Surveying Upper-Secondary Teachers on Programming Misconceptions

Luca Chiodini
luca.chiodini@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Joey Bevilacqua
joey.bevilacqua@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Matthias Hauswirth
matthias.hauswirth@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Abstract

Background and Context Misconceptions in programming have been studied extensively, but most research focuses on uncovering and assessing misconceptions in students. When teachers are involved, it is usually only to elicit their perspective on misconceptions in their students. However, there is no guarantee that teachers do not hold misconceptions themselves. Detecting the possible presence of misconceptions in teachers is a crucial step for improving their content knowledge and pedagogical content knowledge, which benefits hundreds of students each year.

Objectives The study aims to answer the following research questions: Which programming misconceptions do teachers themselves hold? Are teachers aware of these misconceptions, do they observe them in their students, and do they consider them important? Are there differences in the teachers' perspectives, depending on whether they hold misconceptions themselves? Which strategies do teachers employ to deal with the misconceptions in their students?

Method We conducted an extensive, 55-page-long survey of upper-secondary informatics teachers who teach programming in Python. The survey focused on 16 Python misconceptions reported in prior research that involve concepts covered in the teachers' upper-secondary courses. The first part of the survey assessed whether the teachers held misconceptions, probing their knowledge with two related questions for each misconception and asking for mandatory explanations. The second part of the survey asked teachers whether they previously knew about the misconceptions, how prevalent they were in their students, how important they believe them to be, and how they could tell that their students hold the misconceptions.

Findings The number of teachers who gave incorrect answers on programming misconception questions varies considerably by misconception, ranging from 3 % to 40 %. Most teachers report being familiar with the misconceptions that were part of the study, consider them rather important, and have observed them at least once in their students. Teachers who answered correctly consistently rate misconceptions as more important and more prevalent among their students. Strategies to deal with the misconceptions include ways to prevent, detect, and fix them.

Implications When teachers hold misconceptions, all of their students can be affected. This study highlights the importance of professional development for teachers so that they can both correct their own misconceptions and recognize them in their students, ultimately leading to better programming education. Our results also caution computing education researchers against assuming that teachers are free from misconceptions. We recommend that future studies include an assessment of the participants' knowledge, to ensure that findings are properly contextualized.

CCS Concepts

• **Social and professional topics** → **Computing education**.

Keywords

introductory programming, Python, misconceptions, teachers

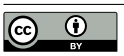
ACM Reference Format:
Luca Chiodini, Joey Bevilacqua, and Matthias Hauswirth. 2025. Surveying Upper-Secondary Teachers on Programming Misconceptions. In *ACM Conference on International Computing Education Research V.1 (ICER 2025 Vol. 1), August 3–6, 2025, Charlottesville, VA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3702652.3744227

1 Introduction

When thinking about misconceptions, the first thought often goes to the struggling student who is having a hard time mastering new concepts. A second thought may then focus on the teacher, to consider the benefits of teachers being aware of misconceptions that their students can hold, so teachers can develop appropriate pedagogies. Rarely does one consider the possibility that teachers themselves may hold some of those misconceptions. The possibility of teachers holding misconceptions as well should not appear too surprising: after all, every teacher was once a student learning the very concepts they now teach.

As computer science has increasingly become a mandatory part of the school curriculum, the need for training new teachers has risen all over the world in the past few years. Several countries created various training programs to meet this demand [5, 13]. Due to time and financial constraints, these training efforts do not always fully achieve the intended learning goals. As a result, teachers may enter the classroom without extensive and adequate preparation. Teachers themselves sometimes voice concerns about not feeling comfortable with their preparation [32].

Research in computing education has extensively focused on misconceptions, dedicating a lot of attention to students' difficulties in introductory programming [23]. Prior research has also investigated the teachers' opinions on the importance and prevalence of misconceptions in introductory programming courses [4, 26]. In this work, we investigate teachers' programming knowledge and use this data to weigh their opinions on the misconceptions.



ICER 2025 Vol. 1, Charlottesville, VA, USA

ACM ISBN 979-8-4007-1340-8/2025/08
https://doi.org/10.1145/3702652.3744227

We conduct an extensive, 55-page-long survey with $N = 97$ upper-secondary informatics teachers[1].

We ask the following research questions:

RQ1 Which previously reported programming misconceptions do upper-secondary informatics teachers hold?

RQ2 What are the teachers' perspectives on those misconceptions: Are they aware of them? How frequently do they observe them in their students? How important do they perceive them to be?

RQ3 Are there differences in the teachers' perspectives, depending on whether they hold misconceptions themselves?

RQ4 Which strategies do teachers employ to deal with the misconceptions in their students?

2 Background and Related Work

In 1986, Shulman proposed three categories to structure the knowledge of teachers with respect to the content: "subject matter content knowledge", "pedagogical content knowledge", and "curricular knowledge". First, it is argued that teachers should possess excellent subject matter content knowledge: teachers should have a level of understanding "at least equal to that of his or her lay colleague, the mere subject matter major" and "not only understand *that* something is so" but also "*why* it is so" [28]. The second category, termed pedagogical content knowledge (PCK), includes various techniques to explain content so that it is "comprehensible to others", as well as "what makes the learning of specific topics easy or difficult" [28]. PCK also includes knowledge of students' preconceptions, which often are misconceptions. This knowledge is helpful to assist learners in reorganizing their understanding.

2.1 Programming Misconceptions in Computing Education Research

Computing education research has a long history of investigating misconceptions (e.g., [20, 29]). Lewis et al. [23] provide an overview of research on misconceptions in computer science. In this work, we direct our attention specifically to *programming misconceptions*. In 2017, Qian and Lehman [27] conducted a literature review that attempted to systematize the broad perspective embraced by the research community over decades. This extends to debating the very definition of what constitutes a misconception, which the authors suggest could be defined as "an error in conceptual understanding".

More recently, Chiodini et al. [10] proposed a definition for a subset of programming misconceptions, which they term "programming language misconceptions" and define as "statements that can be disproved by reasoning entirely based on the syntax and/or semantics of a programming language". They published an online inventory[2] of misconceptions divided by programming language, to acknowledge that some claims that are wrong in one language may be correct in another. Misconceptions are presented with "refutation texts" and "refutation images", where the correct and the incorrect claims are juxtaposed. Keeping these two parts together, one that describes a common misconception, and right next to it one that *refutes* the incorrect conception and instead presents the correct one, has been shown to be helpful to induce conceptual change [30]. Theories of conceptual change started with Posner et al., acknowledging that "learning is the result of the interaction between what the student is taught and his current ideas" [25]. The study of misconceptions is thus important for understanding the incorrect conceptions that learners hold and need to overcome.

Lu and Krishnamurthi [24] acknowledge the language specificity of misconceptions but identify the existence of a common set of core features that are shared by many modern programming languages, and that can be used to ground misconceptions in it. They developed an instrument consisting of multiple-choice questions with curated distractors, and then used it to assess students' understanding. The study noted that the misconceptions are rather widespread, even among students in their third or fourth year at a selective university.

2.2 Teachers and Misconceptions

Research on misconceptions in computing education typically focuses on students. When teachers are brought into the picture, they are usually asked about their students. As an example, Qian et al. [26] surveyed 44 high school teachers of an introductory programming course in Python in the United States. Teachers were asked to rate how often they encountered each misconception, the perceived importance, and how confident the teacher is in addressing the misconception with their students. The study included 37 misconceptions, cataloged under five different topics. The authors suggest that some teachers struggled to understand the misconception statements, possibly due to limited content knowledge. The study did not explore whether teachers themselves held these misconceptions, nor did it collect explanations that could clarify their difficulties in understanding the questions.

However, relying on the teachers' judgments without any form of assurance can produce misleading results. Brown and Altadmri [4] showed how the teachers' beliefs about the kind and prevalence of mistakes students make when programming in Java do not match the evidence offered by analyzing a large repository of code written by students. To the authors' surprise, teachers with more years of experience (in teaching overall, in teaching programming, and in teaching programming in Java) did not give better predictions for the ranking of errors.

Chiodini et al. [9] describe a tool that leverages progmiscon.org's refutation texts to build lightweight assessments (called "conceptual checks") on programming misconceptions for teachers, but they did not conduct any study.

Some studies on teachers' knowledge have been conducted in the related domain of mathematics. Brown and Bergman [6] conducted a short survey with pre-service teachers to test their understanding of the concept of variables. Teachers were tested with items targeting known misconceptions in middle school students. The results demonstrated that misconceptions are also pervasive in teachers, with only 14 % of them answering correctly to all the four questions. Even [16] tested the understanding of functions in mathematics teachers for secondary education during the final year of their training. Many teachers were found to have inadequate conceptions of functions that went beyond outdated definitions. Teachers were asked to help hypothetical students that were struggling on

[1] Throughout the paper, we primarily use the terms "upper-secondary teachers" and "informatics", but some readers may find respectively the terms "high school teachers" and "computer science" more standard. Within this work, the terms are interchangeable.
[2] https://progmiscon.org

those concepts. Interviews with teachers revealed "a tendency to provide the student with the 'vertical line test' as a rule that the student can follow and get the right answers (without needing to understand)" [16].

The results of these studies may be attributed to each study's specific characteristics, but considered together they suggest the need to assess teachers' understanding to both recognize areas of knowledge that need to be improved and put into the right perspective the observations that come from teachers.

3 Methodology

This section describes in detail the methodology of our study, which consists of an extensive 55-page survey of upper-secondary teachers in Switzerland.

3.1 Context and Participants

We aimed to recruit participants among upper-secondary informatics teachers in Switzerland, where the mandatory upper-secondary informatics courses taught in the different cantons involve a significant part of programming, usually in Python.

