Universiteit Leiden
The Netherlands

Variables and variable naming in introductory programming education

Werf, V. van der

Citation
Werf, V. van der. (2025, September 2). *Variables and variable naming in introductory programming education*. Retrieved from https://hdl.handle.net/1887/4259393

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4259393

Note: To cite this publication please use the final published version (if applicable).

Chapter 3

Teachers' Beliefs and Practices on the Naming of Variables in Introductory Python Programming Courses

Vivian van der Werf
Alaaeddin Swidan
Felienne Hermans
Marcus Specht
Efthimia Aivaloglou

Abstract

Variable naming practices are part of the software developer's profession, influencing program comprehension and code quality. Yet, little is known about how variable naming practices are taught in beginner courses. This chapter investigates naming beliefs, self-reported teaching practices, and observations regarding variable naming practices of teachers of introductory Python programming courses. We adopted an in-depth qualitative approach by interviewing ten teachers from secondary education and higher education and developed several themes in order to answer our research questions. Among various opinions and practices, we found that teachers agree on using meaningful names, but have conflicting beliefs about what is meaningful. Moreover, the described teaching practices do not always match teacher's views on meaningful names, and teachers rarely encourage students to use them. Instead, they express that naming practices should not be enforced and that students will develop them by example. Whereas some teachers report focusing solely on conventions, others deliberately dedicate time for students to engage with naming, create their own guidelines, provide continuous feedback, or include naming exercises on exams. Naming practices do not seem to be deliberately taught, even though they influence program understanding and code quality. We also identified inconsistencies in teachers' self-reported naming practices. As such, we encourage intentional conversations about naming practices in educational settings, specifically linking naming to code quality and readability. We see room for group and peer activities as a means to this end, as well as providing formative feedback dedicated to naming. [1]

Keywords

variable naming
programming education
novices
teachers
qualitative interviews
open coding

[1]Published as: van der Werf, V., A. Swidan, F. Hermans, M. Specht, and E. Aivaloglou (2024). *Teachers' Beliefs and Practices on the Naming of Variables in Introductory Python Programming Courses*. In 2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering Education and Training (ICSESEET), **ICSE-SEET 2024**, page 368–379, New York, NY, USA. Association for Computing Machinery. doi: 10.1145/3639474.3640069

3.1 Introduction

Professional developers spend a significant percentage of their time (58%) on program comprehension-related tasks [Xia et al., 2018]. One root cause of this is that code is often written with 'meaningless' identifiers or variable names that are unintentionally misleading [Feitelson, 2023, Xia et al., 2018]. This causes problems in understanding and shows that finding a good name might not be straightforward. Accordingly, software engineering handbooks recommend professional developers to consider naming as a part of high-quality code, focusing on names' expressiveness, readability, and consistency [Stegeman et al., 2014, Börstler et al., 2017]. Evidently, naming plays a big role in understanding code [Avidan and Feitelson, 2017, Hofmeister et al., 2017, Lawrie et al., 2007b, Lawrie et al., 2006], which holds especially true for novice programmers [Teasley, 1994].

While some introductory programming courses include learning objectives that relate to code quality [Stegeman et al., 2014], several works already [Keuning et al., 2019, Börstler et al., 2017] noted that code quality, and naming in particular, do not seem to get equivalent attention in Computer Science Education research. Occasional efforts to incorporate naming include the development of code quality rubrics that involve naming as one explicit aspect to review or give feedback to students in their assignments [Stegeman et al., 2014, Stegeman et al., 2016, Glassman et al., 2015]. To the best of our knowledge, however, there is no research on teachers' perceptions of and approaches toward teaching (variable) naming in classrooms. We are interested in variable naming specifically, as variables are one of the first concepts taught in an introductory course, yet, the concept of variables is challenging for students to understand [Grover and Basu, 2017]. To this end, we conducted 10 in-depth interviews with teachers from secondary education, university, and adult education on the perceptions and practices of teaching the naming of variables. With these interviews, we aim to answer the following research questions:

RQ1 **What are teachers' beliefs and perceptions about variable naming practices?** We aim to identify how teachers think about names and naming practices in general, as we believe that these convictions are the background to which teachers adopt teaching strategies on the subject.

RQ2 **How are variable naming practices taught?** Here we investigate teachers' self-reported approaches regarding naming practices in the classroom. This question considers explicit (active) and implicit (or passive) teaching approaches, how teachers practice naming themselves, and information on feedback and grading.

RQ3 **What do teachers observe in the classroom concerning variable naming?** We are interested in what teachers observe in their students; for example, specific difficulties among their students and other observations.

3.2 Related work

3.2.1 What is good naming for comprehension?

Software engineering research indicates that programmers rely on names for their understanding of code [Avidan and Feitelson, 2017, Hofmeister et al., 2017, Teasley, 1994, Takang et al., 1996, Lawrie et al., 2007b, Lawrie et al., 2007a, Lawrie et al., 2006], and names often

serve as beacons during code comprehension [Gellenbeck and Cook, 1991]. Therefore, names can be of "poor" quality when they interfere with the reader's comprehension. In general, the following types of names are considered "poor" for code comprehension: names that are based on their data type (e.g. IntegerArray) or function within an algorithm (e.g. LoopCount) [Teasley, 1994], arbitrary names (e.g. GWhiz) [Teasley, 1994], and single-letter names [Lawrie et al., 2007b, Lawrie et al., 2006, Hofmeister et al., 2017]. Moreover, names can be unintentionally misleading and should therefore be chosen cautiously [Avidan and Feitelson, 2017, Arnaoudova et al., 2016, Fakhoury et al., 2020, Feitelson, 2023]. It was found that these misleading names resulted in more errors, taking more time, or giving up completely [Avidan and Feitelson, 2017]. Especially general, non-specific names such as "length" appeared problematic [Feitelson, 2023]. Research also identified multiple 'linguistic antipatterns' that misdirect a reader [Arnaoudova et al., 2016]. Common antipatterns that concern misleading names are names that "says one but contains many" or vice versa, and names that "suggests Boolean but type does not." Lexical inconsistencies like these significantly increase cognitive load [Fakhoury et al., 2020].

In contrast, there are also claims and observations on the effect of 'good' naming styles on code comprehension. Firstly, descriptive naming styles are advantageous over meaningless naming styles, like "Function1" or "FunctionA", even when documentation is provided [Blinman and Cockburn, 2005]. Additionally, meaningful abbreviations can be as effective as full-word names [Lawrie et al., 2006], and well-chosen abbreviations can in certain situations also be preferable over full words [Lawrie et al., 2007b]. When comparing letters, abbreviations, and full-word names, the latter still gives the best results on source code comprehension [Lawrie et al., 2006, Hofmeister et al., 2017], whereas letters *can* be meaningful if they convey information that is commonly attributed to that letter (i.e. "i" for index or "s" for string) [Beniamini et al., 2017]. However, attributions to specific letters vary over different programming languages [Beniamini et al., 2017], which of course has implications for learners of different languages.

