A Curated Inventory of Programming Language Misconceptions

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

Igor Moreno Santos
igor.moreno.santos@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Andrea Gallidabino
andrea.gallidabino@usi.ch
Software Institute, Università della Svizzera italiana
Lugano, Switzerland

Anya Tafliovich
anya@cs.utoronto.ca
University of Toronto
Scarborough, Canada

André L. Santos
andre.santos@iscte-iul.pt
Instituto Universitário de Lisboa (ISCTE–IUL)
Lisbon, Portugal

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

Abstract

Knowledge about misconceptions is an important element of pedagogical content knowledge. The computing education research community collected a large body of research on misconceptions, using a diverse set of definitions and approaches. Inspired by this prior work, we present an actionable definition of misconceptions, focused on the area most commonly studied: programming and programming languages. We then introduce an organizational structure for collections of programming language misconceptions. We study how existing collections fit our organization, and we present a curated inventory of programming language misconceptions that aims to follow our definition and structure. Our inventory goes beyond traditional programming misconception collections. It connects misconceptions to the authoritative specifications of languages, to places they may be triggered in textbooks, to research papers that discuss them, and it provides support for integrating programming language misconceptions into educational platforms.

CCS Concepts

• **Social and professional topics** → **Computer science education**; • **Software and its engineering** → *General programming languages*.

Keywords

programming; programming languages; misconceptions; pedagogy

ACM Reference Format:
Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, and Matthias Hauswirth. 2021. A Curated Inventory of Programming Language Misconceptions. In *26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2021), June 26–July 1, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3430665.3456343

ITiCSE 2021, June 26–July 1, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8214-4/21/06…$15.00
https://doi.org/10.1145/3430665.3456343

1 Introduction

Knowledge about misconceptions is a key element of pedagogical content knowledge [31] that affects student learning[1] [29]. "Computing educators are often baffled by the misconceptions that their CS1 students hold. We need to understand these misconceptions more clearly in order to help students form correct conceptions." These are the opening sentences from the top-ranked paper receiving the SIGCSE "Top Ten Symposium Papers of All Time Award"[2], Kaczmarczyk et al.'s "Identifying student misconceptions of programming" [16].

However, much of the work on programming misconceptions is "locked away" in academic papers targeted at researchers and "the dissemination of approaches and tools has been limited" [27]. Given the agreement on the importance and the positive effect teacher knowledge about misconceptions can have on teaching, how then can educators easily find information about misconceptions related to the specific material they teach?

This paper solves that problem for a focused subset of misconceptions, which we define as *programming language misconceptions* (Section 2). It bridges the gap between (1) misconceptions research, (2) programming language theory, and (3) educational practice, with the goal of *making misconceptions easily accessible to educators.* The paper introduces a form and structure for presenting misconceptions (Section 3), presents progmiscon.org, a curated inventory of 198 misconceptions following that organization (Section 4), and evaluates the definition and the inventory by comparing it to two well-known existing collections of programming misconceptions (Section 5) by Sorva [37] and by Lewis [19].

2 Programming Language Misconceptions

There is an extensive body of research on misconceptions of novice programmers. However, few papers define explicitly what they mean by *misconception*. When studies use the word *misconception* without an explicit definition (e.g., [4, 13, 14, 28]), they may implicitly assume the meaning from science education, where misconceptions are usually defined as "flawed ideas held by students, often

[1] Sadler and Sonnert [29] found that "teachers who could identify these misconceptions had larger classroom achievement gains, much larger than if teachers knew only the correct answers".
[2] https://www.acm.org/media-center/2019/march/sigcse-top-10-papers

strongly, which conflict with commonly accepted scientific consensus" [27, 34]. Some works instead contain an explicit definition. Smith et al. [34], analyzing research in science and mathematics education, characterize misconceptions as "student conceptions that produce a systematic pattern of errors". In a context closer to ours, Sorva [38] defines them as "understandings that are deficient or inadequate for many practical programming contexts". In an attempt to further delimit the space, Qian and Lehman [27] add that "misconceptions, per se, are probably best defined as errors in conceptual understanding". Despite progressive refinements, these definitions are still relatively broad.