We reached out directly to teachers who either participated in a national training program to prepare them to teach computer science in high school, or who had previously expressed interest in participating in research studies. The Swiss Association for Informatics in Education (SVIA-SSIE-SSII) also offered to contact its members (approximately 300 people who are predominantly upper-secondary informatics teachers), and pointed them to a form where interested teachers provided us with their contact details so we could invite them to our study. Through both channels, we invited 170 teachers to participate in the online survey. A total of $N = 97$ teachers completed the entire survey and were included in this study.

We estimated each participant would take approximately 90 minutes to complete the study. All invited participants were promised a remuneration for their participation in the form of public transport tickets roughly equivalent to 1.5 hours of their salary. There was no raffle: participants knew they could claim the remuneration if they completed the entire survey before the deadline. To ensure the anonymity of survey responses, information about the recipient of the remuneration was collected in a separate form, the link to which was provided at the end of the survey. The University Ethics Committee approved this study.

3.2 Selection of Programming Misconceptions

Acknowledging the differences in how misconceptions have been defined in the literature on programming education (Section 2.1), in this work we adopt the definition suggested by Chiodini et al. [10] to designate a specific subset of programming misconceptions. This viewpoint considers as "programming language misconceptions" the claims "that can be disproved by reasoning entirely based on the syntax and/or semantics of a programming language" [10].

We focus on 16 previously documented Python programming misconceptions. We started with all published Python misconceptions on the progmiscon.org misconception inventory [10] and we removed those related to programming concepts not typically taught in the mandatory upper-secondary informatics course in Switzerland. Table 1 lists all the 16 programming misconceptions included in our study and shows the concepts they involve (improving on the concept tags provided in the progmiscon.org inventory)[3].

We gathered the set of misconceptions for our study from a specific inventory, but several of them have also been documented in earlier studies and research outputs, often under different names.

The four misconceptions AssignCompares, NoReservedWords, NoShortCircuit, ParenthesesOnlyIfArgument match respectively the misconceptions documented by Hristova et al. [19] as number 1 ("confusing the assignment operator with the comparison operator"), 4 ("confusing 'short-circuit' evaluators with conventional logical operators"), 8 ("using keywords as method or variable names") and 10 ("forgetting parentheses after a method call").

Lu and Krishnamurthi [24] recently published a set of misconceptions that are common across modern programming languages. The misconception VariablesHoldExpressions corresponds to their misconception "DefByRef": "variable definitions alias variables". The misconception AssignmentCopiesObject, which claims that an assignment does not copy the reference to the object but the object itself, corresponds to their "DefsCopyStructs": "variable definitions copy structures recursively". "DefCopyStructs" was one of the most widespread misconceptions reported by Lu and Krishnamurthi [24], with 67 % of students asserting that it was the actual behavior of the language.

The misconception VariablesHoldExpressions is a well-documented novice difficulty, with evidence dating back to Du Boulay [14] in 1986 ("variable holds an unevaluated expression").

3.3 Survey

The survey was administered online and was structured as shown in Figure 1. After the informed consent and the demographic questions, the core of the survey consists of two main parts. Each of the two main parts started with a page to introduce the content and ended with a page that asked for general feedback. The entire survey consisted of 55 pages and was configured so that participants could not navigate back to prior pages to modify their answers. The complete version of the survey, which includes all the questions in full, is available in the online appendix [7].

3.3.1 Background Questions. After getting the consent from participants to include their answers in this study, we asked questions to understand their background in teaching at large and teaching programming specifically.

We asked teachers questions to characterize their background, such as in which years of school they are teaching programming, whether they are teaching only the mandatory course or also elective ones, how many hours of informatics per week they are teaching, and how many years of teaching experience they have overall and specifically in teaching informatics.

3.3.2 Part 1: Assessment. The core of Part 1 consisted of 32 pages, administered in random order. Each of the 16 misconceptions was featured on two pages. For each misconception, we used two different types of questions.

[3]We provide a brief interactive tutorial to explore the 16 misconceptions included in this study at https://pytamaro.si.usi.ch/curricula/luce/misconceptions.

Table 1: The 16 studied misconceptions and their coverage of programming concepts.

	Assignment	Boolean	Call	Composition	Conditional	ControlFlow	Equality	Expression	Function	Literal	Loop	Name	Operator	Return	Statement	Value	Variable
AssignCompares	✓						✓	✓					✓		✓	✓	✓
AssignmentCopiesObject	✓															✓	✓
ComparisonWithBoolLiteral		✓					✓	✓		✓			✓			✓	
ConditionalIsSequence					✓	✓											
DeferredReturn			✓			✓			✓					✓	✓		
IfIsLoop					✓	✓					✓				✓		
MapToBooleanWithIf		✓			✓	✓									✓	✓	
MultipleValuesReturn			✓	✓				✓	✓					✓	✓	✓	
NoAtomicExpression				✓				✓		✓		✓	✓				✓
NoReservedWords												✓					
NoShortCircuit		✓			✓	✓		✓					✓				
OutsideInFunctionNesting			✓	✓		✓		✓	✓								
ParenthesesOnlyIfArgument			✓					✓	✓								
ReturnCall			✓					✓	✓					✓	✓		
ReturnUnwindsMultipleFrames			✓						✓					✓	✓		
VariablesHoldExpressions	✓							✓					✓			✓	✓

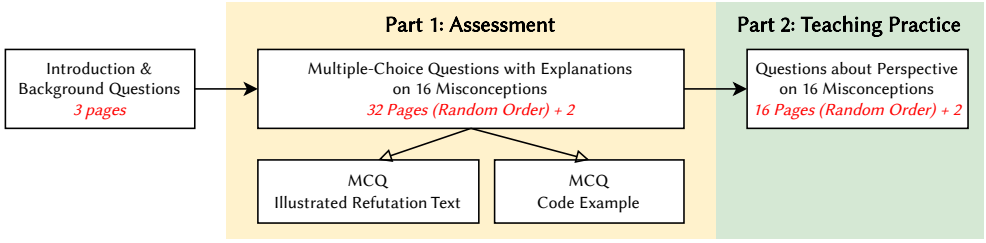
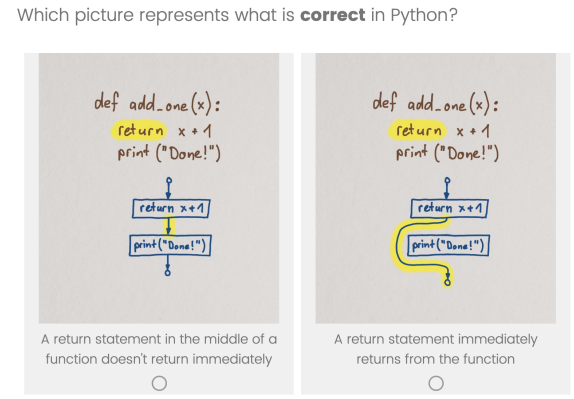
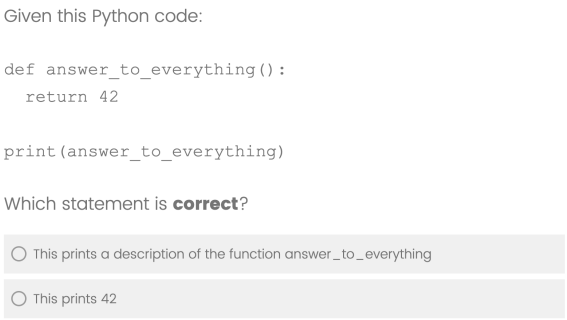


Figure 1: Structure and sequence of the 55-page-long survey.



(a) Question for misconception DeferredReturn: the refutation text is paired with an illustration.



(b) Question for misconception ParenthesesOnlyIfArgument: the two options refer to a example code snippet.

Figure 2: Examples of the two question types used in Part 1 for each misconception.

"Illustrated refutation" questions (Figure 2a) are inspired by the images published on the progmiscon.org inventory. They contain the refutation texts—correct and incorrect claims—paired with illustrations that we redrew to improve clarity and consistency. The two contrasting illustrations depict, respectively, the correct conception and the misconception. The illustrations include code snippets, annotations, and notional-machine-based diagrams such as memory diagrams. They reflect what a teacher could draw on a whiteboard to explain a misconception. They are inspired by the notional machines documented by Fincher et al. [17], such as "Control Flow as Graph" (Figure 2a) and "Expression as Tree" (Figure 5a).

Code example questions (Figure 2b) include a small code example and a correct and incorrect statement about the code. They often are code tracing questions, but sometimes they are questions about other aspects (e.g., the syntax).

Both question types are multiple-choice questions with two possible options, one representing the incorrect conception (misconception), the other representing the correct conception. The two options are presented in random order. Each question asks participants to pick the option that describes the correct conception.

Prior research has noted that multiple-choice questions can lead to imprecise measurements [21], because participants can guess the answer or misinterpret the question. For these reasons, to increase the validity of our assessment, each multiple-choice question was accompanied by a mandatory free-text explanation field. Below both question types, teachers were asked to "Briefly explain why:" in a text field. (This would appear right below Figure 2a and Figure 2b; we omit it here for space reasons.)

3.3.3 Part 2: Teaching Practice. The core of Part 2 included 16 pages, one page for each misconception. In this part as well, the pages were presented in random order. As with the entire survey, participants could not navigate back, which means that they had to answer the assessment questions that tested their understanding before starting with Part 2.

Each page on Part 2 presented a misconception as shown on progmiscon.org and asked the following questions:

- Did you know this is a misconception your students might have? ["Yes", "No"];
- Did you observe this misconception in your students? ["Frequently", "Multiple times", "At least once", "Never", "I don't know (I might have had that misconception myself)"];
- In your opinion, how important is it to detect and fix this misconception in your students? ["Important", "Somewhat important", "Not so important", "Not important", "I'm not sure"];
- How could you tell your student(s) held this misconception? [Free-text response].