Generally accepted recommendations on naming styles are that names must be picked with caution and given careful attention so that they reflect the concept or the role represented by each variable [Avidan and Feitelson, 2017], and, good naming consists of "limited, consistent, and regular vocabulary" with limited name lengths, so as not to overload a programmer's memory (the longer the name, the harder it is to retain the information) [Binkley et al., 2009]. Different roles of variables have been classified and investigated thoroughly in relation to comprehension by Sajaniemi and colleagues [Sajaniemi, 2002, Sajaniemi and Kuittinen, 2005, Sajaniemi and Prieto, 2005, Laakso et al., 2008].

On the subject of intermediate variables to break complex expressions into more manageable ones, Cates et al. [Cates et al., 2021] found that using an intermediate variable is generally beneficial for understanding only if the used name also reflects the meaning of the variables. Concerning camelCase and underscore styles, no difference in accuracy is found between the two styles [Sharif and Maletic, 2010]. Finally, naming styles are related to code quality [Stegeman et al., 2014]. For example, poor-quality identifiers (especially at the method/class level) are associated with lower quality, more bugs, and less readable source code, and natural language and recognized abbreviations can function as indicators for source code quality [Butler, 2009, Butler et al., 2010, Butler et al., 2009].

3.2.2 How do developers use names?

Naming of all identifiers, including variable names, accounts for over 70% of all characters in open-source projects and covers about a third of all tokens [Deissenboeck and Pizka, 2006]. Beniamini et al. [Beniamini et al., 2017] showed that single-letter variable names are common practice, quoting "in C, Java, and Perl they make up 9–20% of the names." Gresta et al. [Gresta et al., 2021] investigated Java naming practices in open-source projects and found that the three most common names are 'value', 'result', and 'name', while also single-letter names, like 'i', 'e', 's', and 'c', are commonly used. In twenty open-source systems that use the languages C, C++, C#, and/or Java, Newman et al. [Newman et al., 2020] looked for the most common grammar patterns in several types of identifier names and found that names typically have a singular noun-phrase grammar pattern (i.e. 'nextArea' or 'max_buffer_size'), with the exception of function names or when representing a Boolean value. More than three-quarters of identifiers containing a Boolean include a verb, likely to show that a question is answered by a true or false. Moreover, plural names often refer to a certain collection (of lists, arrays, etc.) or data grouping. Peruma et al. [Peruma et al., 2018] found that if identifiers are renamed, it is to narrow the meaning and serve code comprehension. Recently, Feitelson et al. [Feitelson et al., 2022] found that the probability that two developers choose the same name is very low, although different names are understood by the majority of developers. They suggest a model to help developers choose better names; in short, select the concepts that need to be included, choose words to represent those concepts, and construct the name with these words.

Swidan et al. [Swidan et al., 2017] analyzed projects in a block-based language originally directed at children (Scratch) to see how variable names are named there. They found that these names tend to be longer than in other languages, with most names between four and ten characters of length and only 4% of names being single letters. When single letters are used, 'i', 'x', and 'y' are the most common, showing both a crossover between languages and a focus on coordinates; the latter reflecting the focus on games and animations in Scratch projects. Additionally, Swidan et al. [Swidan et al., 2017] found that over half of the variable names are single words, with another 30% having a maximum of one space. This suggests that Scratch developers either use underscores or casing to separate words, like in most textual languages, or that single words are most naturally chosen by Scratch developers.

To support using consistent and concise names, a tool was developed striving to follow certain composition rules [Deissenboeck and Pizka, 2006]. This was followed up by Lawrie et al. [Lawrie et al., 2006], who then confirmed prior conclusions that programmers use a limited vocabulary [Antoniol et al., 2002, Caprile and Tonella, 1999]. Furthermore, Butler et al. [Butler et al., 2015] created a library checking naming conventions in Java, also in the context of using certain typography, abbreviations, and phrases. They found that about 85% of Java projects follow standard conventions. Allamanis et al. [Allamanis et al., 2014] presented a framework that learns the style of a codebase and suggests revisions to improve stylistic consistency. They noted that "almost one-quarter of the code reviews examined contained suggestions about naming," highlighting again the relevance of proper naming.

3.2.3 Naming in programming education

In comparison to experienced developers, proper naming styles (expressiveness, readability, and consistency) might be especially relevant for novices learning to program. For example, bugs are easier to find when words are used [Hofmeister et al., 2017], suggesting that good names improve code comprehension and debugging. Additionally, when developers use 'descriptive compound names' (i.e., "convertedInput" instead of "result"), they change their reading direction less often to find and correct a semantic bug, which they do 14% faster than when normal names were used [Schankin et al., 2018]. The effect, however, was stronger for experienced developers compared to novices, which suggests that novices do not benefit the same way, perhaps because they have not learned to 'interpret' specific naming customs yet. In fact, novice programmers often fail to name variables correctly [Gobil et al., 2009] and Scratch students are found to be misled by variables named with a letter, probably because of prior knowledge from their mathematics education [Grover and Basu, 2017]. These findings highlight that opening a discussion between teachers and students about "what is good naming" might be more relevant than just pointing toward naming conventions created by experts. This notion is strengthened by the work of Glassman et al. [Glassman et al., 2015], who, in the context of improving online curricula, developed a tool and a quiz for their MOOC to assess naming in terms of length and vagueness. As a by-product of their tool they found that feedback on naming practices, as well as both good *and* bad examples, was highly valued by students. Also Börstler et al. [Börstler et al., 2017] found that feedback related to code quality was frequently asked for by students. This suggests that topics such as readability, including naming, might not get enough dedicated attention in educational programs.

3.3 Methodology

This study aims to investigate teachers' beliefs, practices, and classroom observations on the naming of variables. Similarly to other works in CS education research [Keuning et al., 2019, Tshukudu et al., 2021], we captured such information by asking teachers directly through the means of semi-structured interviews.

3.3.1 Recruitment and teacher demographics

We targeted a wide range of teachers, including teachers at secondary school, university level, and adult education, who currently teach or recently taught one or more Python introductory courses. Teachers were recruited internationally through the networks of the authors and through the national network for secondary school informatics teachers. Teachers were required to speak either English or Dutch during the interview but could speak a different language in the classroom. Before the interview, teachers gave informed consent and completed a short questionnaire covering their backgrounds, such as programming experience, teaching experience, and other demographics. An overview of the recruited teachers can be found in **Table 3.1**. To minimize self-selection bias concerning naming specifically among volunteering teachers, they were approached with the topic of variables in general, not on the topic of variable naming.

