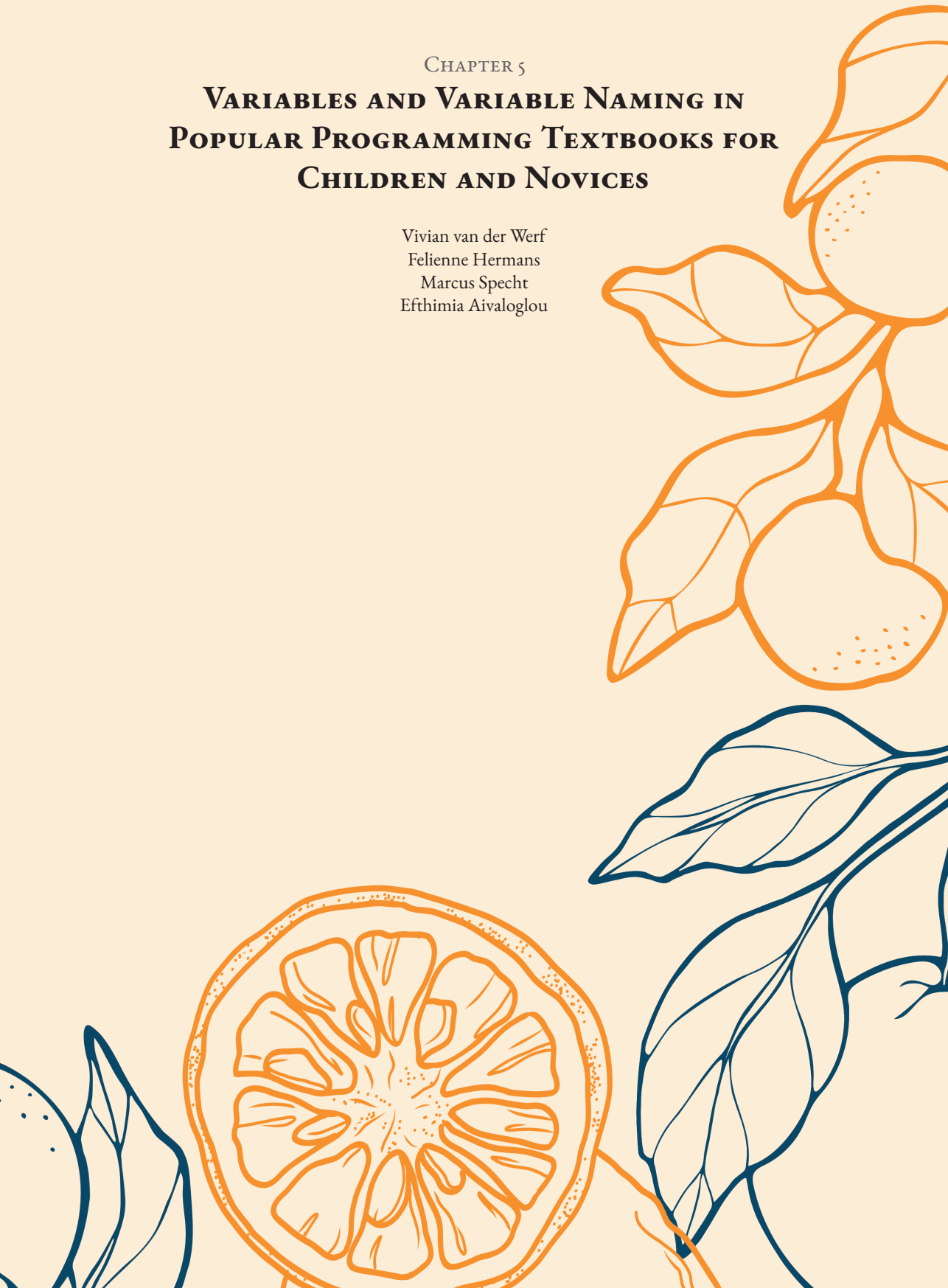
Variables and variable naming in introductory programming education

Werf, V. van der

Chapter 5

Variables and Variable Naming in Popular Programming Textbooks for Children and Novices