3.4 Analysis

We automatically assess the correctness of the answers to the multiple-choice questions in Part 1 (Section 3.3.2) with a binary outcome. Answering those questions was mandatory. We manually assess the explanations following the procedure suggested by Chiodini and Hauswirth [8] to cross-check explanations with multiple-choice answers. Adamoli [1] also followed a similar approach to validate a concept inventory of programming misconceptions: students were asked to clearly explain why they chose a particular option in the multiple-choice question, and experts decided on the actual presence of a misconception based on these explanations.

We classify each explanation into one of the following groups: "correct", when the explanation showed that the participant understood the question and explained the correct answer in a reasonably sufficient way; "imprecise", when the explanation provided insufficient details to assess whether the participant holds the correct conception (e.g., because they misunderstood the question and focused their explanation on a different aspect), "wrong", when the explanation clearly showed that the participant holds the wrong conception, and "missing" in the remaining cases (e.g., a participant stating that they did not know the answer and they guessed, or an explanation containing only gibberish).

We then combine the assessment of the answer of the multiple-choice question and the corresponding explanation to determine the final correctness of an answer. An answer is correct when either (1) the multiple-choice answer is correct, and the explanation is correct or imprecise; (2) the multiple-choice answer is incorrect, but the explanation is correct. In the remaining cases, the answer is considered wrong (i.e., the multiple-choice answer is correct but the explanation is wrong or missing, or the multiple-choice answer is wrong and the explanation is imprecise, wrong, or missing).

For the second part of our survey (Section 3.3.3), we answer the second research question analyzing the distribution (e.g., to find the mode) of the options for questions that use a Likert or binary scale.

To answer RQ3, we analyze data in two ways. On one hand, we discard the "I don't know" option and interpret the ratings as interval data on a four-point scale from 1 to 4, noting that there is no consensus on whether it is acceptable to do so and that this treatment hides important nuances [2]. On the other hand, we compare the ratings given by two groups of teachers, those who gave correct and wrong answers, by conducting a two-tailed Mann–Whitney U test for independent samples. This non-parametric test is appropriate for our data, as it properly accounts for the ordinal nature of the ratings [18]. We adopt the standard significance level $\alpha = 0.05$ and compute the effect size using Cliff's d, which measures how often a score from one group is higher than a score from the other [12].

RQ4 calls for a qualitative exploration. We analyze the answers provided by the teachers to the question with a free-text response in the second part of the survey: "How could you tell your student(s) held this misconception?". We perform a thematic analysis [3, 22] to identify common patterns across teachers and misconceptions. We illustrate the themes with a description paired with quotes as written by the teachers, after translating them into English where necessary, adjusting the typography, correcting spelling errors, and omitting details that may identify the author.

4 Results

We present the results of the study in the order of our research questions. To answer the entire survey, teachers spent an average of 112 minutes (excluding the bottom and top quartiles to remove outliers). This is somewhat longer than what we anticipated but can

be explained by the large number of textual explanations required in the survey.

4.1 Teacher Background

The vast majority of the 97 responding teachers reported teaching in the 10th grade (95 %) and 11th grade (84 %). This corresponds to the two years in which the mandatory programming course typically takes place in Swiss high schools. A number of teachers also work with older students, in elective courses: 35 % of them teach elective programming courses in the 12th grade and 23 % in 13th grade. Overall, 99 % of the respondents are currently teaching a mandatory high school programming course and 31 % teach at least one elective programming course.

The responding teachers report quite an even distribution of school hours of informatics taught per week, on average. 24 % of them are currently teaching less than 6 school hours of informatics per week, 34 % from 6 to 12 school hours, 28 % from 12 to 18 weekly school hours, and the remaining 14 % more than that, up to 27 school hours per week.

The majority of the teachers (68 %) are also currently teaching other subjects. This is expected for our context, as a significant fraction was already teaching a different school subject before receiving training to teach informatics in upper-secondary. 57 % of the teachers reported studying in one of the 3 variants of the national training program.

Participants reported an average length of their teaching career of 12.2 years (minimum 2, maximum 32 years). Their experience with teaching informatics was shorter, with an average of 6.2 years (minimum 0, maximum 25 years). 52 % of teachers have been teaching informatics for no more than four years.

31 % of teachers indicated female as a gender, 68 % male, and 1 % preferred not to disclose such information.

4.2 Teacher Assessment on Misconceptions (RQ1)

As described in the Methodology (Section 3), each of the 97 participating teachers chose between two options for each of the 32 questions that targeted our 16 programming misconceptions. Giving an answer to the multiple-choice questions was mandatory. Overall, 89 % of the answers were correct and 11 % were wrong. We note a minimal difference between the two question types: answers to multiple-choice questions in the illustrated refutation text format were wrong 10 % of the times, while answers to questions using the code example format were wrong in 13 % of the cases.

We then separately manually analyzed each of the 3 104 textual explanations. Some explanations included a note from the teacher indicating that they had already given an explanation for the same question in the other format, which appeared earlier. The order of the questions in the survey was fully randomized, so this reference to an earlier question may be to either of the two question types. We accommodated these requests by considering such earlier explanations when assessing the one coming later. Overall, we classified 77% of the explanations as "Correct", 11 % as "Imprecise", 6 % as "Wrong", and the remaining 6 % as "Missing".

We combined the automatic assessment of the multiple-choice question and the manual assessment of the explanation to get a more reliable, final assessment of the correctness of each answer (details in Section 3.4). Overall, this led to a modest change of the percentage of incorrect answers, which increased from 11 % to 14 %.

These numbers lump together all the misconceptions and obscure the large differences among them. Table 2 presents the data grouping the assessment per each of the 16 programming misconceptions. The breakdown per misconception reveals larger changes due to the explanations (from −10 to +15 percentage points change in wrong answers). The non-negligible differences confirm the necessity of augmenting multiple-choice questions with explanations, as proposed by Chiodini and Hauswirth [8], to detect more accurately which misconceptions teachers hold. In the final assessment, wrong answers vary from 3 % of AssignCompares to 40 % of NoAtomicExpression.

Figure 3 shows, for each of the 97 teachers, the misconceptions they hold (i.e., the ones for which they answered incorrectly in at least one of the two questions). On average, a teacher holds 3.3 misconceptions, with a standard deviation of ±2.7 misconceptions. As Figure 3 illustrates, the misconceptions we detected are distributed among the teachers, with a minimum of 0 and maximum of 12 misconceptions, and are not concentrated only in a handful of teachers. When considering the two most frequently detected misconceptions (NoAtomicExpression and ReturnUnwindsMultipleFrames), only 16 % of all teachers are holding both misconceptions.

4.3 Teacher Perspectives on Misconceptions (RQ2)

The second part of the survey was meant to elicit the teachers' perspective on the programming misconceptions that were part of the study. Teachers were asked to rate the importance of each misconception and the prevalence among their students. We also asked the teachers whether they were familiar with the existence of each misconception. Table 3 shows the number of teachers who selected each option. 10 out of 16 misconceptions were rated at the highest level of importance by the relative majority of the teachers. In terms of prevalence, all misconceptions have been seen at least once by several teachers. The mode indicates 6 misconceptions seen multiple times and 8 misconceptions as never seen.

The absolute majority of teachers reported being familiar with 13 of the 16 misconceptions included in our study, with a peak of 85 % for the AssignCompares misconception. NoAtomicExpression and ReturnUnwindsMultipleFrames were the two misconceptions teachers were the least familiar with. The same two misconceptions are also the ones with the highest percentage of wrong answers in the assessment part of the survey (Section 4.2). This suggests that a number of teachers do not know what is correct, and consequently do not know that the incorrect claim can be a student misconception. Almost all teachers instead correctly answered the questions targeting the third misconception of this group, IfIsLoop. Most teachers report never having seen it in class, but an overwhelming majority still considers it important.

Table 2: For each misconception, the percentage of wrong answers on the multiple-choice question in the illustrated refutation text type (Illustr. MCQ) and in the code example type (Code MCQ). The "Final" columns show the percentage of wrong answers after combining the multiple-choice answer with the explanation (in parentheses, the difference in percentage points). In the last column, the percentage of teachers with at least one wrong final answer (i.e., one or both question types have a wrong final answer). This table and the subsequent ones are sorted by this last column.