In total, we conducted 12 interviews, with 7 teachers from 4 different universities, 4 teachers from 4 different secondary schools, and one teacher in professional "on-the-job"

Table 3.1: Overview of participants; formatted as: T1Um = [T]teacher [1], [U]niversity teacher, [m]ale. Teaching experience (programming and all) is counted in years. No. of students is per class.

ID	Education	m/f	Course target group	Age	Teaching exp. prog.	Teaching exp. all	No. of students
T1Um	university	m	CS + engineering BSc	25-34	2	2,5	70
T2Um	university	m	CS BSc	45-54	20	20	400
T3Am	adult	m	IT professionals	55-64	1	1	1-to-1
T4Sm	secondary	m	HAVO/VWO, optional	25-34	4	4	50-70
T5Um	university	m	CS + engineering BSc	55-64	16	17	400
T6Um	university	m	CS + engineering BSc	35-44	9	23	400
T7Sm	secondary	m	ICT track, mandatory	25-34	8	8	5-20
T8Sm	secondary	m	VMBO/HAVO, optional	25-34	2	5	25
T10Uf	university	f	CS BSc; CS MSc	25-34	1	1	300
T11Sf	secondary	f	HAVO/VWO, optional	<25	1	3	20-30

coaching. Participants worked in The Netherlands, Belgium and Spain. Their mother tongues were Catalan, Dutch (incl. Flemish), French, Italian, and Turkish. The participating university teachers taught in English or Dutch, whereas the secondary education teachers all taught in Dutch. Our participants' teaching experience ranged from 1-20 years, indicating we recruited both experienced as well as starting teachers. Most teachers program themselves, with the exception of one secondary school teacher. Eight teachers also taught other languages besides Python.

3.3.2 Interview process

We used a semi-structured interview protocol consisting of questions on three topics about the teaching of variables: (1) general perceptions, (2) teaching practice, and (3) student difficulties (see Appendix 3.6). To capture a broad view of teachers' perceptions, practice, and experiences, each topic contained various neutrally posed open-ended questions that offered room to dive into detail with follow-up questions. The interview covered both the concept of variables in general and the naming of variables specifically. Variable naming was covered in all three topics both as part of specific predefined questions and as follow-up questions during the interview. Each participant was given an equal opportunity to talk about naming. When naming did not come up naturally, the topic was introduced via follow-up questions. However, not every participant spent equal time on the topic: in cases where the interviewee was not able to elaborate any further, the interviewer moved on to other questions. A pilot interview was used to test and inform the interview protocol, after which it was decided that no further alterations or refinements were needed.

All interviews were conducted online, by the first author, via MS Teams, and recorded for transcription. The average length of the interviews was 62 minutes, about half of that time was dedicated to the topic of naming. The interviews were transcribed manually, in the original language of the interview (Dutch or English), using intelligent verbatim transcription. Transcripts were checked for discrepancies and made anonymous for subsequent processing. From the 12 interviews we conducted, two were excluded from the final analysis: one teacher did not teach Python programming despite indicating this

beforehand, and one interview was considered a duplicate as it was with an assistant who taught alongside a previously interviewed teacher and revealed no new information.

3.3.3 Coding process

To obtain a broad overview of variable naming practices in classrooms, we used a qualitative open-coding approach from a constructivist perspective [Corbin and Strauss, 2008, Kenny and Fourie, 2015, Charmaz, 2014].[2] This means that we prioritized information generated by the data in an intuitive way, rather than creating a framework or hypotheses based on literature which then is used for deductive coding. Accordingly, the complete transcripts were analyzed using an iterative process ("open coding" and "refocused coding"), which means that *all* quotes throughout the interview related to variable naming were coded in a way that best summarizes the quote's intent and meaning [Corbin and Strauss, 2008, Kenny and Fourie, 2015, Charmaz, 2014]. This process is known to generate a large set of individual (in-vivo) codes that can be grouped and merged into themes according to the research interests. In this case, the first author coded three interviews first and used the open codes from these interviews to develop a more structured codebook. The initial version of the codebook consisted of grouped themes that provided information about our research questions, such as "naming beliefs" (RQ1), "teaching strategies" (RQ2), "grading and feedback" (RQ2), "own representations" (RQ2), and "student observations" (RQ3). In iterative rounds, the first author, together with the last author, also identified preliminary main categories that gave direction into the specific topics that teachers brought up. These categories distinguished, for example, between various perspectives (i.e., focusing on meaningful names or letters), teaching strategies (i.e., active or passive), and teachers' own identification of their representation of naming (i.e., using letters or full words).

Using the developed categories as a guideline, the coder then (re)coded all 10 interviews, still maintaining a semi-open coding approach up until saturation was reached. This means we continued creating new open codes if needed, but mostly added codes and quotes to existing themes and categories. This process was done iteratively and repeated for already coded interviews when new insights were made. New insights also meant that the codebook was updated: new themes and categories were added, renamed, split, or merged until all relevant and remaining open codes were summarized and grouped into categories with similar meanings and intentions. For example, the old theme "teaching strategies" was split and renamed into "active" and "passive" teaching approaches, each with its own categories to more accurately describe and interpret the information given by the teachers. During this process, doubts were discussed with the last author during regular meetings, in which the last author also checked the emerging categories and themes for clarity and validity. The final codebook is presented in **Table 3.2**. In total, we ended up with 238 individual codes to analyze. The tools used during the data processing and analysis were Atlas.ti and MS Excel.

[2]Although this research follows Grounded Theory (GT) procedures, the intent of this work is to gather various existing perspectives and teaching practices among a variety of programming teachers. Since we know of no prior work attempting to create such an overview, we considered an iterative process as used in GT procedures most intuitive to discover patterns in teachers' own descriptions.

Table 3.2: Final code book with examples of codes per developed theme, category, and RQ. Examples of specific quotes (utterances) can be found in-text in the results section.