Vivian van der Werf
Felienne Hermans
Marcus Specht
Efthimia Aivaloglou



Abstract

In programming, the concept of variables is central to learning other concepts like loops, functions, and conditions, and the way variables are explained influences students' understanding. ***Chapter 4*** *observed Massive Open Online Courses (MOOCs) on introductory programming to investigate how the topic is addressed in teaching materials. This chapter aims to verify if these results generalize to other materials by analyzing 13 popular Scratch and Python programming books and investigating (1) which definitions and analogies are currently being used to explain the variables, (2) looking into the programming concepts that are introduced alongside variables, and (3) analyzing if and how variable naming practices are introduced. Our results support previous findings from MOOCs, suggesting that CS educators and developers of educational materials for introductory programming could pay more attention to how they explain variables and can be more deliberate and consistent when it concerns the teaching of naming practices. Additionally, we found specific analogies used to explain variables, and differences between programming languages in the order that variables are introduced. Our work can be used to update current educational materials and inform the development of new ones.* [1]

Keywords

programming education
variables
naming practices
analogies
programming concepts
qualitative content analysis
Python
Scratch

[1] Published as: van der Werf, V., F. Hermans, M. Specht, and E. Aivaloglou (2024). *Variables and Variable Naming in Popular Programming Textbooks for Children and Novices*. In Proceedings of the 2024 ACM Virtual Global Computing Education Conference V. 1, **SIGCSE Virtual 2024**, page 242–248, New York, NY, USA. Association for Computing Machinery. doi: 10.1145/3649165.3690112

5.1 Introduction

While variables are important for core programming skills such as reading and understanding code [Pelchen and Lister, 2019, Lister et al., 2009, Sajaniemi, 2002], they are also a hard concept to grasp for novice programmers [Hermans et al., 2018b, Kohn, 2017]. However, since several programming concepts expand on the concept of variables (i.e. control flow, functions), it is essential that variables are well understood. At the same time, variable naming practices are also relevant to the act of reading and understanding code: meaningful identifier names help readers understand code more easily than when abbreviations or (random) letters are used [Avidan and Feitelson, 2017, Lawrie et al., 2006]. Yet, other work also found that full names can be misleading if they do not correctly represent their contents or purpose [Arnaoudova et al., 2016, Caprile and Tonella, 2000].

Prior work [van der Werf et al., 2023] **(Chapter 4)** already investigated teaching practices regarding the concept of variables and variable naming by observing introductory programming MOOCs. To verify whether their findings generalize to other materials, the current chapter investigates the same topics in programming textbooks. Additionally, since previous work on textbooks [McMaster et al., 2016, McMaster et al., 2018] investigated *which* concepts are covered, but do not detail *how* these are covered, this chapter also aims to expand on the current state of knowledge on teaching practices. Following [van der Werf et al., 2023] **(Chapter 4)**, but in the context of introductory programming books, our research questions are:

RQ1 How are variables explained? (use of definitions and analogies)

RQ2 What other programming concepts are introduced either together with, right before or right after variables?

RQ3 How is naming addressed when variables are introduced?

5.2 Related work

5.2.1 Analogies for explaining variables

Analogies are often used to explain programming concepts [Fincher et al., 2020]. In education, an analogy, metaphor or notional machine is a 'tool' that supports learning by simplifying a concept through a representation that highlights the most important aspects of the concept, while obscuring less important aspects [Fincher et al., 2020]. For example, 'variables as parking spaces' transfers knowledge about parking spaces to the comprehension of a variable. Waguespack [Waguespack, 1989] explains a variable of a particular data type as a 'container with the corresponding shape' (*shape* refers to the data type). With metaphors like 'container' or the popular 'variables as a box', it is important however to stress that the container or box can hold only a single value. It has been found that, even though this analogy can support an initial understanding of the concept, it is also susceptible to the common misconception among novices that variables can hold multiple values at the same time [Hermans et al., 2018b, Boulay, 1986, Chiodini et al., 2021]. Any analogy might thus only partly or incorrectly represent a concept and can, therefore, leave novice students with an incorrect understanding. Nevertheless, Doukakis et al. [Doukakis et al., 2007] found that using an analogy appears preferable over using

none. More research is needed to understand which analogies are suited in which contexts, and if we should abandon the box metaphor entirely, perhaps replacing it with 'variables as labels' [Hermans et al., 2018b]. In introductory programming MOOCs, prior work [van der Werf et al., 2023] **(Chapter 4)** found common use of the metaphor variables as a (mail)box in explanations and visualizations and also a 'question-and-answer' format to think about variable names and contents. Another promising way of introducing and teaching variables is with the help of Sajaniemi's theory of "roles of variables" [Sajaniemi, 2002, Sajaniemi and Kuittinen, 2005], which categorizes variables based on their dynamic nature, i.e., *fixed value, stepper, gatherer*, or *most-wanted value*.