In the context of *programming education*, there is an opportunity for a more focused and actionable definition. In his seminal paper, Du Boulay [7] groups the difficulties novices have when learning to program into five areas, which could be summarized as *orientation* (general problems of orientation in understanding what programming means and what one can achieve with it); *semantics* (difficulties with understanding "the properties of the machine one is learning to control, the notional machine"); *syntax* ("problems associated with the notation of the various formal languages that have to be learned"); *strategies* (difficulties of acquiring standard structures or plans used to achieve small-scale goals); and *pragmatics* ("how to specify, develop, test, and debug a program"). Subsequent work [23, 27] followed a similar but slightly more focused partitioning of the space of *programming* knowledge: syntactic knowledge, conceptual (semantic) knowledge, and strategic knowledge. In this paper we narrow the focus even further, by excluding strategic knowledge—knowledge about the programming *process*. We focus on syntactic and semantic knowledge—the kind of knowledge captured in the specification of a programming language (PL). We call the corresponding misconceptions *programming language* misconceptions:

Definition 2.1. A programming language misconception is a statement that can be disproved by reasoning entirely based on the syntax and/or semantics of a programming language.

Our definition is centered around *statements*. Such statements are related to ideas from the fragmented side of conceptual change research: *p-prims* in knowledge-in-pieces [5] (e.g., in physics, "increased effort begets greater results") and the idea of *facets* [21, 24] (e.g., again in physics, "horizontal movement makes a falling object fall more slowly"). Like facets, our statements "describe students' thinking as it is seen or heard in the classroom", and they are "individual pieces or constructions of a few pieces of knowledge" [24].

2.1 An Actionable Definition

A definition is actionable if it has practical value. So what is the practical value of the above definition? First, our definition allows us to decide with some certainty whether something is or is not a programming language misconception. This means that it enables researchers and educators to categorize student difficulties as programming language misconceptions.

What makes our definition actionable? PLs are explicitly and completely specified either in the form of a language specification, which can be formal or not, or in the form of their implementation. These specifications represent an unambiguous "ground truth". Thus, programming language misconceptions have a big advantage: they allow us to check the correctness of a *conception* by referring to the specification of the corresponding PL.

For example, according to our definition, the following statements are **not** programming language misconceptions:

- "JavaScript is better than Java"—does not involve language syntax or semantics.
- "Garbage collection slows down program execution"—cannot be disproved solely based on the language syntax and semantics.
- "Assembly programs are longer than equivalent high-level programs"—cannot be disproved solely based on the language syntax and semantics.

On the other hand, the following statements **are** programming language misconceptions:

- "In Java, an object cannot access private members of any other object." (PrivateFromOtherInstance).
- "In JavaScript, objects are assigned by value." (AssignmentCopiesObject).
- "In Python, expressions must consist of more than one piece" (NoAtomicExpression).

2.2 Programming Language Dependence

As the examples in the previous section show, a programming language misconception is related to a specific PL. Some misconceptions are very specific to a given language, or even a given version of the language. Other misconceptions exist across multiple language versions (e.g., Java before and after version 8), multiple related languages (e.g., C and C++), or even across languages from different paradigms (e.g., functional and imperative).

If misconceptions about syntax and semantics are *not* tied to PLs, one often cannot conclude that certain conceptions are wrong, even for relatively simple ones. Consider the statement: "The operator && returns the logical conjunction of its operands". This statement is correct in many PLs (among which Java and JavaScript), but it is wrong in Python (the *syntax* is invalid). Even if one targets a common syntax, the *semantics* of different languages can lead to different results. For example, in Python the expression `op1 == op2 == op3` is equivalent to the semantics students are used to from math: it means `(op1 == op2) AND (op2 == op3)`. In many other PLs, including Java and JavaScript, that expression has an entirely different semantics: it evaluates as `(op1 == op2) == op3`.

3 Organizing a Collection of Misconceptions

An actionable definition allows us to catalog misconceptions. However, what information about programming language misconceptions should our inventory contain to support its purpose *to turn misconceptions into learning opportunities*? We split this discussion into aspects of form (of an individual misconception description) and structure (of the overall collection).