RQ	Theme	Category (# of codes)	Examples of individual codes
1	beliefs and perspectives	meaningful names (47)	describes what is in the variable; provides its function; should be intuitive
		using letters (16)	i, j, k, n, l are not very informative
		size and detail (14)	prefers longer names; depends on situation
		overall structure (30)	use a certain structure; use underscores
2	active teaching approach	coding conventions, rules, guidelines (7)	uses or focuses on a personal coding guideline; focus on community practice
		readability, programmers attitude (7)	focus on readability; focus on job expectations; focus on human aspect
		other active approaches (6)	focus on errors (pointing out, discussion); stresses code works independent of naming; provides explicit naming assignments
2	passive teaching approach	mentioning no teaching (4)	not explicitly taught; no discussion on naming
		students learn by practice (12)	naming comes naturally; lead the way; only during other assignments
		other (5)	no specific way is required, taught on demand
2	own representation	"meaningful" or similar term (8)	representative; descriptive
		single letters or abbreviations (5)	uses single letters: loops; uses single letters: basic operations
		depends (2)	depends on the program; depends on the purpose
		other (5)	practical reasons, no particular style
2	grading and feedback	no evaluation (4)	not graded; no points deducted
		unclear (5)	part of general assignments; part of other skills
		evaluated (6)	graded on test; continuous feedback
3	student observations	difficulties (23)	typos; too long names; what is 'i' in a loop
		causes of difficulties (10)	students lack creativity; confusion because of renaming
		other observations (22)	better students give better names

3.4 Results

3.4.1 RQ1: What beliefs do teachers have about variable names?

Below we present different topics that teachers mentioned when they reflected on naming practices. The results are summarized by statements reflecting teachers' beliefs about variable naming.

Names should be 'meaningful' and 'intention-revealing'

Most teachers agree that naming is important and should be meaningful. Names are considered meaningful when they are simple, straightforward, and intuitive. They have to be descriptive, clearly represent the contents of the variable, or show its purpose. Mentioned examples are usually nouns: *studentName*, *interest*, *length*, *result* or *index*. To sum up what is regarded as 'meaningful', T1Um tells his students: *"try to name it a name that makes sense to you and two other people."* He also notes that the addition of adjectives, for example, *current_maximum*, can be extremely helpful in loops, but should be used *only* when it makes sense, to avoid new confusion. For example, if there is only one maximum in the code, adding a *current* to *maximum* is not helpful. Moreover, names are to be intention-

revealing. Teachers emphasize this specifically when it concerns functions: names have to reflect the function's purpose so that *"just by looking at the name of the function you can tell, okay this function is supposed to perform this, and so on" (T11Uf)*. Mentioned examples are structured with a verb to indicate it "does something": *calculate_weight*, *organize_file* and *find_cost*.

Teachers disagree on using letters as names

Letters and abbreviations are generally considered to provide too little information to be descriptive. Especially in the context of teaching, T4Sm explains: *"if I start using very bad names like 'a', 'b', and 'c', then, the code still works the same way, but it's not telling students what it does. And it can be a good exercise, but not in the parts where I'm explaining what they do. It's a good exercise on a test where [the students] need to know better but not during teaching or not during comprehending the concept that I'm trying to explain."*

However, there is disagreement on whether letters should be used. In particular, T11Sf mentions that *"letters in the case of operations are meaningful because [my students] can easily relate it to their math classes from before, which makes it an appropriate naming scheme."* Also, T8Sm remarks, *"with small assignments I will use letters, especially with basic math operations, using 'a + b' is just more logical than writing 'number1'. It's more like mathematics" (translated)*. Another consideration to use letters is the traditional practice in the (online) community. This is especially true for (nested) loops, where the use of *i - j - k* is common practice: *"if [students] would google to something, they would find it like that. So I try to teach them also in how they would find it if they would google online" (T11Sf)*. However, some rather use an *x - y - z* structure in nested loops: *"Now, for me [i-j-k] is an example that it doesn't make much sense because if I'm going through a matrix in which there is an 'x' and there is a 'y', why am I using 'i' and 'j'? I know, it's tradition to use i-j-k etc., I just think that in some cases it would make more sense to use 'x' and 'y'. (...) Imagine that I want to use 'x in y', I have to put 'i in j', and then, depending on how long is the loop, I have to remember by heart that 'i' is 'x' and 'j' is 'y' " (T1Um)*.

Whether letters are considered meaningful thus seems to depend on whether the letter itself carries meaning. In other words, using random letters from the alphabet is generally viewed as 'bad practice' whereas particular letters are accepted in certain contexts, like loops, short codes, or codes that are not intended for sharing: *"If it is for myself and nobody else is ever going to see it, I might even use 'x', 'y', 'a', but as soon as it's something that I will share... yeah, no, for sure. I put the variables with the right names. I have to consider the fact that somebody else is going to read this" (T1Um)*.

Names have an ideal size and level of detail depending on context

Several teachers report that names should not be "too short," or "too long." As is the case with random letters, it is reasoned that names that are too short create confusion because they do not convey enough information to the reader, which in turn makes it hard to remember what contents are behind which names. Too long names, on the other hand, create confusion because the reader might not read the whole name and rather assumes its contents or function after reading only a part of the name (T4Sm). Teachers furthermore mention that "enough detail" should be provided, but not "too much." For example, the name *student* is not detailed enough when its contents are numbers: it remains unclear

if the variable represents a student's age or grade or something else. On the other hand, variables named *Max* with contents "Max" or *seven* with an integer 7 are "too detailed," as well as *all_names_of_name_list_starting_with_a*.

According to teachers, writing efficiency also affects the balance between longer and shorter names: longer names are less efficient and more prone to typing errors (T8Sm, T4Sm, T10Uf). However, T11Sf mentions, *"especially with beginners, I would prefer the longer names, where we give the purpose of the variable or what it does over short and concise names, even though I get that it's more time-consuming."* Nevertheless, the ideal size of a name varies per teacher and context. Short names, and even single letters, are considered "okay" in short programs, whereas in longer programs names should also be longer (T1Um). The simplicity and conciseness of a single word are valued, but only if the name is unambiguous in its meaning. A maximum of one to three words are preferred, connected with an underscore or via casing.