5.2.2 Variable naming

That (variable) naming is important for comprehension and code quality is indisputable from the existing literature focusing on the effect of naming on program comprehension, code quality, and coding skills. Most importantly, 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., 2006], and names often serve as beacons during code comprehension [Gellenbeck and Cook, 1991]. Moreover, bugs are easier to find when words are used [Hofmeister et al., 2017]. Additionally, names that are not descriptive enough, for example, single letters or abbreviations from which meaning is not directly clear, interfere with code comprehension [Lawrie et al., 2007b, Lawrie et al., 2006, Hofmeister et al., 2017, Beniamini et al., 2017]. The same holds true for names that are too long, making them difficult to remember [Binkley et al., 2009]. Additionally, names can be unintentionally misleading and should therefore be chosen cautiously [Avidan and Feitelson, 2017, Arnaoudova et al., 2016, Feitelson, 2023, Feitelson et al., 2022]. Especially general, non-specific names, such as 'length' [Feitelson, 2023] or 'result' [Schankin et al., 2018], appear problematic. Finally, novices can wrongly believe that computers interpret or assign values based on the semantic meaning of variables' names, and thus incorrectly apply semantic assumptions to syntax [Kaczmarczyk et al., 2010].

Consequently, it is relevant to think about how we teach variable naming in introductory programming courses. Thirty years ago, Keller [Keller, 1990] indicated that variable naming was rarely included in programming textbooks. Since then, little research observed teaching practices on this topic. Two recent studies [van der Werf et al., 2023, van der Werf et al., 2024c] **(Chapter 3, 4)** found that teachers do address naming practices in their learning materials, but inconsistently: variable naming practices are not always taught explicitly, taught practices are sometimes conflicting, and given example code does not always match the provided rules and recommendations. Moreover, research investigating code quality perceptions among students and teachers [Börstler et al., 2017] confirmed students' desire for 'more and more specific feedback about what was good and bad in their code'. Other studies on variable naming in education found that novice programmers often fail to name variables correctly [Gobil et al., 2009] and that Scratch students are misled by variables named with a letter, probably because of prior knowledge from their mathematics education [Grover and Basu, 2017].

5.3 Methods

To answer our research questions, we analyzed thirteen textbooks that aim to teach Scratch *or* Python to children and novices. To systematically select programming books we used two Amazon best sellers lists (top 100 popular products based on sales), both visited on April 18, 2023. For Scratch books, we selected the five books ranked highest within the *Amazon Best Sellers: Best Children's Programming Books*. Also for Python books, we selected the five books ranked highest within the same list. However, since teens and young adults might prefer using adult textbooks, we also added the three books ranked highest within the *Amazon Best Sellers: Best Python Programming*. For all lists the following selection criteria were applied: 1) being a physical book, 2) written in English, and 3) focused *solely* on learning Scratch or *solely* on learning Python. The selected books and their details are presented in **Table 5.1**.

Table 5.1: Overview of selected programming books

ID	Target	Bestseller	Title	Year
S1	Children	#6	Coding Games in Scratch [Woodcock, 2015]	2019
S2	Children	#13	Coding Projects in Scratch [Woodcock, 2016]	2019
S3	Children	#14	Code Your Own Games! [Wainewright, 2020]	2020
S4	Children	#22	Coding for Kids Scratch [Highland, 2019]	2019
S5	Children	#60	Learn to Program with Scratch [Marji, 2014]	2014
P1	Children	#4	Coding for Kids python [Tacke, 2019]	2019
P2	Children	#7	Python Coding for Kids Ages 10+ [Makda and Mamazai, 2022]	2022
P3	Children	#8	Coding Games in Python [Vorderman et al., 2018]	2018
P4	Children	#12	Python for Kids [Briggs, 2023]	2023
P5	Children	#19	Coding Projects in Python [Vorderman et al., 2017]	2017
P6	Adults	#2	Python Crash Course [Matthes, 2023]	2023
P7	Adults	#4	Python Programming for Beginners [Robbins, 2023]	2023
P8	Adults	#6	Automate the boring stuff with Python [Sweigart, 2020]	2019

