found by Du Boulay [Bou86].

When it comes to variable naming, teachers say they encourage their students to make their variable names readable, intuitive and preferably longer than shorter. They say it is important to set the right example with this during the lectures and in solutions to exercises, so that students will take the habit of making their variable names meaningful. It was mentioned that variable naming is addressed in an implicit way, instead of explicitly telling students how to name their variables or to stick to naming conventions. Previous research [Tea94] also found that especially novices benefit from clear and meaningful variable naming when it comes to understanding code.

6 Limitations

This study analyzed the perceptions of seven programming teachers, which is quite a small number of participants. Furthermore, the teachers we interviewed all teach either in The Netherlands or in Belgium, and work mostly with university students. Consequently, the experiences of the participants of this study may not be generalizable to all programming teachers and students. However, the current research is a qualitative study, which means that the focus lies on gaining a deep understanding of the experiences of the participants, instead of generalizability.

Another limitation to this study is that programming teachers may not be aware of all difficulties and misconceptions that students have. Some teachers at university do not have much direct contact with their students, as there are often teaching assistants who interact with students during practical sessions. These teaching assistants might have a more accurate picture of what common difficulties are for students.

Furthermore, as the teachers had not seen the questions before the interview, they had to come up with answers on the spot. Therefore, some answers that teachers gave might have been less complete than would be the case if they had been able to prepare for the interview. Teachers also had to orally describe their teaching practices, such as how they draw memory diagrams. Because teachers could only give a verbal description, we were not able to reproduce any precise example of such memory diagrams.

Because the interviews were semi-structured, the participants had lots of room and freedom to talk about their experiences. During the interviews, teachers did not always answer every question precisely. This meant that some participants did not give an exact answer to every question, regarding the cause of a difficulty or their teaching strategies for a certain difficulty. Sometimes, teachers talked about causes of difficulties and their teaching practices more in general, instead of mentioning specific causes and strategies per difficulty. This is why not every teacher who mentioned a certain difficulty is discussed in terms of their suggested cause or their teaching strategies for this difficulty.

7 Conclusions and Further Research

In this study, student difficulties with variables in introductory Python courses were identified by means of data from interviews with programming teachers. Furthermore, using the interview data, we examined the causes of these difficulties and the teachers' strategies to address them. We grouped the difficulties that we found into five categories: type errors, Python-specific features, the relation of variables to memory, naming and a category for the remaining difficulties that we included. The causes of difficulties that were found are distraction, a lack of familiarity with Python-specific features, a flawed mental model, variable naming and a lack of programming experience in general. Often, combinations of these contribute to causing difficulties. Common teaching strategies were repeating and paying extra attention to pitfalls using examples and memory diagrams, allowing students lots of practice time with a trial-and-error strategy, and giving students the right example in choosing variable names. It was also found that teachers sometimes struggle when they have to make a trade-off between, on the one hand, explaining a concept elaborately and correctly and, on the other hand simplifying it by using an analogy, because the students are still novices. It can be hard to establish which of the two will help students in the best way. According to prior research, it may be preferable to explain a concept correctly directly from the beginning, possibly using an accurate analogy, instead of simplifying it with a less accurate or even problematic analogy.

The results of this research may be of interest to other, perhaps aspiring, programming teachers. By considering the difficulties, their causes and the teachers' strategies found in this study, they could take on new ideas on how to prepare their lectures to address and prevent difficulties for students in their own courses. Furthermore, if teachers are more aware of what elements can play a role in causing these difficulties, it could help them better understand and guide the learning process of their students.

Areas of further research might link the academic background of students to difficulties that they have. It would be interesting to know if students with different backgrounds struggle with different elements of variables or programming in general. These different backgrounds could be, for example, students in a computer science program versus students in a different program with only one programming course, or students in different levels of secondary school. If common difficulties of students from different backgrounds were established, this would make it possible to even better tailor the education that students with specific academic backgrounds receive.

References

[ADPAG13] Venera Arnaoudova, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. A new family of software anti-patterns: Linguistic anti-patterns. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 187–196. IEEE, 2013.

[Bae21] Baeldung. Statically typed vs dynamically typed languages. https://www.baeldung.com/cs/statically-vs-dynamically-typed-languages, 2021. Accessed: 2022-06-04.

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

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

[Del22a] Delve. The essential guide to coding qualitative data. https://delvetool.com/guide, 2022. Accessed: 2022-05-16.