Overall structure is important but naming is a personal style

The overall naming structure and the relationship between names are considered important. Some (T1Um, T6Um, T10Uf) prefer a numbered structure, for example, *plant1, plant2, p1, p2, a1, a2, example1, example2, str1, str2, df1, df2*, or a logical structure between the names. However, T3AM cautions that such structures can get too complex and confusing, for example, when names are structured like *aa, ab, ba, bb*, etc. Moreover, T1Um and T4Sm stress that names should not be too similar to each other to avoid confusion between names (*apple* vs. *apples*). Additionally, T4Sm also warns that *"if all or most variables look the same, students think it should be done that way."* He experienced this with a structure consisting of *myInput, myText, myInt*, as used in a KhanAcademy module: students copy it, and start creating names such as *myLastTwoValues*. This *"does not help and is not mandatory (...) It is not bad, but it is not what I expect from [my students] when using variables (...) [and they] have to be able to make more complicated names if necessary."*

Naming conventions are also mentioned as important. While T11Sf prefers following traditional Python or community guidelines (i.e. PEP), others adopt self-created guidelines in their teaching (T4Sm, T7Sm). For T2Um and T6UM, using a certain naming style is not very important, as long as their students are consistent. Furthermore, depending on the teachers' own programming background they prefer underscores over camel-case or vice versa, for example, T1Um: *"I do like underscore because it gives me a visual interruption."*

Finally, some teachers consider names that include data types to be helpful to novices or prevent issues when (accidentally) combining data types. For example T10Uf, *"I try to associate the variable with its type. If it's a list then the name has a list, if it's a string then the name (...) most likely is going to have a string in its desirable name"* and T3Am, *"to keep a certain type-safety or reminder by including it in the name (...), especially for beginners, I recommend using naming that is as clear as possible, and possibly even include data types" (translated)*. However, since Python is a dynamically typed language, T7Sm notes: *"it is not that important for students that don't use that kind of programming languages to really be constantly reminded of the datatype"* and also T6Um mentions: *"in my opinion, it's not necessary. (...) I don't have a tendency to say that the type should be reflected in the variable name."*

3.4.2 RQ2: How are naming practices taught?

We found various teaching practices that we grouped into *active teaching approaches* - naming is explicitly taught or mentioned in the classroom, *passive teaching approaches* - naming is not or implicitly taught, *own representation* - the way teachers use naming themselves when teaching, and *feedback and grading* - whether or not naming is evaluated.

Active teaching approaches

We consider teaching approaches as active when naming is explicitly taught as part of the curriculum. There are two major topics: (1) coding conventions, including guidelines, community practice, and specific naming rules, and (2) readability, including clear and meaningful naming and the human aspect, such as teamwork and job expectations.

Coding conventions, guidelines and rules. Most teachers mention coding conventions; however, T11Sf remarks: *"I try to use the conventions of the languages, but that is quite difficult when the students learn multiple different languages during the three short years that they have computer science."* Consistent with this statement, conventions, guidelines and rules are not very homogeneously taught among the teachers, which complies with the diverse beliefs we have identified among teachers concerning naming practices. Some teachers set up their own naming guidelines or recommend students to make their own structure, others mention to include tradition and community practice (i.e. PEP) in their teaching and focus on naming conflicts or recommending their students to include data types in their names. T5Um mentions consistency to be most important in teaching naming: *"what I would stress is more that things are done in a consistent way rather than having, doing it always one way or another; the point is you shouldn't mix and match in the same program different styles, whether it's for naming variables or for even programming style or indentation and the comment style, all of that."* To help students develop their naming practices, our teachers regularly mention to provide tips and show examples. Additionally, tools such as Visual Studio Code or PyCharm are sometimes adopted for correcting and teaching coding guidelines.

Readability, meaningful naming, and the human aspect. Most teachers merely mention to students to use clear and meaningful names, for example: *"We do insist on trying to give names which are as readable and as complete as possible" (T2Um)*. However, some teachers (T4Sm, T6Um, T7Sm, T8Sm) (also) discuss why naming is important. This usually includes a human aspect such as organizing your code to remember or find stuff back. Other human aspects are working in teams, future job expectations, and naming something in a way that you and at least two other people can understand what you mean. When names are not "readable," or, "according to the set guidelines", T7Sm goes as far as telling his students *"Okay this thing, I don't know what you mean here so I can't read your code, right now"*, even if he does understand the names. Additionally, he likes to prepare the most frequently seen mistakes in student projects, in order to discuss and evaluate them in class and to show how students can improve their own projects. With these strategies, he wishes to provide continuous feedback, prioritize the importance of naming, and motivate his students to first fix naming issues before they can get help from him on other aspects of the code.

More active approaches. Teachers also indicate using strategies such as pointing out naming errors (T1Um, T7Sm, T11Sf), discussing mistakes in class (T7Sm), and providing explicit naming assignments (T4Sm, T7Sm) that include bad smells and error-hunts. T1Um explicitly stresses that code works independently of naming and that naming therefore is only important for a human reader: *"I put a lot of stress on the fact that [naming] does not matter but that it matters on our level of organization. (...) I don't oblige them to rename [their variables] because (...) I want to stress that the code could work anyway. (...) You could call it 'banana' and it works, you just have to know where to put 'banana'. On the other hand, I also tell them it has to make sense for somebody who reads it."*

Self-reflections. Several teachers reflected upon their own practice and mentioned wanting to incorporate more specifics about the practice of variable naming. T8Sm: *"This is something that I now will start to think about much more than I ever have before, that is also nice for me" (translated)*, and T4Sm: *"I underestimated how I teach variable names because I thought it was one lesson and involved less and I can teach them everything about variables [in one lesson]. But I've already split that into two, three lessons, just for Python. Not only because it's not as uncomplicated as I thought, but also because it's a lot bigger than I thought (...) It has to be done because it's not as natural as I think it worked."*

Passive teaching approaches

We consider passive teaching approaches all strategies that do not *explicitly* teach naming practices. This includes all statements where teachers mention that they do *not* give specific attention to naming practices, and all statements pertaining to practices where students are (sometimes explicitly) assumed to learn by themselves. This thus involves indirectly taught naming practices (i.e. "through general exercises" or by "leading the way"). Furthermore, some teachers do not require students to use specific naming styles. **Table 3.3** presents an overview.

Table 3.3: Overview of passive teaching approaches used

Passive teaching approach (N)	Teachers
No explicit focus on naming (6)	T2Um, T5Um, T6Um, T8Sm, T10Uf, T11Sf
Naming is practiced through examples (8)	T1Um, T2Um, T3Am, T4Sm, T5Um, T8Sm, T10Uf, T11Sf
Specific naming is not enforced (6)	T1Um, T2Um, T5Um, T8Sm, T10Uf, T11Uf

No explicit focus on naming. Teachers report having no specific focus on naming in their courses. For example, T2Um reports: *"we don't have an explicit theory session where we talk about the naming conventions for variables would be this or that"*, and T11Sf mentions: *"It is not something that I start focusing on but it is something that [students] do start noticing along the way."* Interestingly, this finding is in contrast to what we see under *active approaches*. Specifically, teachers tell us not to have a specific focus on naming practices, while they *also* indicate telling students to "choose meaningful names" or to

"follow the conventions." However, teachers with this inconsistency do not seem to follow up their instructions with further explanations or assignments; instead, they remain with general tips. When elaborating on why they do not explicitly teach naming practices (see below), teachers assume that students will pick up "good naming practices" on their own and that naming practices do not require more explicit teaching or attention. It is also mentioned that naming practices should not be enforced as it is seen as an individual choice or preference.