To systematically collect our data and ensure good operational definitions, the first author created a a codebook in a Microsoft Form, which was tested on three random books (one from each category) by the first author and an independent data collector. Issues were resolved and a new version of the form was designed by the first author. This version was then independently used by both parties to gather all information relevant to the research questions. The data collector was recruited from a pool of research assistants and hired to reduce bias in the collection of data. As such, after transferring the data to MS Excel, the first author compared the two sets. Any information found by only one collector was reassessed for inclusion.

Each research question covered different chapters and was analyzed separately, as specified below:

RQ1: Explanation of variables We collected all definitions (quotes) and analogies (quotes and pictures) from the section in the book that introduces the concept of variables. We then also checked all other sections, and, when applicable, glossaries, for any definitions of variables. For example, sometimes a summary with a definition was also given at the end of a chapter, which was included.

The collected definitions were analyzed on the object (what are variables: nouns, i.e., a 'box', a 'memory location'), the purpose (what do variables do: verbs + addition, i.e., 'store information'), any additional information that was provided (i.e., 'data can change'), and if any, used analogies. For each definition, the relevant information was recorded. Additionally, when images were provided to accompany the definition, they were described and it was recorded which analogy it represents. The independent data collector and first author had no disagreements.

RQ2: Other programming concepts To investigate how variables are connected with other programming concepts, we examined the concepts discussed right before and right after the concept of variables. To this aim, we investigated three chapters: the chapter in which variables are introduced, the chapter before, and the chapter after. This means that if a topic is not represented in our results, it was either not covered in the book, or it was introduced in other chapters and therefore not considered in the analysis.

To collect the different concepts, we first made a list of expected programming concepts (based on [van der Werf et al., 2023] **(Chapter 4)**) to check for in the chapters and added to this list when we encountered a different concept. We then systematically analyzed the different chapters for the presence of these concepts. For the analysis, we categorized the concepts into the following topics: data types, operators, control flow, print-input statements, and others.

RQ3: Naming rules and guidelines To search for naming rules and guidelines, we looked at the chapter where variables were introduced. Any rules discussed here were collected, following the categories found in [van der Werf et al., 2023] **(Chapter 4)**: (1) *syntax rules*, including case sensitivity, accepted symbols, reserved keywords, and restriction of spaces; (2) references to specific naming *conventions*, such as camel case or underscore styles, and (3) any guidelines on *variable name meaning*.

For the first two categories we collected which rules and conventions were mentioned much like a closed coding process. For the third category, we used an iterative and open coding process which meant we analyzed the books several times. Based on an initial glance at the chapters, we first collected whether one of the following topics was addressed: 'use descriptive/meaningful names', 'avoid single-letter names', 'avoid misleading names', and 'you should be able to understand your name'. During this phase, we also gathered other quotes or statements on naming we encountered, if any, such as 'use a simple naming method,' 'too long names are hard to read,' 'consistency in naming is important,' and 'you can use any name you want.' Then, after going through each book, we went through all the books again to see if any newly encountered statements were missed in earlier books. To continue the analysis, we grouped all naming statements and quotes into four subtopics: those (a) suggesting to use meaningful/descriptive names, (b) addressing reasons for using such names, (c) addressing the length of the name, and (d) highlighting that names can be whatever you like. We furthermore noticed several other interesting quotes that were collected under 'other'. After the grouping, to ensure a complete overview, all books were checked a final time for any additional input on any of these four topics.

Besides this, we collected and investigated explicit examples and naming exercises, when provided. We then checked other parts of the books to see if naming was (also)

addressed elsewhere, for example, some books include a section or chapter on "how to improve your code". If naming was addressed elsewhere in the book, we recorded the context.