Misconception	Illustr. MCQ	Illustr. Final	Code MCQ	Code Final	Miscon. Final
NoAtomicExpression	18 %	33 % (+15 pp)	21 %	31 % (+10 pp)	40 %
ReturnUnwindsMultipleFrames	16 %	24 % (+ 7 pp)	26 %	32 % (+ 6 pp)	38 %
ConditionalIsSequence	29 %	31 % (+ 2 pp)	16 %	14 % (− 2 pp)	35 %
NoShortCircuit	23 %	25 % (+ 2 pp)	19 %	16 % (− 2 pp)	31 %
MultipleValuesReturn	20 %	20 % (+ 0 pp)	36 %	26 % (−10 pp)	30 %
AssignmentCopiesObject	18 %	23 % (+ 5 pp)	14 %	21 % (+ 6 pp)	27 %
MapToBooleanWithIf	6 %	15 % (+ 9 pp)	10 %	19 % (+ 8 pp)	26 %
ComparisonWithBoolLiteral	8 %	14 % (+ 6 pp)	16 %	14 % (− 2 pp)	22 %
ReturnCall	2 %	8 % (+ 6 pp)	8 %	14 % (+ 6 pp)	20 %
ParenthesesOnlyIfArgument	0 %	5 % (+ 5 pp)	12 %	12 % (+ 0 pp)	15 %
NoReservedWords	6 %	10 % (+ 4 pp)	4 %	6 % (+ 2 pp)	14 %
OutsideInFunctionNesting	5 %	6 % (+ 1 pp)	7 %	8 % (+ 2 pp)	11 %
DeferredReturn	3 %	7 % (+ 4 pp)	7 %	8 % (+ 1 pp)	9 %
VariablesHoldExpressions	4 %	6 % (+ 2 pp)	6 %	3 % (− 3 pp)	8 %
IfIsLoop	1 %	1 % (+ 0 pp)	2 %	3 % (+ 1 pp)	4 %
AssignCompares	1 %	3 % (+ 2 pp)	1 %	0 % (− 1 pp)	3 %

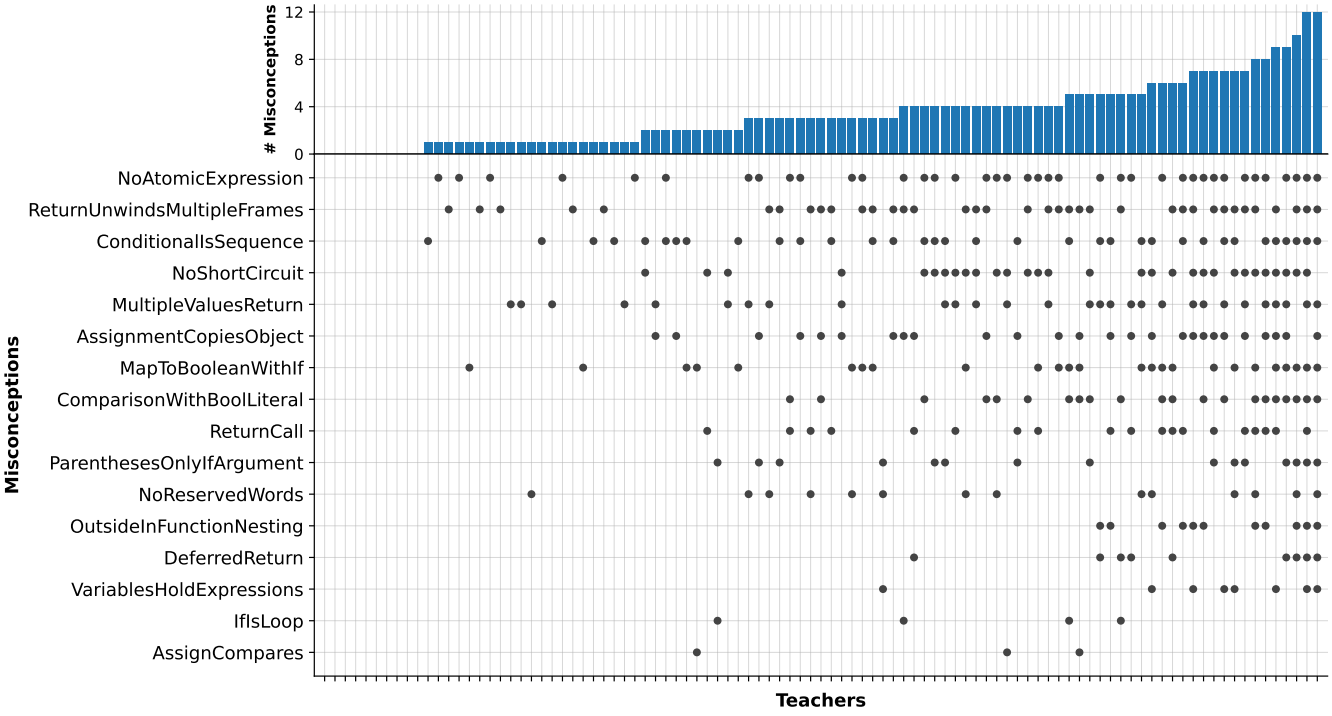


Figure 3: For each teacher, number of misconceptions (above) and the actual misconceptions (below) held.

Table 3: Number of answers for the multiple choice questions about the importance, prevalence and familiarity of each misconception (refer to Section 3.3.3 for the full version of the questions). The most selected option of each question is bold.

	Importance					Prevalence					Familiar	
	Important	Somewhat Imp.	Not So Imp.	Not Important	I'm not sure	Frequently	Multiple Times	At Least Once	Never	I don't know	Yes	No
NoAtomicExpression	18	25	**26**	18	10	8	16	9	**40**	24	42	**55**
ReturnUnwindsMultipleFrames	**34**	26	15	11	11	3	12	11	**54**	17	45	**52**
ConditionalIsSequence	**51**	28	8	5	5	16	**34**	16	21	10	**78**	19
NoShortCircuit	18	28	**35**	11	5	7	19	20	**33**	18	**68**	29
MultipleValuesReturn	32	**34**	23	5	3	3	24	19	**32**	19	**61**	36
AssignmentCopiesObject	**46**	28	9	6	8	13	**35**	13	18	18	**74**	23
MapToBooleanWithIf	10	**38**	26	17	6	18	**37**	15	15	12	**73**	24
ComparisonWithBoolLiteral	12	30	**34**	19	2	28	**34**	15	10	10	**82**	15
ReturnCall	11	17	30	**34**	5	18	21	23	**24**	11	**60**	37
ParenthesesOnlyIfArgument	**68**	17	8	2	2	**33**	30	18	14	2	**81**	16
NoReservedWords	**46**	25	21	4	1	9	21	**37**	26	4	**83**	14
OutsideInFunctionNesting	**55**	26	8	1	7	6	21	15	**46**	9	**64**	33
DeferredReturn	**64**	28	4	0	1	14	**32**	21	24	6	**71**	26
VariablesHoldExpressions	**63**	14	10	3	7	7	24	17	**42**	7	**58**	39
IfIsLoop	**62**	19	9	3	4	7	17	17	**51**	5	43	**54**
AssignCompares	**71**	19	5	1	1	30	**38**	14	10	5	**85**	12

4.4 Differences In Perspective Depending on Correctness (RQ3)

To understand if and to what extent there are differences in how teachers rate the misconceptions, depending on whether they themselves can correctly answer questions about them, we analyzed separately the ratings expressed by the two groups. (As discussed in Section 3.4, we remark that this analysis treats Likert scales as interval data to be able to compute numerical averages, but doing so hides important qualitative differences between the options.)

Teachers who gave incorrect answers to the misconception questions give an average rating of importance of 2.97 and a rating of prevalence of 2.08. When we focus exclusively on the teachers who gave correct answers, both ratings increase, to 3.11 for importance and 2.32 for prevalence. Figure 4 shows the differences in the ratings by misconception.

After separating the ratings into two groups, those given by the teachers who answered correctly and those who answered incorrectly, we conducted two-tailed Mann-Whitney U tests to check whether one group expressed higher ratings than the other. Statistically significant differences are revealed: teachers who answer correctly indicate that the misconceptions are more prevalent in their students ($U = 128810.5$, $p < 0.001$, effect size $d = 0.23$) and also consider the misconceptions more important ($U = 110873.0$, $p < 0.001$, effect size $d = 0.15$).

The differences persist also for the teacher's familiarity with the misconceptions. 73 % of the teachers who answered correctly the assessment questions indicate being familiar with the misconceptions, as opposed to only 54 % of those who answered wrongly. The difference between groups is statistically significant ($U = 160944.0$, $p < 0.001$, $d = 0.19$).

4.5 Teachers' Strategies to Deal With Misconceptions (RQ4)

In the second part of the survey, we asked teachers how they could tell their students held each misconception (Section 3.3.3). The question aimed to elicit the strategies adopted by teachers to detect misconceptions in their students. How teachers handle students' struggles is an important component of pedagogical content knowledge, but data from actual teachers is difficult to obtain [31].

Answering this question was not mandatory, as we considered that some teachers have limited experience in teaching informatics and others may hold the misconception themselves. Teachers provided an answer in 76 % of the cases, yielding a total of 1 184 texts.

The majority of the answers (61 %) did not contain useful information. This is due to a large number of different reasons. Some teachers misunderstood the question, others expressed the realization that they too were holding a misconception, and other teachers commented on the misconception's importance or prevalence (*"this misconception is important, but I have to focus on other aspects of coding, short-circuit operators are not at the top of my priorities"*). One teacher asked for help dealing with a misconception in students: *"I really don't know. I wish I would. Can you help? I see this misconception a lot. It's hard to explain. [...] I've been 'fighting it' for a long time"*.

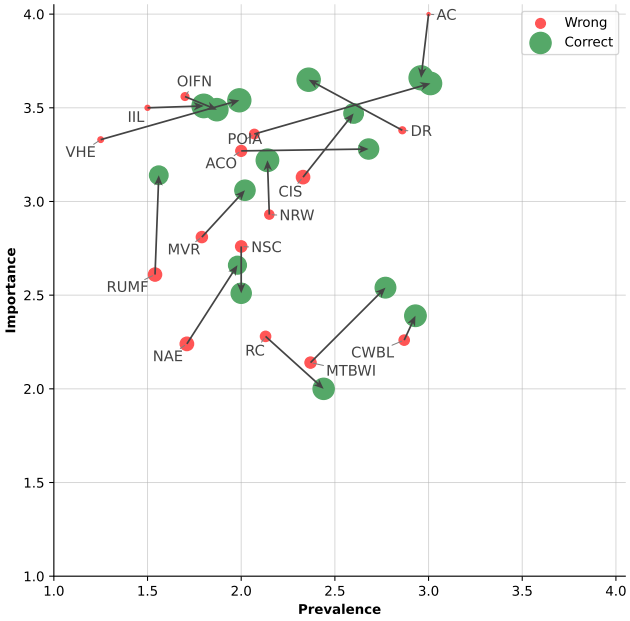


Figure 4: Change of the ratings for importance and prevalence of the misconceptions between teachers who answered wrongly (red) and those who gave correct answers (green). Dot sizes are proportional to the number of teachers who gave that answer (averaging importance and prevalence). Labels refer to misconceptions with their acronym.