3.1 Form of a Misconception

The core of our misconception descriptions is based on the idea of a *refutation text*, an approach to induce conceptual change with a decades long history in science education [41]. It deliberately juxtaposes the *incorrect* conception (the misconception statement,

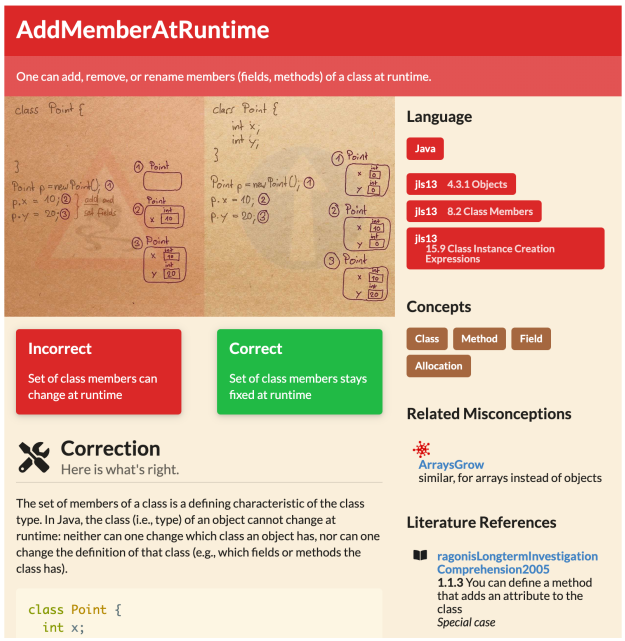

Figure 1: Excerpt of a Misconception Page

in red at the top in an extended form and also in red below the image) with an explanation of the *correct* conception (in green below the image). We present each misconception on its own web page (see Figure 1) and surround the refutation text core with pedagogically relevant information and information that connects the misconception to coherent larger structures:

- *Name* - Inspired by design patterns, we give each misconception a unique, concise, meaningful, and memorable name (e.g., AddMemberAtRuntime, FunctionsMustBeNamed).
- *Statement* - Each misconception is represented as a short statement. This is the statement that can be disproved by reasoning based on the syntax and/or semantics of the related PL as mentioned in Definition 2.1.
- *Description* - The description elaborates on the statement of a misconception with a longer paragraph that may include short inline snippets of code or examples. Instructors can use this text to better understand the scope of a misconception. Both the shorter *statement* and the longer *description* represent the wrong conception: they violate in some way the associated PL's syntax and/or semantics.
- *Image* - The image captures the essential aspects of the misconception. It visually juxtaposes the correct and incorrect conceptions. The image is a visual equivalent to a "refutation text" [41] and is somewhat similar to a "concept cartoon" [18], two ideas from science education that have been used to induce conceptual change.
- *Correction* - For each misconception we provide a description of the corresponding correct conception. Corrections can include textual explanations, diagrams and notional machine representations, or *virtual demonstrations*, which teachers can re-use and present in their lessons. The virtual demonstrations are examples of executable snippets of code which allow the reader to test the incorrect and correct conceptions on the spot.
- *Symptoms* - Misconceptions include a description of the symptoms one can expect students holding that misconception to exhibit (e.g., code smells or erroneous snippets of code observed in students' solutions, which can point to the misconception). This helps teachers to identify more easily whether their students might have this misconception.
- *Observation Videos* - In addition to textual descriptions of symptoms, a small number of misconceptions also include videos of student performances that exhibit that misconception. These videos were collected in an observational study that investigates conceptual change in learning to program.
- *Origins* - Where available, we describe where a given misconception might have originated from. When we find a given educational intervention, a specific description, example, or diagram that is useful to instill a certain misconception, we capture that information. Being aware of possible origins can help teachers to address potential misconceptions right at the source.
- *Value* - Misconceptions can be very valuable. They are more than just errors in conceptual understanding. Often students hold conceptions that may be incorrect in a given PL, but that would reasonably hold in a different (existing or potential) language. Moreover, sometimes a misconception involves elements that a teacher can productively use as seeds to grow additional conceptual understanding.
- *Specification Sections* - By definition each of our misconceptions is specific to a PL. Exploiting the availability of authoritative PL specifications which describe their syntax and semantics, we directly point to the pertinent sections. This provides the background and context that allows reasoning about the misconception.
- *Textbook Sections* - Misconceptions refer to specific sections of well-known programming textbooks, where students learning to program might first be exposed to them. This allows teachers who use a given textbook to easily predict which misconceptions they should expect when teaching a specific chapter of their book.
- *Related Misconceptions* - Misconceptions often are related. For example ArraysGrow is to arrays what AddMemberAtRuntime is to objects. Our inventory captures these relationships and makes it easy to navigate among them. These connections create a network of relationships between all the misconception of a PL, which enables the exploration of the conceptual space.
- *In Other PLs* - Each captured misconception is described for a specific PL. However, there are many similarities between PLs, and thus a given misconception may apply similarly to multiple languages (e.g., we captured ThisAssignable both in Java and JavaScript). To enable this transfer between languages, we use the same name for equivalent misconceptions in different languages, and we provide links to navigate between them.
- *Research Papers* - Where possible we back our misconceptions with references to the existing research literature, extending the breadth and depth of the available evidence.
- *Explanation Video* - A misconception can be accompanied by a short explanation video. Such videos provide a modality for learning about the misconception that can resonate with teachers or students used to finding information in online videos.