5.4 Results

5.4.1 How are variables explained?

Most Scratch books explain **variables as a box**, as opposed to only 3/8 Python books (see **Table 5.2**). This analogy is often accompanied by a picture that affirms it. A typical definition looks like 'a variable works like a box that you can store information in, such as a number that can change' (S1). Some books explain **variables as a place** or (memory) location, for example, a variable 'describes a place to store information, such as numbers, text, lists, and so on' (P4), or, 'a variable is a named area of computer memory' (S5). Few books (also) explicitly address **variables as a label**, for example, a variable is 'a fancy name or a tag' (P1) or 'essentially a label for something' (P4). Others include it more implicitly, mentioning that the variable is a 'labeled box' or needs a name 'to label the information.' To address the common misconception that often happens with the variable as a box analogy, P6 writes: 'Variables are often described as boxes you can store values in. This idea can be helpful the first few times you use a variable, but it isn't an accurate way to describe how variables are represented internally in Python. It's much better to think of variables as labels that you can assign to values. You can also say that a variable references a certain value.'

Table 5.2: Explaining variables with analogies and purpose.

Variables...	**Scratch**	**Python**
...as a box **with image*	S1, S2*, S3*, S5*	P3*, P5*, P8*
...as a place	S4, S5	P2, P4, P8
...as a label *(implicit)*	*(S2, S5)*	P1, P4, P6 *(P3, P5, P8)*
To store information	S1–S5	P3–P8
To track information	S2	P1, P3, P5
To access information	S5	P3, P4, P5
To interact w/ information		P7
To support code writing		P4
To use later		P3, P4, P5
Their value can change	S1, S2, S4, S5	

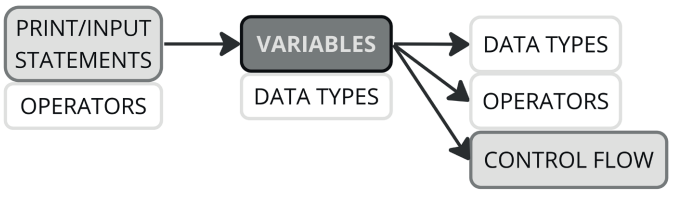
Most explanations address the purpose of variables. Books most often write that variables 'store information'. Other purposes mentioned are to 'keep track of information', 'to access information', 'to interact with information', 'to support code writing', 'and to use later'. In addition, only Scratch books mention explicitly that a variable's information (value) can change, for example, 'notice that the value of the score changes throughout the program. This is why we call it a variable – its value changes' (S5).

Scratch books primarily explain variables as a box; Python books use more diverse explanations. The emphasis is on 'storing information', while other purposes of variables get less attention. Only Scratch books explicitly mention that a variable's value can change.

5.4.2 What programming concepts are introduced alongside the concepts of variables?

Scratch and Python books apply different trajectories when it comes to which programming concepts are introduced alongside variables (see **Figure 5.1**). Below we discuss detailed results per concept.

(a) Scratch



(b) Python

Figure 5.1: Trajectory of programming concepts as introduced by (almost) all [in gray] or about half [in white] of the books.

Simple data types (string, integers, float, boolean) are discussed by all Python books, either in the same chapter as variables (P1, P3, P5, P6, P8) or in the next (P2, P4, P7). Only one Scratch book (S5) introduces them (right before variables). More **complex data types** or structures (arrays, lists, dictionaries, tuples) are covered in 4/8 Python books, either in the same chapter (P3, P5), and/or the next (P3, P6, P7). They are not addressed in Scratch books.

All Python books and three Scratch books address **mathematical operators** (+, -, /, *) in the predefined chapters. While most books prefer to introduce them *after* variables (S2, S3, S5, P1, P2, P6), Python books also introduce them right *before* (P4, P7, P8) or *together with* (P3, P5) variables. We see a different pattern for **comparison operators** (==, !=. <, >) and **logical operators** (and, or, not): they are introduced by almost all Scratch books and only half of the Python books, most frequently *after* variables are introduced, either in the same chapter (S2, P3, P5) or the next (S3, S5, P1, P8). Only S4 introduces comparison operators right before variables.

While all Scratch books introduce control flow concepts like **if-else statements** (5/5) and **loops** (4/5) in the predefined chapters, only three Python books do so. Moreover,

Scratch books (except for S5), introduce them *before* variables, while all three Python books (P3, P5, P8) introduce them *after*. We noticed that in Python books, control flow is often introduced later in the books.