Naming is practiced through examples. Teachers assume that students learn naming by themselves, either by following the traditional or given conventions or through practicing in other assignments. For example, *"so not very explicitly, but often naming is featured in the context of an assignment [that shows] that it eases understanding" (T8Sm)*, and *"we introduce the rules as we go, by the examples we give them" (T2Um)*. Even more strongly, T1Um chooses to lead the way as he assumes his students will copy him: *"I do it passively. For example, saying, "for 'index' " because they're indexes, "for 'length' " because it's a list of lengths. So I try to make them get there."* Furthermore, T5Um argues that naming practices do not need explicit teaching. He states *"I don't insist on [naming] very much (...) I mean, that comes more naturally by example"*, and emphasizes that it is not *"a major source of concern for the students"* as they are confronted with a lot of code through exercises, examples, and their own written code. Following this, he mentions: *"I don't think naming is a big concern to us [teachers]."* Two more teachers mentioned focusing only on naming *if* and *when* that was necessary, for example, in the context of an error or when the topic was brought up by a student.

Specific naming is not enforced. Teachers do not like to emphasize -or require- specific ways of how variables should be named, but rather leave it up to the students. T5Um: *"we don't specifically insist very much on how variables should be named"*, and T1Um: *"I don't want to stress a lot they have to use these names or use that name."* Instead, T2Um tells his students: *"It's okay, your code will work and it is not so important in this course, we are happier if your code works."*

Own representation

We consider how teachers use names themselves when they are using examples or show live coding to their students as their *own representation* of naming practices. It can be seen as setting an example to the students, as such, we also consider teachers' own representation a passive teaching approach. However, since it is always present in a course, and therefore complementary to other approaches used, it deserves its own category.

Almost all teachers report that they use, or try to use as much as possible, meaningful names or equivalent terms. For example, T1Um: *"I'm very straightforward, so like for (...) doing the for-loop, I do: for index in the list of indexes. Because they're indexes, so, let's use index."* Interestingly, many equivalent terms are mentioned (see **Table 3.4**), possibly showing that no one single way of good naming is present, and perhaps also showing slight nuances in what teachers find most important in choosing a name. The variety in the self-reported own representations presented here is consistent with the variety of naming beliefs that we discussed previously.

Table 3.4: Descriptions used by teachers to show what type of naming they use themselves when teaching. Interpreted as variations of "meaningful names."

Description used	Teachers
meaningful names	T4Sm, T5Um, T11Sf
clear names	T3Am, T4Sm, T10Uf
representative names	T3Am, T4Sm, T8Sm
descriptive names	T4Sm, T8Sm
straightforward names	T1Um, T3Am
informative names	T6Um
adequate names	T5Um
useful names	T5Um
one-letter names	T5Um, T8Sm, T10Uf, T11Sf
abbreviations	T8Sm, T6Um
no one-letter names	T3Am, T6Um

Some teachers note that their naming depends on the program or the purpose of the code. While T3Am and T6Um explicitly told us that they do not use one-letter names, others told us that they do use abbreviations and one-letter names, sometimes depending on the purpose of the program or the complexity of the code. Single letters were especially used when teaching loops and basic (math) operations, for reasons already discussed in Section 3.4.1 (*Teachers disagree on using letters as names)*. In particular, these reasons concern a connection to prior knowledge (mathematics) and tradition or community practice. Moreover, short names, abbreviations, and single letters were also used for practical reasons or convenience. T8Sm: *"If it doesn't matter much, or the code is small, I usually use just a letter, to have overview [and] for time efficiency. If the code grows larger or more complex I prefer abbreviations" (translated)*. T6Um: *"I would prefer to avoid these too short names, although, actually, on some of my slides, I do use these short names."* His reasoning is to avoid using a font size that is too small while still being able to compare two pieces of code on the same slide. During the interview, he realized that *"at the same time, if you do that too often you give a bad example, that's... that's a difficulty [laughs]" (T6Um)*. Interestingly, as his first response to the question of how he used names himself in teaching, he said: *"I like consistency a lot (...) that you approach things in a consistent manner, that students get a certain, learn a certain way of thinking"* (T6Um).

Feedback and grading

The topic of feedback and grading came up in 7/10 interviews, from which we identify three approaches to evaluating naming practices: (1) no evaluation, (2) indirectly evaluated or plays a minor role in grading, and (3) explicit grading and/or feedback. First, there is a strong tendency to not grade or evaluate students on their naming practices. Most teachers explain that naming is not part of the evaluation of student's work, or that students do not get "punished" (i.e. subtraction of points) when improper names are included in their submissions, for example, T10Uf: *"We don't do a lot with variable naming (...) I don't think we pay attention to readability."* One reason mentioned is that auto-graders do not look at naming quality. However, teachers indicate that it does not make sense to grade it separately since naming is interwoven with performance on other concepts (T7Sm,

T10Uf). Even though consistency within a code is often desired, this is not enforced (T5Um, T8Sm).

Second, when naming is part of grading it is usually connected to "programming basics," "using conventions" and "good commenting," or it is graded through practice with weekly assignments. However, although these weekly assignments are not necessarily focused specifically on naming practices, they cover various programming topics, including variables, and they are explicitly mentioned by the teachers as opportunities to practice naming. Therefore, how naming is actually evaluated remains unclear.

Third, two teachers show explicit evaluation of naming practices. T7Sm mentions that although naming plays only a minor part in the grading of his students' work, he finds it important to always provide continuous feedback on naming conventions and good naming practices. This includes the active teaching approach of not evaluating a student's work if the names are "unreadable," or in other words, not to the standards that were taught in class. Only T4Sm specifies that naming practices are explicitly considered in the grade: *"During the projects, I assess how readable the code is. It's part of the readability, it depends on how they describe their variables. If they're all like x, a, z, and b, then I don't have a clue what's happening, so they'll get point reduction because it's not readable code. But, if they use the naming conventions that I've taught them and describe what's happening in the code then it's a lot more clear to read, so they'll get points for that."* He even implements specific naming assignments on the final test: *"there's a specific assignment in the test that's about what's happening in this program, and [I ask them to] rename the variables to make more sense [and] to be more descriptive" (T4Sm).*

Overarching patterns

We also investigated overarching patterns by grouping individual teaching approaches. The results are shown in **Table 3.5**. In short, we identified three different teaching profiles: (1) teachers that primarily use active approaches, (2) teachers that use a mix of approaches, and (3) teachers that hardly or do not at all incorporate naming practices in their courses. While teachers with an "active" profile show deliberate design choices for including naming practices, and those who do not teach naming practices either deliberately "opt out" of it, or were previously unaware that naming *could be* part of their course, most teachers show a "mixed" profile. This could indicate that teachers act intuitively based on their own experiences and beliefs regarding naming practices.