• *Concepts* - Each misconception is connected to a set of well-known programming language concepts. These are the concepts the misconception's statement relates to.

3.2 Structure of the Collection

Theories of conceptual change like knowledge-in-pieces [5] and facets [21] show that conceptions do not exist in isolation. Instead, they are related to each other in rich structures. progmiscon.org structures the space of misconceptions by connecting misconceptions by concept, by programming language, by language specification section, and by textbook section. Moreover, it directly connects misconceptions with commonalities, and it connects misconceptions to equivalent misconceptions in other PLs. The web site allows navigating these connections and enumerating misconceptions based on the above properties. Developers of educational resources can go further, and use our static API to retrieve an up-to-date machine-readable list of misconceptions.

4 Content

The content of our inventory comes from observing students, their programming activities, and the artifacts they produced, in different programming courses over more than a decade. An initial set of misconceptions was collected in an undergraduate object-oriented programming course over 15 years. More recently, the content has been augmented with information collected in a graduate-level advanced programming course (in Python, Java, and JavaScript) and with two micro-genetic studies by the first author, who conducted, video-taped, and analyzed 38 hours of mastery checks with student volunteers in two different courses, in Java and JavaScript.

All our collection efforts placed a strong emphasis on analyzing *activities* instead of *artifacts*. We identified misconceptions in mastery check sessions [12, 42], in oral exams, and in video observations of students in different programming courses.

We also placed emphasis on analyzing students' *statements* about programming, instead of focusing on students' *programs*. Specifically, we identified students' recall statements [11] in courses using flipped-classroom pedagogies, where students had to read textbook sections before class, and submitted written recall statements as evidence of their preparation.

progmiscon.org currently contains misconceptions in three different languages: Java, JavaScript, and Python. Its structure easily accommodates other languages. Before being published, misconceptions go through a *draft* state while we enrich them with the characteristics presented in Section 3.

5 Evaluation

Previous sections proposed a *definition* of programming language misconception, described a way to *organize* a collection of such misconceptions, and introduced progmiscon.org, an inventory based on the above definition and organization. We now compare our inventory to prior published collections both in terms of the definition as well as the organization.

Table 1: Applying our definition of programming language misconception to misconceptions from different sources.

			Not PL misconception								
Source	**Items**	**PL misconception**	Teaching suggestion	Not about programm.	About NM	Symptom	Difficulty	About a library	Strategic knowledge	Other	**Unknown**
Sorva	162	114	0	2	1	2	32	0	1	0	10
Lewis	126	26	36	4	0	14	32	9	1	3	1
(Our)	198	175	0	0	12	0	0	7	4	0	0

5.1 Evaluation of the Definition

Our definition of programming language misconceptions is strongly focused. If the definition rules out most of the captured misconceptions, then it might be of limited use. We thus now evaluate which fraction of collected misconceptions fit our definition, and we characterize those misconceptions that do *not* fit our definition.