[Del22b] Delve. The guide to qualitative methods of research. https://delvetool.com/qualmethods, 2022. Accessed: 2022-05-16.

[dJ10] Ton de Jong. Cognitive load theory, educational research, and instructional design: some food for thought. *Instructional Science*, 38, 2010.

[FJM+20] Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L. Pearce, and Andrew Petersen. Notional machines in computing education: The education of attention. ITiCSE-WGR '20, page 21–50, New York, NY, USA, 2020. Association for Computing Machinery.

[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*, UIST '15, page 609–617, New York, NY, USA, 2015. Association for Computing Machinery.

[KPEH10] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, page 107–111, New York, NY, USA, 2010. Association for Computing Machinery.

[Lex22] Lexico. misconception. https://www.lexico.com/definition/misconception, 2022. Accessed: 2022-04-12.

[MFRW07] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. Investigating the viability of mental models held by novice programmers. *SIGCSE Bull.*, 39(1):499–503, 2007.

[MHS14] Matthew Miles, Michael Huberman, and Johnny Saldaña. *Qualitative Data Analysis A Methods Sourcebook*. Sage, California, 2014.

[MSL15] Craig S. Miller, Amber Settle, and John Lalor. Learning object-oriented programming in python: Towards an inventory of difficulties and testing pitfalls. In *Proceedings of the 16th Annual Conference on Information Technology Education*, SIGITE '15, page 59–64, New York, NY, USA, 2015. Association for Computing Machinery.

[Onl15] Master Code Online. Python tutorial: How python variables reference objects. https://www.youtube.com/watch?v=z_55F4zSjoA, 2015. Accessed: 2022-06-04.

[Pea86] Roy D. Pea. Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2(1):25–36, 1986.

[Pla15] Danny Plass. Identifying and addressing common programming misconceptions with variables (part 1). 2015.

[Pyt21] PythonForEveryone. Is python a dynamically typed or strongly typed language? or both? https://www.youtube.com/watch?v=0nw8d0dUNhw, 2021. Accessed: 2022-06-04.

[QL17] Yizhou Qian and James Lehman. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1), 2017.

[SHS18] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. Programming misconceptions for school students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, page 151–159. Association for Computing Machinery, 2018.

[SK18] Mi Hyun So and Ja Mee Kim. An analysis of the difficulties of elementary school students in python programming learning. *International Journal on Advanced Science, Engineering and Information Technology*, 8(4-2):1507–1512, 2018.

[Sor08] Juha Sorva. The same but different students' understandings of primitive and object variables. In *Proceedings of the 8th International Conference on Computing Education Research*, page 5–15. Association for Computing Machinery, 2008.

[Sor12] Juha Sorva. *Visual Program Simulation in Introductory Programming Education*. PhD thesis, 05 2012.

[Sor13] Juha Sorva. Notional machines and introductory programming education. *ACM Transactions on Computing Education*, 13:8:1–8:31, 2013.

[SPBK86] D. Sleeman, Ralph T. Putnam, Juliet Baxter, and Laiani Kuspa. Pascal and high school students: A study of errors. *Journal of Educational Computing Research*, 2(1):5–23, 1986.

[Tea94] Barbee E. Teasley. The effects of naming style and expertise on program comprehension. *International Journal of Human-Computer Studies*, 40(5):757–770, 1994.

[Tec17] Technopedia. Strongly typed. https://www.techopedia.com/definition/24434/strongly-typed#:~:text=Strongly%20typed%20is%20a%20concept,and%20error%20(exception)%20occurs., 2017. Accessed: 2022-06-19.

[Tut22] Python Tutor. Python tutor: Visualize code in python, javascript, c, c++, and java. https://pythontutor.com/visualize.html#mode=edit, 2022. Accessed: 2022-06-19.

[Van16] Jake VanderPlas. Basic python semantics: Variables and objects (from a whirlwind tour of python). https://jakevdp.github.io/WhirlwindTourOfPython/03-semantics-variables.html, 2016. Accessed: 2022-06-04.

A Interview protocol

THE TEACHING OF VARIABLES IN PROGRAMMING COURSES
(perceptions, teaching strategies, difficulties)

WORKING TITLE: do variables get enough attention in programming courses?

RQ1: How are variables taught in novice Python programming courses?

RQ2: According to teachers, what is it that students find hard to understand about variables?