Our analysis further points towards a distinction between secondary and tertiary education: only secondary education teachers show an active approach to teaching naming practices, whereas university students are mostly expected to rely on their own abilities to learn and use appropriate naming practices. However, the small amount of teachers in our sample is not suited to draw any such conclusions definitively, and the distinction made here is purely based on the profiles emerging from the teaching approaches. As such, there is no clear indication (yet) that the teaching of naming practices requires different approaches across educational levels.

Table 3.5: Results of our cross-analysis into overarching patterns: three teaching profiles related to the teaching of the naming of variables are identified based on several common characteristics of our research questions.

Themes		**profile 1 - active**	**profile 2 - mixed**	**profile 3 - not taught**
	Dominant philosophy	naming is an essential skill	mixed, naming is learned by example	naming should not be taught
	Part of course	dedicated time allocated	no dedicated time, but woven into the course or by leading the way	no attention given, students rely on themselves
Active approach	*focus on conventions*	own guideline created	mixed	traditional conventions
	focus on readability	yes	mixed	no
	explicit naming assignments	yes	rarely	no
	attention to naming errors	yes	mixed	no
	naming tips and examples	on meaning and pro-actively given	mixed	on conventions and upon request
Passive approach	*required to follow guidelines*	yes	no	no
Own representation	*uses single letters or abbrev.*	no	mixed	mixed
	consistent with teaching	yes	mixed	mixed
Grading and feedback	*includes naming*	yes	no	no
	Teachers	T7Sm, T4Sm	T1Um, T2Um, T6Um, T8Um, T11Sf	T3Am, T5Um, T10Uf

3.4.3 RQ3: What do teachers observe in the classroom concerning variable naming?

Reported student observations

Teachers didn't observe many issues with naming practices among students. They tell us that naming practices are not a major source of concern and students 'get the hang of it' very quickly. For example, T2Um: *"We do insist on trying to give names that are as readable and as complete as possible, and I tend to believe [students] do that quite quickly. Apart from the first few sessions, where obviously they will use variables like 'x', 'y', 'z', and use names that don't say anything. We do insist on that during the practical sessions and in all the examples we give them. I think they very quickly catch on doing that."* Additionally, T1Um tells us once he points out a mistake, students often recognize their mistake immediately. Furthermore, students tend to use a mix of single letters, abbreviations, and single words in their first programs. However, throughout the course, and once students start to recognize and experience the importance of naming, they pick up the habit of using meaningful names (T2Um, T8Sm, T11Sf), and even start asking what convention they are expected to use at that point: *"After about three months of programming, and we start touching upon new items, students will start asking me themselves 'okay but what naming convention should we use for this thing'" (T7Sm).* Also, T11Sf says, *"Currently, the students usually go back to 'a', 'b', 'x', 'y', and sometimes something more useful. And when they start PHP, of course not Python but PHP for their website, they start to notice 'o wait, the naming is kind of important'."* Furthermore, teachers observe that more experienced students choose more appropriate lengths for names (T11Sf), and, students that use better naming also present better programs in general (T7Sm). T5Um and T8Sm also note that their students seem to copy the examples that they are shown for their own naming practices. These observations are interesting in relation to our previous finding regarding the teaching strategy "lead the way": although teachers may not be directly aware of it, they seem to say that such a passive approach to teaching naming practices is valid, sufficient, and effective to teach naming practices.

Specific difficulties and reported causes

Mentioned difficulties were considered of minor importance by the teachers and we did not find any patterns among them. Firstly, teachers observe that most mistakes concerning naming appear because students are inconsistent, make typos, or lack creativity. T2Um notes that this might originate from an inconsistency between teachers, examples, and learning materials, which might further confuse students. Although the presence of typos could be just an oversight on the students' part, teachers mention this proneness to typos as a reason to not use too long names: mistakes are often and easily made in spelling, wrongly placed capitals, or using invalid names (T4Sm, T11Sf). T11Sf furthermore mentions that sometimes a student might lack creativity, possibly caused by a lazy attitude and a desire to make the assignments with the least amount of effort. Thinking of a good name might be considered "too much" effort, especially because students may not have been taught about "what is a good name". Teachers also observe confusion caused by names that are too similar, especially with longer names (T4Sm). Secondly, teachers note difficulties in connection to other identifiers such as functions and parameters (T2Um, T5Um, T10Uf, T11Sf). These difficulties include name conflicts or using the same names

for different objects, causing unintentional overwriting. Teachers attribute this issue to not being sufficiently introduced to the new concepts, meaning that with more practice and exposure, this mistake will disappear. Thirdly, teachers observe that students may (gradually) change names during debugging and writing (T1Um, T10Uf), leading to confusion when students have forgotten that names have changed, or when the name was only changed in one part of the code, but not yet in another. Finally, T6Um and T10Uf observe that there exists confusion among students about the "i" in a for-loop: students do not seem to understand that this is also a name.

3.5 Discussion

3.5.1 Impact on teachers and educators

The results from the interviews suggest possible impact for teachers and educators in three directions:

Teaching approaches for naming

While many teachers indicated that they mention naming conventions and guidelines, only teachers with an "active" approach indicated they use explicit pedagogic approaches that focus on the naming of variables in their classes. These teachers focused on an instructional approach with direct assignments or tips. At the same time, the interviews suggest that teachers realize that choosing a proper variable name is context-dependent and dependent on who will read the code. As a result, we see room for adopting a wide range of sociocultural teaching approaches that focus on group and peer activities. Especially considering teachers' philosophy in which naming is learned by example, some activities can include the use of live coding sessions, peer instruction-assessment-review, and pair programming, all with a focus on the naming of variables.

Developing CS Teachers' PCK on code quality

Teachers need to develop further their Professional Content Knowledge (PCK) on code quality in general and on the effect of naming on code comprehension. This is especially important for introductory courses that include learning objectives related to code quality. However, from the interviews, it seems that there is a matter of priority, as some teachers indicate that there are more important concepts to focus on than naming, especially when it comes to grading or feedback. To tackle this, the link between naming and code quality needs to be stressed. Having readable and expressive variable names is not a matter of code aesthetic, but rather an important aspect of code quality that is known to affect code comprehension [Lawrie et al., 2006, Schankin et al., 2018, Avidan and Feitelson, 2017]. The effect of bad naming will extend to the professional life of the student as a developer and will have an impact on their ability to contribute to projects and on the performance of daily programming tasks [Xia et al., 2018].