For this purpose we analyzed, besides our own progmiscon.org inventory, the two largest collections of misconceptions we are aware of: (1) Sorva's catalog [37] of 162 misconceptions, which were collected in his large survey of published research papers on the topic, and (2) Lewis's CS teaching tips web site [19] which contains over 120 tips tagged as "content misconceptions".

The definition ties a misconception to a particular programming language; thus, even though a programming language was not always clearly stated for each misconception, we inferred the programming language of each misconception whenever possible.

Two authors classified independently each misconception determining whether it fulfills the definition or not and if it not, choosing 1 out of 8 predetermined reasons that explain why that was the case. The overall accuracy of the rating across the three sources and the ten categories was 81%, and the corresponding Cohen's kappa coefficient was $\kappa = 0.66$, indicating "substantial agreement"[3].

The two authors then discussed the differences and arrived at an agreed upon classification summarized in Table 1. Column *Items* shows the total number of misconceptions in each repository. Column *PL misconception* shows the number of misconceptions that fulfill our definition of programming language misconception. The columns under *Not PL misconception* show how many do not fulfill our definition, with each column underneath showing the reason for not fulfilling it. Column *Unknown* shows the number of misconceptions that were ambiguous or whose meaning we could not otherwise determine. The reasons for a misconception to not fulfill the definition were:

[3]Note that in this case, since most of the data has been classified as "PL misconception", it applies the remark by Kaijanaho: "If the coders are extremely (but not perfectly) accurate, but some values are much rarer than others, then the value of κ is low despite the coders' accuracy" [17].

- *Teaching suggestion* - Statements like "be explicit about what direction references point when teaching about objects and references", found in Lewis' list, constitute valuable teaching suggestions, but do not fulfill the requirements imposed by our definition.
- *Not about programming* - Some statements refer to activities around programming (e.g., "Students have difficulty understanding how to share App Inventor projects between different computers"), but are not about programming itself.
- *About notional machine* - Some misconceptions are about notional machines, as is the case with the Java misconception ZeroInEdges ("A control-flow graph node can have zero incoming edges"). These do not constitute a programming language misconception because they are not about a PL.
- *Difficulty* - Statements about students' difficulties, although valuable teaching information, are also not programming language misconceptions according to our definition. In Sorva's list an example is "Difficulties understanding the invocation of a method from another method".
- *Symptom* - Some statements describe something students *do* rather than a belief they have about the syntax or semantics of a language. They are not themselves misconceptions but may be symptoms of a misconception, which would require further investigation to determine. Sometimes these symptoms are errors the students make while programming. For example, an off-by-1 error, described in Lewis' list, could indicate that the student believes arrays are indexed starting from 1, which would be a programming language misconception, but the error itself is not.
- *About a library* - Our definition of programming language misconception restricts the scope to the syntax or semantics of a PL. This means that a wrong belief about the semantics of a *library* is out of scope. The statement "println(String) prints a newline character followed by the given String" (misconception PrintNewLineFirst), from progmiscon.org, is a statement about the semantics of a library, not the semantics of a programming language (Java).
- *Strategic knowledge* - Some statements refer not to knowledge about a programming language but to strategic knowledge about programming in general. For example, Sorva's "A set (such as "team" or "the species of birds") cannot be a class", which refers to an issue about program design.
- *Other* - This category includes statements that cannot otherwise be disproved by reasoning about syntax and semantics of a language. For example, "Students may write code in HTML and CSS that contains many errors yet still renders correctly, leaving them with faulty understandings of concepts and acceptable code", from Lewis' list, describes a problematic behavior of browsers that could be the origin of misconceptions for students.

Analyzing Sorva's list of misconceptions, we see that 70% of misconceptions fulfill our definition, demonstrating that our definition captures a significant fraction of the space of misconceptions documented by the community. Moreover, we see that most of the items that are not programming language misconceptions are actually statements of student difficulties. As the last row of Table 1 shows, our own progmiscon.org inventory also contained some misconceptions that violate our own definition. Thanks to this analysis, we can now improve our collection.