Several other concepts are (sometimes) introduced in the predefined chapters. Here we mention only those that are covered in two or more books. For Scratch, these concepts are user input, random numbers, first program, error messages/bugs, and procedures. For Python books, these are print-statements/first program, using comments, error messages/bugs; user input, and functions/classes.

*Scratch books introduce mathematical operators after variables; Python books **also** introduce them right before or together with variables. Control flow concepts are introduced before variables in Scratch books, and after variables in Python books, if at all in the chapters surrounding variables.*

5.4.3 How is naming addressed?

Variable naming is addressed in almost all books, except for S3 (see **Table 5.3**). Nine books provide a dedicated section on naming, whereas three books only briefly mention naming. Three books provide dedicated naming exercises. Especially Python books also address naming in chapters, in the context of functions, scope, name errors, conventions, or readability. When naming is reintroduced, books mostly repeat what is mentioned in the chapter on variables, or explicitly refer back to it.

Table 5.3: Overview of how naming is addressed.

	Scratch	Python
Is naming addressed in the chapter that introduces variables? *(N)*	**4**	**8**
yes, naming is briefly mentioned	S2, S4	P4
yes, naming has a dedicated section	S1, S5	P1–P3, P5–P8
Are naming exercises provided? (yes)	–	P1, P2, P8
Are there explicit examples of "good" or "bad" naming? *(N)*	**4**	**8**
yes, but only on syntax rules ('valid' or 'invalid' names)	S4, S5	P7, P8
yes, the good examples also address the descriptiveness of names	S1, S2	P1–P6
yes, the bad examples also address the descriptiveness of names	–	P1–P6
Is naming addressed in other parts of the book? *(N)*	**1**	**6**
yes, when functions are introduced	–	P1, P3, P5, P7
yes, in a section on readability / conventions	S1	P2, P8
yes, in another section	S1	P2, P3, P5, P8

Syntax rules

Syntax rules, like reserved keywords and case-sensitivity (see **Table 5.4**), are addressed by all Python books (8/8) and just 2/5 Scratch books (S1, S5). Both Scratch books mention that spaces are technically allowed, but it is better to avoid them because other languages do not allow it, for example, *DogSpeed* instead of *dog speed* (S1) and *SideLength* instead of *side length* (S5). Besides mentioning the rules, several specific example names are also given in 5/8 Python books, for example, *Good1* (P2) is accepted, whereas *2Good* (P2), *100_days_of_code* (P1), and *TOTAL_$UM* (P8) are names disrespecting the rules.

Table 5.4: Overview of syntax rules addressed in the books.

Syntax rules	Scratch	Python
Names are case-sensitive	S5	P3, P5, P8
Use Unicode-letters, no symbols		P2–P8
Do not start with a number	S5	all
Do not use spaces	S1, S5	all
Do not use reserved keywords		P2, P4–P7

Naming conventions

The same two Scratch books (S1, S5) mention naming conventions, such as using underscores to separate words. All Python books refer to such conventions or 'community guidelines' (P7), although the specific conventions mentioned vary and can deviate from the common underscores, for example, camel case (P1, P2, P8), Pascal case (P1), Hungarian notation (P2), PEP (P8), and *Zen of Python* (P6, P7, P8). P2 and P8 also use camel casing in their example code. Furthermore, P2, P3, and P6 mention that constants are fully capitalized: 'constants will be named in all caps to spot them easily' (P2), for example, *PI* or *SPEED_OF_LIGHT* (P2).

Variable name meaning

Three out of five Scratch books note that it is preferable to use meaningful (S1, S5), sensible (S2), or descriptive (S5) names that 'tell you what the variable is for' (S1) and 'to make the code readable' (S1). Given examples are *speed, score, dragon* (S1), *High Score, Player Name* (S2), *firstName*, and *interestRate* (S5). On the other hand, 7/8 Python books instruct students to use descriptive (P1, P3, P5, P6, P8), meaningful (P2, P3, P4), or useful (P4) names. For example: 'When naming a variable you want to be as descriptive as possible but also follow the rules of Python (P1),' and, 'the variable name should be meaningful e.g. if a variable stores the name of my friend, then the variable name should be *friendName* not just *name* which can be confusing or misleading' (P2). Avoiding confusion is not the only reason given for using descriptive names. The idea of variables (and names) *storing* something *inside* them (see also Section 5.4.1) is again highlighted. For example, P5 writes to 'think of a name that will remind you what's inside the variable', others note that a good name 'describes the data it contains' (P3, P8). The most common argument, however, is to improve *readability*, explicitly mentioned by S1, P1, P7, and P8. Interestingly, the latter addresses its own examples as too generic: 'most of this book's examples use generic variable names like *spam*, *eggs*, and *bacon*, but in your programs, a descriptive name will help make your code more readable' (P8). Finally, two books note that good names will help you to *understand* (P1, P5) the code.