The remaining 39 % of answers contained instead relevant information. Thematic analysis on these 461 answers resulted in seven different themes. These themes correspond to seven strategies that teachers use with three different aims: (i) prevent misconceptions from developing altogether, (ii) detect their presence, and (iii) fix them.

We report the themes in order of frequency. Each theme, corresponding to a strategy, is illustrated with a description, exemplary quotes from the teachers, and the aims (prevention, detection, fixing) it can serve.

(1) *Explain* (149 answers). The teacher actively addresses the misconception in their explanation. This may also involve the use of specific examples to highlight the cases in which holding the misconception leads to wrong predictions of how certain programs behave, as opposed to the actual execution by a Python interpreter. For example, to fix the DeferredReturn misconception, a teacher explicitly informs students that code written after a return statements is not evaluated: *"I clearly point out that after the return statement, no code will be regarded within the function body"*. Or, for the ConditionalIsSequence misconception: *"I covered this myself in the introduction of* `if-elif-else`*, I showed an example where the change to* `if-if-else` *generates a different output"*. This strategy can be used both to prevent and fix misconceptions.

(2) *Test with Exercise* (90 answers). The teacher tests the students for specific misconceptions using coding or tracing exercises. For example, a teacher reported testing for the NoAtomicExpression misconception during an exam, in which the majority of students failed to identify `while x:` as valid Python code, while they were able to recognize the validity of `while True: :` *"In a multiple choice question during the first graded quiz of a programming course in grade 11, I asked the students to identify which headers of a* `while` *loop were syntactically correct. Almost nobody selected the option* `while x:`*, even if it was near* `while True:` *which many recognized"*. This anecdote shows the importance of explicitly teaching expressions (cf. Section 5.2.1), a concept teachers sometimes neglect [15]. For OutsideInFunctionNesting, a teacher reports using questions similar to those found in our survey: *"It is easy to detect with questions like in this survey"*. This strategy is mainly about detecting misconceptions.

(3) *Receive Student Report* (58 answers). The teacher responds to some student report that could be triggered by the presence of a misconception and addresses it. A teacher anecdotally reports that their students "*panic*" and ask for help whenever they encounter *strange* errors caused by the lack of parentheses in a supposed function invocation. For the teacher this could signify the presence of the ParenthesesOnlyIfArgument misconception, which then needs to be addressed: *"They 'panicked' that something really strange happened. As always, I point out how IMPORTANT it is to follow the syntax. This happens most frequently when I introduce functions"*. Another teacher reports the following: *"They are puzzled as to why their array changed even thought they had changed another copy of the array [...]"*. This is a clear symptom of the AssignmentCopiesObject misconception. This strategy too has the aim of detecting misconceptions.

(4) *Notice Code Pattern* (55 answers). Another strategy to detect misconceptions: the teacher observes patterns in students' code that suggest the student could hold a specific misconception. For instance, teachers may look for the presence of the = token in the condition of `if` statements to detect the AssignCompares misconception: *"They use '=' in* `if`*-statements"*.

(5) *Use Program Visualization* (55 answers). The teacher employs some form of program visualization (such as a notional machine [17] or the debugger) to help students understand aspects of programming in ways that fix misconceptions. As an example, teachers may ask students to draw a flowchart to reason about the difference between `if` and `while` statements, to address the ConditionalIsSequence misconception: *"[The misconception] can be demonstrated and taught through concrete examples with a flowchart. I find this to be the most effective solution"*. This strategy can be used to detect, prevent, or fix misconceptions, depending on the moment and the context in which the program visualization is used.

(6) *Connect to Math* (32 answers). The teacher deals with the misconception by connecting to mathematical concepts. For OutsideInFunctionNesting, teachers reported referring to the same concept of function nesting in mathematics to

emphasize the importance of the evaluation order of nested function invocations: *"Since this is a concept also used in mathematics, it is even more important to them, and they already have some experience with it"*. This strategy can be used to either prevent or fix misconceptions.

(7) *Rely on Compiler Tools* (23 answers). The teacher relies on the compiler to address the misconception. Compiler tools can detect various issues with respect to the syntax and static semantics of programs. This category also includes tools that rely on compiler tools to report errors, such as editors with syntax highlighting. Teachers reported relying on `SyntaxErrors` generated by the parser to ensure that students do not use reserved keywords as variable names, to address NoReservedWords: *"If they try [to use a reserved word as the name of a variable], it gives them errors. They will learn 'on their own'"*. This strategy can be used to both detect and fix misconceptions.

We note that the strategies reported by teachers vary considerably depending on the specific misconception. This is expected, as not all strategies are adequate to prevent, detect or fix all the misconceptions, and part of what makes teaching effective is possessing a repertoire of techniques to choose the right one for the situation at hand.

5 Discussion

In summary, our results show that (1) teachers, and not only students, can hold misconceptions: for the 16 misconceptions included in our study, the percentage of teachers holding a specific misconception varies between 3 % and 40 %; (2) teachers are familiar with the misconceptions that were part of the study, they consider them quite important, with a few exceptions, and they express different views on whether they have seen them in their students; (3) teachers who correctly answered the assessment questions indicate being more familiar with the misconceptions, claim that they are more important, and report having seen them more often in their students. In the discussion section, we dive deeper into some aspects worth examining.

5.1 Multiple-Choice Questions and Explanations

Researchers in the learning sciences have extensively studied the issue of creating validated instruments to assess knowledge as accurately as possible. There have also been efforts in computer science education research to develop validated instruments, known as concept inventories. Unfortunately, researchers often work within specific contexts and this requires that they tailor their questions to examine the intended area of knowledge.

This specific study dealt with programming as taught in Swiss high schools, thus we selected among existing programming misconceptions the ones relevant for our context (cf. Section 3.2) and developed an ad hoc instrument with multiple-choice questions of two types. Following the recommendation by Chiodini and Hauswirth [8], we paired the multiple-choice questions with a free-text explanation, which we then manually analyzed. The analysis of these explanations suggests four methodological considerations.

First, the explanations indicate that some participants can indeed misunderstand the questions, and therefore do not give the answers according to what the researcher expects. For example, a teacher wrote this explanation to the question shown in Figure 5a: *"an expression is something that gives a result and can be evaluated.* `v = 19` *and the others, are not expressions"*. This participant thought the question was asking whether the entire statements, such as `v = 19`, and not only the highlighted parts, were expressions.

Second, we assessed misconceptions with two related but different questions. The differences go beyond the pure format of the question and include using different code fragments, as seen in Figure 5. This may not constitute a source of difference for the specific misconception of Figure 5, which indeed exhibits a similar rate of wrong answers for the two question types (33 % for the illustrated refutation and 31 % for the code example). Other misconceptions, however, show rather large discrepancies, likely due to the information provided in the questions. Figure 6 shows the two questions that involve the misconception ConditionalIsSequence. While the question shown in Figure 6a uses generic functions without showing their implementation, the question in Figure 6b contains concrete fragments of code that modify a variable `flag`, showing a specific program that falsifies the claim "`if-else` is equivalent to sequence of two `if`s".

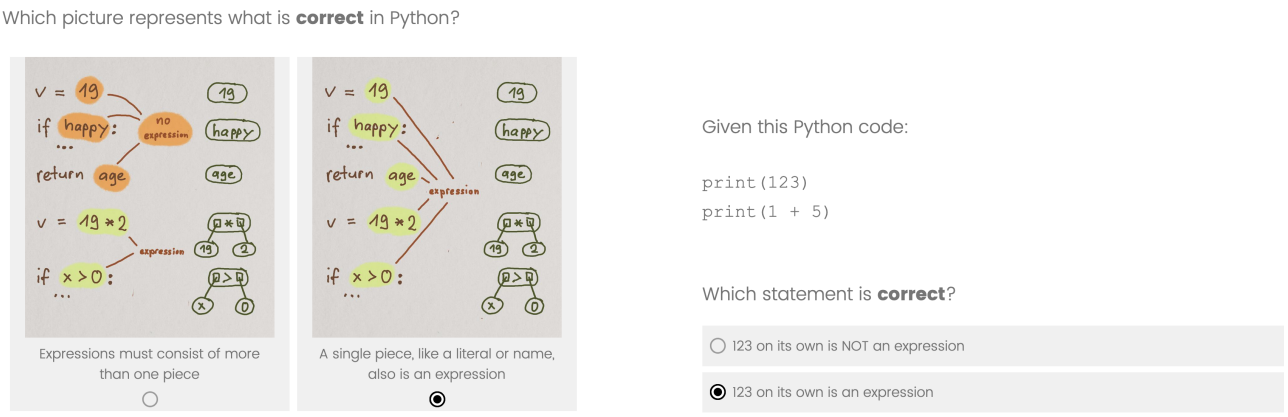
Third, the way a certain question is formulated may have brought about some learning on the spot. Figure 6b can serve again as an example: a teacher may have never thought about the difference between an `if/else` and a sequence of `if/if not`, but assuming that they are able to accurately trace both programs, they could realize that the claim is false in general and acquire the correct conception in the moment.

Fourth, as a consequence of the previous two points, the difference in correctness between the two question types could be wider than what we actually measured. Participants answered the two question types for the same misconception in a random order. If they acquired some knowledge from the first question type, and they were able to transfer and use that knowledge for the second question type (e.g., if they saw Figure 6b before Figure 6a), this may have impacted the correctness on the questions.