Using existing tools

Teachers are also capable of giving constructive feedback on naming. We believe that while teachers are obtaining more PCK on code quality, they could already implement such feedback. Prior work has found feedback on naming both desirable and valuable [Börstler

et al., 2017, Glassman et al., 2015]. Practical examples of existing teaching resources are rubrics and tools that are recently developed for such goals [Stegeman et al., 2014, Stegeman et al., 2016, Glassman et al., 2015]. These tools can be a good starting point to evaluate where students stand in their variable naming so that teachers can give constructive feedback to help improve the level of readability, expressiveness, and consistency of the naming of variables in their code assignments.

3.5.2 Reflection on teaching themes

When looking at the emerging themes on teaching variable naming, generated from the interviews, we can see that these themes follow the two mainstream theoretical pedagogic approaches in computer science education [Fincher and Robins, 2019]. On one side, the 'active' teaching theme follows an "instructivist" approach: the focus is on the structure and presentation of learning materials more than on the learners who are seen as recipients. Yet, this is not a pure picture: within the profiles of teachers who presented quotes fitting to the active teaching theme, we also observe aspects of "constructivist" approaches. In particular, some teachers refer to programming languages' guidelines on naming as a way to 'support that construction of knowledge' rather than to communicate knowledge. This, in effect, delegates learning goals to the students who will discover the topic of naming on their own and decide which names to use, without the teachers integrating their students' activities into the classroom. These and similar constructivist approaches of teaching variable naming are even more visible within the passive teaching theme, again with less focus on students' activities in the classroom. From the interviews, we observe that such teaching approaches have roots in the teachers' beliefs and perceptions that naming does not need explicit attention because it "comes naturally by example".

3.5.3 Limitations and future work

As our research is based on self-reported data, teachers may have given us socially accepted answers. We have tried to limit this by making the topic of the interviews more general about variables, formulating questions neutrally, and taking into account the order of the questions to avoid leading. Especially regarding teaching practices and student difficulties, our findings are self-reported: we did not observe classroom practices ourselves. However, to look at actual practices, we have conducted research into Massive Open Online Courses (**Chapter 4**) and Programming Textbooks (**Chapter 5**) and found similar results [van der Werf et al., 2023, van der Werf et al., 2024b].

Although there was no indication before this work that naming is or should be addressed differently across educational levels, our findings suggest this might be the case. However, as often with qualitative research, our sample set is too small to make representative conclusions, and being representative was not our current aim. Nevertheless, our study could be followed up with a large-scale (international) questionnaire to generalize and compare target audiences, class sizes, and class duration. Such research might also provide further insights into the different teaching profiles that we have found and could further dive into comparing naming practices among different programming languages.

The effect of the programming language

Some findings are specific to Python. For example, one difficulty that teachers described was that of students using the same name for variables and functions, causing unintentional re-assigning, which would not be a difficulty in statically typed programming languages. Furthermore, naming conventions and guidelines, which were often mentioned during the interviews, are to a large extent language-specific. Finally, characteristics of other programming languages not native to Python, such as pointers and static types, will not be reflected in our findings, even though they might have effects on teachers' perceptions and practices on variable naming.

Future work

Our future line of research is to analyze programming textbooks, to further understand how practices are represented in different forms of education. Additional research is planned covering in-class observations, which can be compared with what teachers say about the topic. We also aim to design and experiment with specific naming tasks to investigate how naming can be easily but effectively implemented in existing curricula.

3.6 Conclusion

This chapter aimed to investigate the current teaching practices and beliefs concerning variable naming. Primarily we want to encourage discussion on teaching naming practices in programming education. Hence we investigated teachers' beliefs and perceptions about variable naming (RQ1), their practice (RQ2), and their observations in the classroom concerning naming (RQ3).

Our results show a diversity of opinions; however, in line with most existing literature on 'good naming' for comprehension, our teachers all advocate for simple, straightforward, and intuitive names that clearly represent the content or show the purpose of the variable. Nevertheless, when it comes to the actual teaching practice, this promotion of meaningful names is not so directly demonstrated. Even though teachers tell their students to use meaningful names, they seem to rarely incorporate practices that encourage or force students to think critically about what a good name entails, or how names might be misleading. Moreover, teachers themselves do not always use meaningful names in their examples to students, even though they agree that students learn naming practices by example.

Interview protocol

Questions that explicitly cover the topic of NAMING of variables: 2b, 2c, 3, 4b, 6, 7, 10. Other questions might include naming if the interviewees brought up the topic themself.

Introduction (5 minutes)

- Overview, duration, recording, confidentiality, anonymity
- Introduction interviewer + interviewee

Practice (15-20 minutes)

(1) Can you shortly describe the setting of your course?

  a) Follow up on the level of education, online/offline, class size, language, duration

(2) Can you tell me something about how you explain variables in your course(s)?

  a) Can you give me an example?
  b) Follow up on topics related to variables (assignment, **naming**, role), dedicated time/attention, when introduced, etc.
  c) In your courses, what type of **names** are you promoting? Why? Motivate.
    i) Follow up on short & concise (abbreviations, letters) vs. full words
    ii) Follow up on examples of promoted names
    iii) Follow up on underscore vs. camel case
  d) If not taught: Why not? Can you provide a reason? (Is it a conscious choice?)

(3) Can you provide me an example of how you **name** variables yourself while you explain other concepts throughout your course(s)?

  a) 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?)
  b) Follow-up on name length, letters, words, conciseness
  c) Follow-up on underscore vs. camel case

(4) (if time) Are variables evaluated in your course? Why? How?

  a) Can you give me an example?
  b) Follow up: formally/informally, which elements (inc. **naming**?), why (not)

Student difficulties (15-20 minutes)

(5) What are common errors that you see your students making when it comes to the concept of variables?
  a) Follow up on misconceptions identified, causes, how to overcome

(6) Can you give me some examples of how your students struggle when it comes to variable **names**? What difficulties do they experience?
  a) Follow up on why they might occur and how teachers solve them.

General perceptions (10 minutes)

(7) In your opinion, what should variable **names** consist of? What information should it contain?
   a) What do you consider "good naming"? What do you consider "bad naming"?
(8) 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?
(9) As a teacher, how important do you consider variables for teaching programming skills to students?
(10) If you could make one recommendation to other teachers about teaching variables and their **naming**, what would it be? Motivate.
(11) Recommendations about what to stop doing?

Closing

(12) Do you have final remarks or questions?