In 5/13 books, the length of variable names is also related to a name's meaning (S5, P3, P4, P5, P6). For example, S5 writes to avoid using single-letter names such as *w* or *z*, 'unless their meaning is very clear'. The book also continues with that 'names that are too long can make your script harder to read.' P6 stresses that names 'should be short but descriptive [therefore] *name* is better than *n*, *student_name* is better than *s_n*, and *name_length* is better than *length_of_persons_name*.' P3 and P5 are less explicit but give examples such as using *attempts* rather than *a* (P2) and *lives_remaining* rather than *lr* (P5). On the other

hand, P4 writes: 'Sometimes, if you're doing something quick, a short variable name is best. The name you choose should depend on how meaningful you need the variable name to be,' however, no further explanation is provided besides '*Fred* probably isn't a very useful name.'

Finally, some books (S4, S5, P4, P8) make an explicit mention that variables can be named anything. Whereas most do so while highlighting that descriptive and meaningful names are highly recommended, S4 only writes: 'you can name a variable anything you want–get creative (...) *points*, *goals*, or yes, even *hippo farts*.'

Regarding naming practices, most books focus on syntax rules that, when not adhered to, break the program. Python books give more attention to naming guidelines and variable name meaning, yet, like Scratch books, also present conflicting information, take a 'free-for-all' approach, or remain vague on what is a 'meaningful' name.

5.4.4 Patterns between children and adult books

The analysis highlighted two differences between the Python books for children and adults. First, in the books for children, the topic of functions was sometimes introduced within our predefined chapters, however in adult books this topic had a chapter elsewhere. Second, regarding naming, we found that all children's books and P6 provided explicit examples of "good" or "bad" names that cover what is and is not a descriptive name. The other two adult books focused on examples regarding syntax rules.

5.5 Discussion

We investigated how the concept of variables, and the respective naming practices, are taught in thirteen popular introductory Scratch and Python programming textbooks. Our collected data was qualitative in nature and included definitions and analogies used to explain variables, other programming concepts introduced with or near variables, and any naming practices that are addressed. Our most important findings are:

5.5.1 Variables are commonly explained as a box

From the literature, we know that analogies come with a risk of carrying over misinformation from one topic to the other [Boulay, 1986, Chiodini et al., 2021, Hermans et al., 2018b]. Consistent with prior work [van der Werf et al., 2023] **(Chapter 4)**, we found a tendency to explain variables as a box, which is prone to cause misconceptions when learning new programming concepts. Nevertheless, one book explicitly addresses this issue, while others opt for alternative explanations, such as variables as a label or place. This might indicate that the community is looking for new analogies, however, the consequences of these are yet to be investigated [Hermans et al., 2018b]. We also found most explanations to focus on 'storing information', which is again consistent with prior work [van der Werf et al., 2023] **(Chapter 4)**. Few other purposes of variables were mentioned, including tracking information, accessing information, and the ability to flexibly reuse data elsewhere in the code.

Hence we see room for using a wider variety of definitions and analogies and extending

the explanation to include different purposes. Domain isomorphic analogies [Bettin and Ott, 2023, Bettin et al., 2023], which are flexible in use across domains while preserving the analogical mapping, and roles of variables [Sajaniemi, 2002, Sajaniemi and Kuittinen, 2005] might be promising directions, keeping in mind students' background and cognitive load.