5.2 Teachers' Explanations on Questions about Programming Misconceptions

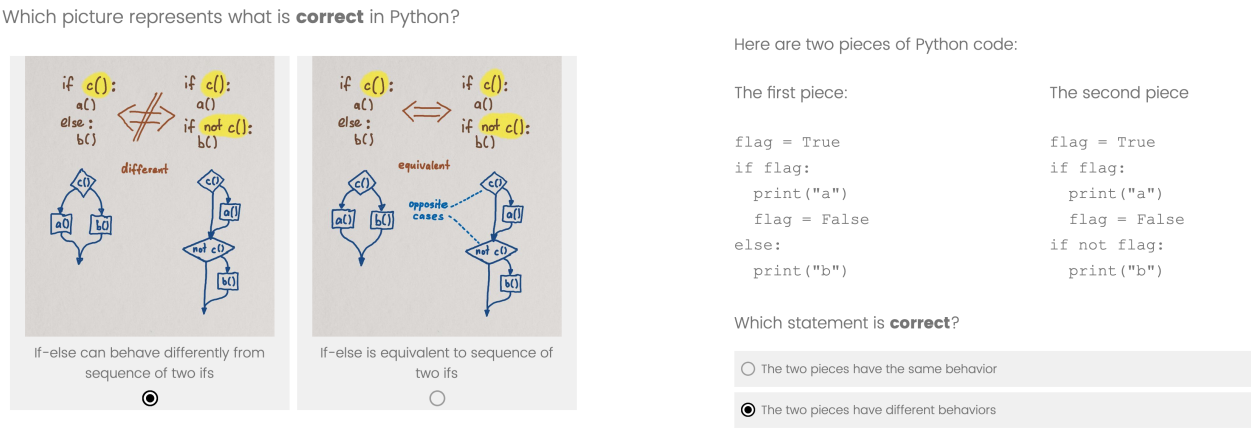
The first part of the survey required teachers to provide textual explanations, to increase the validity of the survey beyond what multiple-choice questions alone would guarantee. These explanations contain a rich source of claims from the teachers: some state the correct conception, while others express an incorrect fact (i.e., the misconception). Explanations offer insights into the teachers' thinking processes and complement our results by shedding light on RQ1 beyond the numbers reported in Section 4.2. Occasionally, teachers also added comments to their explanations about the prevalence of a misconception among their students or about the strategies they use to deal with it (RQ2 and RQ4).

We discuss each of the 16 programming misconceptions we studied, starting from the most commonly held (the prevalence of each misconception is repeated in parentheses). Each misconception is briefly summarized, where possible using the teachers' own words,



(a) "Illustrated refutation text" type. **(b) "Code Example" type.**

Figure 5: The two questions targeting the NoAtomicExpression misconception.



(a) "Illustrated refutation text" type. **(b) "Code Example" type.**

Figure 6: The two questions targeting the ConditionalIsSequence misconception.

and enriched with concrete incorrect explanations that refer to the specific questions in our study (the full questions are available in the online appendix [7]). These quotes are not meant to serve as quantitative evidence of certain phenomena, but rather offer a glimpse into the teachers' rich reasoning to answer the questions.

5.2.1 NoAtomicExpression (40 %). The correct claim is that, using the words of a teacher who answered correctly, *"an expression is any piece of code that returns a value"*. Pragmatically, another teacher observes: *"I can evaluate the value of* 123*, it is* 123*. Therefore it's an expression."*

Some teachers questioned the exact definition of expression. There is also controversy on the fact that a fragment of code may play multiple roles (cf. Figure 5b): "123 *is an argument,* 1+5 *is an expression"*, "123 *is only a value"*.

Expressions are a fundamental concept in programming languages such as Python [11]. One can start from atomic expressions and use them as building blocks to compose larger expressions, which can then be composed further. This compositional nature of expressions allows programmers to do abstract reasoning and avoid memorizing lots of special-purpose rules. Despite this, some teachers report not covering them: *"I've never used the term in a school context"*, *"I have not covered the concept of expression in my own teaching and have not used it anymore since the training course"*.

5.2.2 ReturnUnwindsMultipleFrames (38 %). To properly explain what happens when a function returns, a number of teachers referred to the call stack, demonstrating a proper understanding: *"When a return statement is executed in a function, it completes that function's execution and returns control to the point where the function was called. This action pops only the current function's call stack frame, meaning it exits the current function but does not directly affect any other frames further down the stack."*, *"A return statement in Python only pops the current function's call stack frame, returning control to the immediate caller, not multiple frames"*, *"Functions*

on call stack must be removed function by function", *"Because the functions lie on an 'execution stack'"*.

Many teachers claimed to be uncertain. Some confused the action of returning, when the callee terminates and gives back the control to the caller, with the return of a value, and got puzzled by the presence of a return statement without any expression or the lack of a return statement altogether. (In Python, in both cases the value actually returned is `None`.) *"*`b` *doesn't return to* `a`*, it gives back nothing"*, *"*`a` *has no return-statement"*, *"only* `b` *returns"*.

A couple of teachers noted that the answer would be different if Python had support for *"tail-call optimization"*.

5.2.3 ConditionalIsSequence (35 %). A sequence of two `if` statements, where the second condition is the negation of the first, is not necessarily equivalent to an `if/else` statement, due to potential side effects.

Teachers report that this misconception commonly leads to bugs for students: *"Indeed, in the second code,* `flag` *is set to* `False` *before the second* `if` *statement, which will as a result be validated. That is a common mistake my students do"*, *"Pupils often state the 2nd statement wrongly so I encourage them to always use if-else"*.

Some teachers pointed out that in one of the two questions (Figure 6a) `c()` must return a boolean value for the question to be well defined. This is a reasonable assumption, but Python actually permits arbitrary values in conditions, which are interpreted according to their truthiness.

5.2.4 NoShortCircuit (31 %). Python does not always evaluate both operands in expressions with `and`/`or` operators. Several teachers are aware of this "lazy" behavior, mentioning efficiency as the reason for short-circuiting: *"it's more efficient, there is no need to check* `b` *if* `a` *is already false"*, *"avoiding unnecessary operations and thus working more efficiently"*, *"Python is designed to take advantage of the logic in order to improve the efficiency of execution time"*, *"running the second test is not needed and not running it would allow a programmer to optimize away some expensive tests by ordering things the right way"*. We did not find explanations mentioning the possibility of exploiting this behavior to guard the execution of a method and call it only when a variable does not refer to `None` (e.g., the idiom `o and o.m()`).

Some teachers know Java and express doubts about whether Python's `and` corresponds to Java's `&` or `&&`: *"Here I am not sure whether* `and` *behaves like* `&&` *in Java"*, *"I don't know whether in Python there is a difference between* `&` *and* `&&`*"*.

5.2.5 MultipleValuesReturn (30 %). Python's lightweight syntax for creating a tuple (e.g., `return a, b`) induces some teachers to claim that functions can actually return multiple values: *"two variables are given in the return statement"*, *"functions can return many values separated by commas"*. The language and common idioms may deepen this misunderstanding, as one can use the unpack syntax to assign two values to two variables, making it look like two values are actually returned. Indeed, a teacher observed: "`get_names` *returns 2 values (here strings) that will be stored in two different variables. For example here we should call this function as:* `firstname, lastname = get_names(thisperson)`*"*.

Several teachers explained that values are packed into a list, although Python actually creates an immutable tuple. The flexibility can admittedly be confusing: *"strange 'inconsistency' in Python [...] that* `return (a)` *does not return a tuple,* `return [a]` *returns a list, but* `return a, b` *produces a tuple"*. Creating a single-element tuple in Python requires a trailing comma, with or without parentheses, as in `return (a,)`.

5.2.6 AssignmentCopiesObject (27 %). Several teachers explained that the correct behavior, i.e., objects are not copied but aliased with a simple assignment, is due to "pass by reference". Lu and Krishnamurthi [24] argue that while the terminology "call-by-reference" is widespread, it would be more appropriate to use "bind-by-reference", as the semantics not only affects calls but also definitions and copies of structured data. Teachers likely heard, understood and memorized the terminology "pass by reference" during their training and are now using it to explain these related behaviors, even when, like in our question with a simple assignment, there is no call and therefore nothing is "passed". This kind of explanation was commonplace: *"pass by reference:* `a` *and* `b` *point to the same place in memory so if you modify one of the two you modify both"*, *"Using* `=` *to assign to* `b` *the values of* `a`*, you pass everything by reference, so the changes on* `b` *are also valid on* `a`*"*, *"I think it is call by reference"*, *"lists are pass by reference, not by value"*.

Students struggle with problems caused by this misconception: *"To me a questionable design choice in Python causing problems for students"*, *"This misconception sometimes causes problems for students who are used to have copied values"*.

5.2.7 MapToBooleanWithIf (26 %). Some teachers are convinced that an `if` statement is necessary to produce a boolean value, even when one already has a boolean expression: *"Without if-statement there would be no check"*, *"a value has to be returned, not a boolean comparison"*, *"you can't return a relational comparison which then makes no sense"*, *"for a* `True` *or* `False`*, I need an if"*. This belief may stem from teachers having hardly ever seen an example where a return statement includes a relational expression, and may have internalized the `if` statement pattern as a necessary structure.

5.2.8 ComparisonWithBoolLiteral (22 %). The questions for this misconception featured an `if` statement with either `stop` or `stop == True` as a condition. Some teachers' explanations featured the very definition of this misconception, which claims that a variable alone is not enough to be a condition: "`stop` *is a variable here whose value must be checked"*, "`stop` *needs a comparison with something, otherwise the expression doesn't make sense"*.