RQ3: What perceptions do teachers have about the relevance of variables in novice Python programming courses?

A. Introduction (5-10 minutes)

- Introduce myself (name, affiliation, research topic)
- Content + walkthrough (three parts: your own practice, student difficulties, general perceptions)
- Duration
- Recording and access
- Confidentiality + anonymity
- Is everything clear? Any questions before starting the interview?
- Confirm recording

You have already given your consent via the questionnaire. Now that we are about to start the interview, could you tell us if you still give consent to participate in this study? (recorded answer) Thank you!

You have also already given use some general background information prior to this interview. Perhaps it is nice if you can shortly introduce yourself here as well.

0. Can you tell me something about yourself? (who are you, what do you do, what is your background, what do you like?)

B. Practice (RQ1) (15-20 minutes)

Now I am going to ask you some questions about your own teaching practice. You are currently teaching an introductory course for python programming (or have recently done so). Is that correct? (yes/multiple) For the following questions I would like you to keep this course(s) in mind and draw examples from it. Do you have your course(s) in mind? (yes)

1. Can you shortly describe the setting of your course? (who are you teaching, how are you teaching, how long does the course take?)
    - *Level of education, Online/offline, Class size / nr. of students, In which language, Duration*

As has been said, we are interested in the teaching of variables in programming courses. When I say "variables", I am talking about the assignment of variables and their naming, and also their function within a code (also known as role of variables). Since we are discussing Python courses for beginners, variable names can also include function names but do not include class or method names.

2. Can you tell me something about how you explain variables in your course(s)?
    - *Can you give me an example?*
    - *What topics related to variables do you cover in your courses? (assignment, naming, role)*
    - *How much time/attention do you give variables in your courses?*
    - *When do you introduce variables in your course?*
    - *In your courses, are you promoting short & concise variable names (abbreviations, letters) or full words? Why? Can you give me an example of such a name?*
    - *Do you ask your students to use / read under_score or camelCase variable names? Why?*
    - *If not taught: Why not? Is it a conscious choice? Can you provide a reason?*

3. Can you tell me about a time that you use variables yourself while you explain other concepts throughout your course(s)?
    - *Short & concise (abbreviations, letters) or meaningful words?*
    - *under_score or camelCase?*
    - *Would you consider this example to be generic for the way you use variables in your teaching? (Why not? // Are there other ways that you use variables in your teaching yourself?)*

4. IF TIME: Are variables evaluated/assessed in your course? For example formally or informally.
    - *Which elements? (naming, assignment, role)*
    - *What do you use to assess? How do you assess?*
    - *Examples!*
    - *If not, why not?*

C. Student difficulties (RQ2) **(10-15 minutes)**

5. What are common errors that you see your students making when it comes to the concept of variables?
    - *Can you identify any underlying misconception that could explain that error(s)?*
        - *Or why do you think they make these mistakes?*
    - *What do you do to help them overcome the difficulties? (example, explain)*

6. Can you give me some examples of how your students struggle when it comes to **variable names**? What difficulties do they experience?
    - *At least two examples*
    - *Why do you think they occur?*
    - *What do you do to help them overcome the difficulties?*

7. Tell me about differences you experience in difficulties between reading/understanding code and writing code?
    - *Examples?*
    - *Why do you expect that to be?*

D. General perceptions (RQ3) **(~10 minutes)**

For the remaining of the interview I would like to ask you about your general ideas concerning variables.

8. In your opinion, what should variable names consist of? What information should it contain?
    - *What do you consider "good" variable naming?*
    - *What do you consider "bad" variable naming?*
    - *Is there a difference when it comes to "normal" variables vs. functions?*

9. As a programmer, and speaking in general, how important are variables to you while programming your own code and/or understanding someone else's code?
    - *Try to connect to what the participant mentioned before!!*
    - *Can you elaborate your answer?*

10. As a teacher, how important do you consider variables for teaching programming skills to students?
    - *Can you elaborate your answer? Which are the most important aspects according to you? Why? When do they become important?*

11. Finally, if you could make one recommendation about teaching variables and their naming to other teachers, what would it be?
    - *Please elaborate on your motivation for mentioning this recommendation.*
    - *Recommendations about what to stop doing?*

E. Closing: Other questions / remarks

This was the interview. Do you have any final remarks?

12. IF TIME: *How did you learn programming yourself?*

Thank you for participating!

[stop recording, save recording]