5.5.2 The concepts introduced near variables vary

Like prior work [van der Werf et al., 2023] **(Chapter 4)**, we found that variables are often taught in close connection to *data types*, *operators* (arithmetic expressions), and *control flow*. Additionally, we found that Scratch and Python textbooks introduce different programming concepts alongside variables. The order in which these concepts are introduced also differs between Scratch and Python books and among Python books. This raises questions such as when is the best moment to introduce variables, is there a "one-size-fits-all" trajectory crossing audience and programming language, or should such learning trajectories naturally depend on the programming language (and audience). Rich et al. [Rich et al., 2017, Rich et al., 2022] advocate for a language-independent learning trajectory focused on variables. However, our results hint towards current learning trajectories being influenced by language. This then also raises the question of how different trajectories influence transfer from Scratch to Python (or another programming language). Moreover, the variations we found within Python programming books suggest that a single "natural" trajectory, as we found for Scratch books, might not exist for Python. Alternatively, there might be unclarity or disagreement among developers on what order is most desirable, for example in terms of prior knowledge, avoiding or tackling misconceptions carried over from other disciplines or languages, or varying teaching purposes or learning philosophies. If the order of concepts was chosen carefully by the books' authors, there is an opening to investigate underlying motivations.

5.5.3 Naming is addressed inconsistently

In line with related work [van der Werf et al., 2023, van der Werf et al., 2024c] **(Chapter 3, 4)**, we also see that when naming practices are introduced, most books focus on syntax rules that, when not adhered to, break the program. Sometimes community guidelines and naming conventions are mentioned, but these are not consistent between *and* within books, therefore some books even provide conflicting information. Although the effects of style and casing on a programmer's accuracy might be limited [Sharif and Maletic, 2010], inconsistent approaches could confuse a learner, or unintentionally undermine the development of a critical attitude towards naming.

A careless attitude can be further encouraged in a learner by unclear definitions or examples of what is a 'meaningful' name. We have seen most books telling their reader to use meaningful or descriptive names, but some without indicating why naming is important. Moreover, some of those do not give explicit examples of what is considered meaningful, mention that variables can be named anything, or use generic variable names themselves. The limited attention to what is meaningful could be explained by that developers of educational materials chose a 'constructivist' pedagogical approach, in which students themselves discover by example what is good naming [van der Werf et al., 2024c] **(Chapter 3)**. In fact, two Python books (P2, P8) hint at using such an approach, writing that with

experience 'you will naturally know how to name [variables]'. However, for students to learn by example, we would expect the given guidelines and examples to be more consistent with each other. Perhaps we would even expect more emphasis on why naming is important rather than on certain rules and guidelines. Any inconsistencies, together with a limited explanation of why naming is important, could insinuate that one does not need to pay attention at all to naming practices.

Therefore, we suggest that our results demonstrate a potential misalignment between developers of educational materials and what research already knows is important for comprehension. Because our results are in line with prior work [van der Werf et al., 2023, van der Werf et al., 2024c] **(Chapter 3, 4)**, we suggest that if we want students to adopt good naming practices and develop a critical attitude, developers of educational materials *and* practitioners pay attention to how they address naming practices and be consistent in their approach. Moreover, considering that naming is context-dependent, there is room to focus on what makes a name (in)appropriate and why.

5.5.4 Limitations

Since our research analyzed only a limited number of books, our results might not be representative. However, by selecting the most popular books from Amazon, we aimed to include those books that people are most likely to buy and be exposed to, now and in the (near) future. However, even though Amazon is a popular platform, we cannot say if these bestsellers represent the books children and adults are truly exposed to. Using other (local) platforms or renewing the search at a different time might result in a different selection of books and hence influence our results. Nevertheless, the results we found correspond with results from prior studies, which suggests that our selection of books is reasonably representative. Even so, since most of the books included in this study were published relatively recently, older books, which could be designed differently, may likely still be in use. Finally, Scratch and Python are the languages most used by children. Had we focused on adults, other programming languages should be taken into account. We expect some differences due to the nature of the language, just like we found between Scratch and Python.

5.6 Concluding remarks

Our observations strengthen existing insights into how variables are presented in programming MOOCs, and extend them to programming textbooks for children and novices. More attention in research is needed to, for example, when to introduce the topic within the curriculum (in which language). Our insights also call for a (more) careful approach regarding variables and their naming, to be taken by educators and developers of learning materials in the fields of Computer Science and Software Engineering. Most importantly, we encourage the community to use (1) a wider range of definitions and analogies while teaching the concept of variables and (2) a more consistent teaching approach regarding variable naming that goes beyond syntax rules, personal preferences, and naming conventions. This includes a discussion on the importance of the topic and what makes a name (in)appropriate and why.