Many teachers noted that the two expressions are equivalent only if `stop` is a boolean: *"if* `stop` *is a boolean, then the value is* `True` *or* `False`*, we don't need to test it"*. This is a reasonable assumption, but it is not strictly correct in Python (e.g., `1` is a value considered to be true, and the expression `1 == True` is also true).

5.2.9 ReturnCall (20 %). A return statement is not a function call and therefore does not require parentheses. Most teachers are well aware of this: *"return is not a function"*, *"it is not a function but a reserved word"*, *"parentheses are not needed"*. A teacher pointed out that the evolution of the Python language has been confusing on the distinction between statements and function calls: *"I think this has changed from Python 2 to 3. Similar to* `print 5 + x`*"*.

Another teacher incorrectly noted in their explanation for this misconception that *"return needs a value not an expression to return"*,

which may be a symptom of the NoAtomicExpression misconception (Section 5.2.1).

5.2.10 ParenthesesOnlyIfArgument (15 %). Parentheses are necessary even for function calls without arguments. Some teachers mentioned in their explanations that they and their students frequently encounter a related mistake, when they do not add parentheses: *"we print a representation of the function itself, which includes the position in memory (frequent mistake among students)"*, *"mistake made multiple times [at the university during the training program]"*. Others are aware that without parentheses the function is not called, but are unsure about what happens: *"I thought* `print(f)` *yields error message, but certainly not* `42`*"*.

5.2.11 NoReservedWords (14 %). Some words are reserved by the language and cannot be used as identifiers. Most teachers answered correctly, but some explanations showed uncertainty on the extent of the reserved words. For example, a teacher notes that *"at least for the word 'sum' I know that it works"*, likely observing that it is possible to override the meaning of certain words. In Python, `sum` is the identifier of a built-in function, which indeed can be reassigned. The general flexibility of the programming language may also induce someone to think that certain operations may be possible: *"I am not sure about that... Python is so tolerant sometimes..."*.

5.2.12 OutsideInFunctionNesting (11 %). Nested function calls are not invoked outside-in, but inside-out. A handful of teachers described strategies to remind themselves and their students of the correct order, relating the evaluation of nested Python function calls to mathematics: *"This is the same as with nested functions in mathematics"*, *"Like 'chaining' in math, or formulas in Excel"*, *"Like in mathematics, working inside out"*, *"Otherwise it would contradict mathematics"*.

5.2.13 DeferredReturn (9 %). As many teachers note, a return statement *"immediately terminates the function"*, regardless of its position, and thus any other subsequent statement in the body of the function is not evaluated. The question with the illustrated refutation text contained a sequence diagram, which is already an explanation of sorts.

Some teachers clearly showed the misconception in their explanation, claiming that the `print` function invocation that in our question comes after the `return` statement is still executed, before the function actually returns: *"*`print("Done!")` *is executed first, then the return value of the method is printed."*

A couple of teachers pointed out that the code we provided, which contained another statement after the return statement, would not be accepted by a certain educational IDE: *"the print-statement is not reachable, with e.g., TigerJython would even produce an error message 'return returns to the call site'"*.

5.2.14 VariablesHoldExpressions (8 %). When assigning to a variable, the expression on the right-hand side of the `=` operator *"[...] is evaluated and the resulting value assigned to a variable"*, as explained by a teacher. The variable holds a reference to a value, not to an expression that needs to be evaluated: *"[Assignments] do not work like functions."*

The illustrated refutation questions for this misconception (as well as the ones for AssignCompares) included a memory diagram depicting numbers as objects on the heap. Some teachers incorrectly claimed that the diagram was wrong: *"*`v` *is an integer, not an object"*, *"numbers are primitive data-types and are stored by value"*, *"*`v=8` *does not change* `a`*. Assignment concerning numbers uses copy by value"*, *"a simple integer isn't an object"*. Most probably, teachers were transferring knowledge from other programming languages where numerical values are not stored as objects on the heap. Given our context, we expect Java to be the main source of these explanations. (The choice of including a memory diagram may thus have misled some teachers, who answered incorrectly because they misunderstood the question. The analysis of the explanations helped to rectify some of these cases.)

A couple of teachers correctly observed that both diagrams, even the one in the correct answer, were slightly incorrect, as they showed `2` instead of `2.0` as a result of the division: *"*`v/2` *evaluates to the float* `2.0`*, not the int* `2`*. For example,* `range(v/2)` *would raise a* `TypeError`*. This is something students struggle with [...]"*.

5.2.15 IfIsLoop (4 %). The correct claim is that *"*`if` *is executed only once"*. Almost all teachers correctly answered both questions targeting this misconception. Interestingly, however, two teachers verbalized the exact misconception, mistaking an `if` statement for a `while` statement: *"The variable* `a` *is reduced by one while it's greater than 0. Thus* `a` *decreases 3, 2, 1, 0 and when it reaches 0 the loop finishes and this last value is then printed"*, *"The variable* `a` *is decremented until* `a` *reaches 0"*.

5.2.16 AssignCompares (3 %). Python, like many other programming languages, uses `=` to denote the assignment of values to variables and the `==` binary operator for equality. This misconception, believing that `=` actually compares for equality, is widely known [10, 19]. Teachers are well aware of it and of the problems it causes: *"this is a common source of errors"* and even suggest a possible remedy: *"In my opinion, this is a design flaw in many programming languages because the equal sign does not signal the inherent right-to-left direction of the operation. I'd rather use <- for assignment"*.

5.3 Years of Experience Are Not Correlated With Better Performance

Figure 7 compares the years of teaching experience with the number of correct answers by a teacher. Computing the correlation using Pearson's r reveals that the number of years spent teaching informatics is only very weakly correlated with the number of correct answers ($r = 0.09$, $p = 0.37$); such correlation is insignificant. The correlation completely disappears when we consider the years of teaching experience in general: $r = -0.01$, $p = 0.90$.

Contrary to what one may naïvely expect, these results show that years of teaching experience may not accurately predict the presence of programming misconceptions in a teacher. As a consequence, carrying out an assessment is necessary when one intends to get a more reliable measure of the teachers' knowledge.

6 Threats To Validity and Limitations

External validity. We only surveyed teachers in one country, at one educational level, and focused only on one programming language. Our results might not generalize to other countries, other

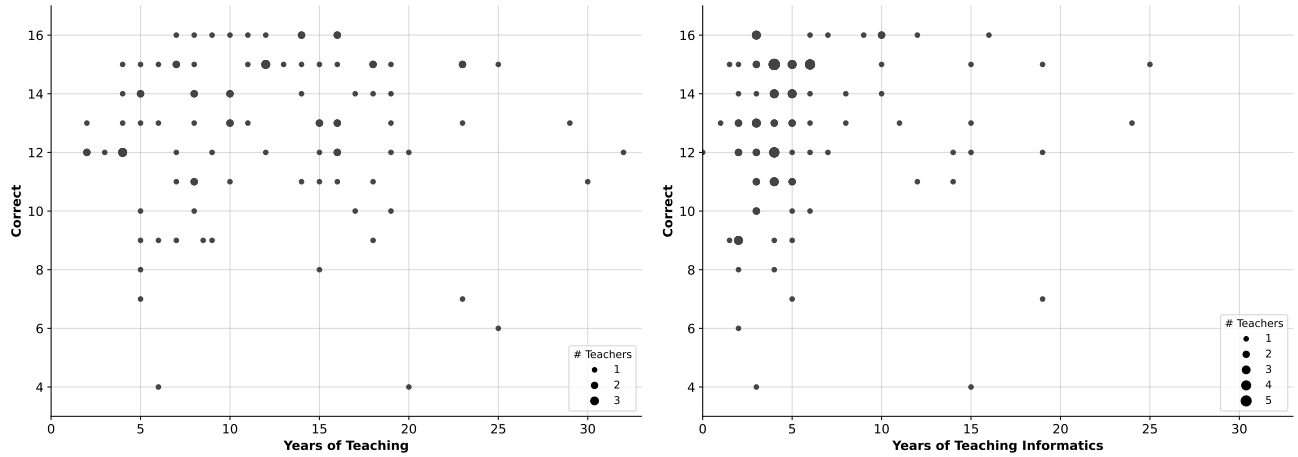


Figure 7: Number of correct answers for the 16 misconceptions given by the 97 teachers, according to the number of years of teaching experience overall (left) and teaching experience in informatics (right).

levels, and other programming languages. However, the set of teachers consisted of 3 groups speaking 3 different natural languages, some of whom went through related but different retraining programs (different instructors, different programming curricula).

We only studied 16 misconceptions. We carefully picked that set: they were all previously reported and unambiguously documented in the progmiscon.org inventory. They all focused on Python, and specifically on concepts the teachers covered in their courses. While there certainly are other misconceptions we did not include in our study, we did include every Python misconception reported on the inventory related to the concepts teachers cover in their courses.

Internal Validity. The teachers completed the survey on their own and could theoretically have cheated in the assessment part. We explicitly asked them to answer honestly and without any help, and the survey was anonymous, thus their incentive to cheat was minimal.

The survey was long, possibly leading to a decrease of attention over time. We randomized the presentation of the misconceptions in both parts, which meant that each misconception had an equal chance to appear early in the process.

To increase the validity of an otherwise multiple-choice-only instrument, we augmented the multiple-choice questions with mandatory textual explanations. Each explanation was manually assessed by a single rater. Despite agreeing beforehand on the criteria to assess the explanations, we did not have multiple raters assess the same explanation. While this would have increased the validity of the study even more, subsequent checks provide diminishing returns, and the explanations are already a check of the multiple-choice answers themselves.

Teachers may have provided random answers to claim the reward with minimal effort. Given that we personally knew most of the invited teachers, chances that they clicked through the survey just to claim their reward are small. The manually reviewed explanations confirm that all teachers wrote reasonable answers throughout the assessment part of the survey. The average time spent on the survey was considerably longer than expected (cf. Section 4).

Teachers might have overstated the importance they attribute to the misconceptions, because they might have believed we were interested in seeing a high importance. We believe this is possible, but the relative importance between the misconceptions would equally be affected by this factor. Our results focus on the relative importance.

7 Conclusions

We conducted an extensive survey about programming misconceptions with teachers at the upper secondary level in Switzerland. On a set of assessment items focusing on 16 Python misconceptions, the majority of teachers answered correctly, but we found that between 3 % and 40 % gave incorrect answers, depending on the misconception. Overall, teachers were familiar with the misconceptions, considered them rather important and reported having seen them in their students.

This study contributes to the body of literature that assesses the prevalence of programming misconceptions. Similar to prior research, it surveys teachers about the prevalence and importance of misconceptions they encounter in their classrooms, and about their strategies to detect and fix them. However, unlike prior research, it also determines whether the teachers themselves hold these misconceptions. The study shows that focusing on responses by teachers who do *not* hold the corresponding misconception has a significant effect on the results. Teachers who did not hold a misconception considered it more prevalent and more important.

Commenting at the end of the survey, teachers felt that taking the survey helped them reflect on their own conceptions (*"always valuable to reflect one's own conceptions. :)"*) and their own teaching (*"Some of the possible misconceptions throw an interesting light on the difficulties encountered by students."*, *"I realize that there is a lot to improve my teaching!"*). We hope that this study helps other educators to recognize and address the discussed misconceptions, and that it provides a solid foundation for future research.

Acknowledgments

We are immensely grateful to all the teachers who participated in this study. This research would not have been possible without them. Their dedication to great teaching shines through in many of their comments, and we feel lucky to have learned from their experience and insight.

We thank the Swiss Association for Informatics in Education (SVIA-SSIE-SSII) for the help in recruiting teachers for this study.

This work was partially funded by the Swiss National Science Foundation project 200021_184689.

References

[1] Andrea Adamoli. 2023. An Agile Concept Inventory Methodology to Detect Large Sets of Student Misconceptions in Programming Language Courses. In *Responsive and Sustainable Educational Futures*, Olga Viberg, Ioana Jivet, Pedro J. Muñoz-Merino, Maria Perifanou, and Tina Papathoma (Eds.). Springer Nature Switzerland, Cham, 1–15. https://doi.org/10.1007/978-3-031-42682-7_1

[2] I. Elaine Allen and Christopher A. Seaman. 2007. Likert Scales and Data Analyses. *Quality progress* 40, 7 (2007), 64–65.

[3] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative Research in Psychology* 3, 2 (Jan. 2006), 77–101. https://doi.org/10.1191/1478088706qp063oa

[4] Neil C. C. Brown and Amjad Altadmri. 2017. Novice Java Programming Mistakes: Large-Scale Data vs. Educator Beliefs. *ACM Trans. Comput. Educ.* 17, 2 (May 2017), 7:1–7:21. https://doi.org/10.1145/2994154

[5] Neil C. C. Brown, Sue Sentance, Tom Crick, and Simon Humphreys. 2014. Restart: The Resurgence of Computer Science in UK Schools. *ACM Transactions on Computing Education* 14, 2 (June 2014), 1–22. https://doi.org/10.1145/2602484

[6] Sue Brown and Judy Bergman. 2013. Preservice Teachers' Understanding of Variable. *Investigations in Mathematics Learning* 6, 1 (2013), 1–17.

[7] Luca Chiodini, Joey Bevilacqua, and Matthias Hauswirth. 2025. Supplementary Material for the Paper "Surveying Upper-Secondary Teachers on Programming Misconceptions". https://doi.org/10.5281/zenodo.15065896

[8] Luca Chiodini and Matthias Hauswirth. 2021. Wrong Answers for Wrong Reasons: The Risks of Ad Hoc Instruments. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research (Koli Calling '21)*. ACM, New York, NY, USA, 1–11. https://doi.org/10.1145/3488042.3488045

[9] Luca Chiodini, Matthias Hauswirth, and Andrea Gallidabino. 2021. Conceptual Checks for Programming Teachers. In *Technology-Enhanced Learning for a Free, Safe, and Sustainable World*, Tinne De Laet, Roland Klemke, Carlos Alario-Hoyos, Isabel Hilliger, and Alejandro Ortega-Arranz (Eds.). Vol. 12884. Springer International Publishing, Cham, 399–403. https://doi.org/10.1007/978-3-030-86436-1_43

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

[11] Luca Chiodini, Igor Moreno Santos, and Matthias Hauswirth. 2022. Expressions in Java: Essential, Prevalent, Neglected?. In *Proceedings of the 2022 ACM SIGPLAN International Symposium on SPLASH-E (SPLASH-E 2022)*. ACM, New York, NY, USA, 41–51. https://doi.org/10.1145/3563767.3568131

[12] Norman Cliff. 1993. Dominance Statistics: Ordinal Analyses to Answer Ordinal Questions. *Psychological Bulletin* 114, 3 (1993), 494–509. https://doi.org/10.1037/0033-2909.114.3.494

[13] Jan Cuny. 2012. Transforming High School Computing: A Call to Action. *ACM Inroads* 3, 2 (June 2012), 32–36. https://doi.org/10.1145/2189835.2189848

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

[15] Rodrigo Duran, Juha Sorva, and Otto Seppälä. 2021. Rules of Program Behavior. *ACM Transactions on Computing Education* 21, 4 (Nov. 2021), 33:1–33:37. https://doi.org/10.1145/3469128

[16] Ruhama Even. 1993. Subject-Matter Knowledge and Pedagogical Content Knowledge: Prospective Secondary Teachers and the Function Concept. *Journal for Research in Mathematics Education* 24, 2 (1993), 94–116. https://doi.org/10.2307/749215 jstor:749215

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

[18] Catherine O. Fritz, Peter E. Morris, and Jennifer J. Richler. 2012. Effect Size Estimates: Current Use, Calculations, and Interpretation. *Journal of experimental psychology: General* 141, 1 (2012), 2.

[19] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. Identifying and Correcting Java Programming Errors for Introductory Computer Science Students. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03)*. Association for Computing Machinery, New York, NY, USA, 153–156. https://doi.org/10.1145/611892.611956

[20] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. 2010. Identifying Student Misconceptions of Programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education - SIGCSE '10*. ACM Press, Milwaukee, Wisconsin, USA, 107. https://doi.org/10.1145/1734263.1734299

[21] Cazembe Kennedy and Eileen T. Kraemer. 2018. What Are They Thinking?: Eliciting Student Reasoning About Troublesome Concepts in Introductory Computer Science. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. ACM, Koli Finland, 1–10. https://doi.org/10.1145/3279720.3279728

[22] Michelle E. Kiger and Lara Varpio. 2020. Thematic Analysis of Qualitative Data: AMEE Guide No. 131. *Medical Teacher* 42, 8 (Aug. 2020), 846–854. https://doi.org/10.1080/0142159X.2020.1755030

[23] Colleen M. Lewis, Michael J. Clancy, and Jan Vahrenhold. 2019. Student Knowledge and Misconceptions. In *The Cambridge Handbook of Computing Education Research* (1 ed.), Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, Cambridge United Kingdom, 773–800. https://doi.org/10.1017/9781108654555.028

[24] Kuang-Chen Lu and Shriram Krishnamurthi. 2024. Identifying and Correcting Programming Language Behavior Misconceptions. *Proc. ACM Program. Lang.* 8, OOPSLA1 (April 2024), 106:334–106:361. https://doi.org/10.1145/3649823

[25] George J. Posner, Kenneth A. Strike, Peter W. Hewson, and William A. Gertzog. 1982. Accommodation of a Scientific Conception: Toward a Theory of Conceptual Change. *Science Education* 66, 2 (April 1982), 211–227. https://doi.org/10.1002/sce.3730660207

[26] Yizhou Qian, Susanne Hambrusch, Aman Yadav, Sarah Gretter, and Yue Li. 2020. Teachers' Perceptions of Student Misconceptions in Introductory Programming. *Journal of Educational Computing Research* 58, 2 (April 2020), 364–397. https://doi.org/10.1177/0735633119845413

[27] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education* 18, 1 (Oct. 2017), 1–24. https://doi.org/10.1145/3077618

[28] Lee S. Shulman. 1986. Those Who Understand: Knowledge Growth in Teaching. *Educational Researcher* 15, 2 (Feb. 1986), 4–14. https://doi.org/10.3102/0013189X015002004

[29] Teemu Sirkiä and Juha Sorva. 2012. Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research - Koli Calling '12*. ACM Press, Koli, Finland, 19–28. https://doi.org/10.1145/2401796.2401799

[30] Christine D Tippett. 2010. Refutation Text in Science Education: A Review of Two Decades of Research. *International Journal of Science and Mathematics Education* 8, 6 (Dec. 2010), 951–970. https://doi.org/10.1007/s10763-010-9203-x

[31] Aman Yadav and Marc Berges. 2019. Computer Science Pedagogical Content Knowledge: Characterizing Teacher Performance. *ACM Transactions on Computing Education* 19, 3 (Sept. 2019), 1–24. https://doi.org/10.1145/3303770

[32] Aman Yadav, Sarah Gretter, Susanne Hambrusch, and Phil Sands. 2016. Expanding Computer Science Education in Schools: Understanding Teacher Experiences and Challenges. *Computer Science Education* 26, 4 (Dec. 2016), 235–254. https://doi.org/10.1080/08993408.2016.1257418