5.2 Evaluation of the Organization

Section 3 described the form and structure of progmiscon.org, that is, how our inventory *represents* programming language misconceptions. To evaluate that, we compared our form for a misconception with the one used in prior work. We considered all the sources referenced in Sorva's catalog of misconceptions (Appendix A of [37]), excluding the studies that describe misconceptions only tangentially because they have a different focus (e.g., mental models, designing programs) or are not published work in English. To also capture more recent publications, we added all the papers referenced in [27] in the two sections that describe difficulties in syntactic and conceptual knowledge, and we searched for papers targeting misconceptions published at two of the main Computer Science Education research conferences (ITiCSE and ICER), ending up with 24 references.

Table 2 summarises our attempt to compare such body of literature with our own inventory across every aspect described in Section 3, with the following modifications. We grouped under *Statement* (the central sentence that describes the misconception) *Description* and *Image*, which we use to enrich it. The column *Observations* is an abstraction which accommodates our observational videos with students and textual excerpts reported in prior work.

Table 2 is the result of two authors independently assessing the form of the misconceptions described in the references. While the agreement was substantial (79%[4]), three characteristics have been a source of differences worth discussing.

One rater considered *Research papers* to be partially present when, even without a misconception being directly connected to a reference, it is possible to establish the link between misconceptions and the related work. Those modifications have been integrated in the table.

Name also allows room for differences in interpretation, as authors have employed different techniques to give unique identifiers to misconceptions. Some of them assign incremental numeric values [10, 28], while others use a combination of letters and numbers to group them by topic [3, 16]. Most of the references, though, do not use meaningful and memorable names like the ones we propose.

The third source of disagreement has been on *Related misconceptions*. Some works indeed present misconceptions in groups (often divided on the basis of concepts, topics or themes) and one rater considered this to be a possible indirect way to link to related misconceptions. It is certainly true that being able to navigate misconceptions via concepts is a desirable feature, that is also offered by progmiscon.org, but we argue that direct links among misconceptions can be even more powerful, as they can establish different kinds of relationships.

Although the characteristics we include in progmiscon.org are not novel, this evaluation shows that they have been reported in a relatively sparse manner. Our inventory stands out for tying each misconception to the relevant section of the language specification of the corresponding programming language, and for being centered around the idea of a refutation text (both in textual and

[4]We considered a disagreement of one point when one rater marked "present" and the other one "absent", zero points when there was a match, and half point in the remaining cases. We excluded from the computation of the agreement five columns (Statement, Concepts, PL Specification, Textbooks, and Symptoms) in which the characteristic was marked in almost all or almost no papers.

Table 2: Comparison with the form of misconceptions in literature. ●: present (at least implicitly), ◐: partially present

Authors	Year	*Name*	*Statement*	*Correction*	*Concepts*	*PL Spec.*	*Papers*	*Textbooks*	*Rel. Miscon.*	*In other PLs*	*Symptoms*	*Observations*	*Origins*	*Value*
Bayman and Mayer [2]	1983	◐	●		◐						●			◐
du Boulay [7]	1986		●		●		◐			◐	●		●	●
Pea [25]	1986	◐	●	●	●		◐			●	●	●	●	●
Putnam et al. [26]	1986	●	●		●		◐				●	●		
Sleeman et al. [33]	1986	●	●		●		◐				●	◐		
Fleury [9]	1991	●	●		●		●				●			●
Holland et al. [13]	1997		●	◐	●					◐	●		◐	●
Madison and Gifford [22]	1997		●		●						●	●	●	◐
Fleury [10]	2000	●	●	●	●						●		◐	●
Hristova et al. [14]	2003		●		◐						●			●
Eckerdal and Thuné [8]	2005		●		●		◐			◐		●		●
Jackson et al. [15]	2005		●	●							●	◐		
Ragonis and Ben-Ari [28]	2005		●		●		◐				◐	◐		
Teif and Hazzan [40]	2006	●	●		●		◐			◐	●	●	●	
Doukakis et al. [6]	2007		●		●					◐	●		●	●
Ma [20]	2007	●	●	●	●						●	●		
Sorva [35]	2007		●		●				◐		●	●		●
Sajaniemi et al. [30]	2008	●	●	●	●						●	●	●	
Sorva [36]	2008		●		●				◐		●	●		●
Kaczmarczyk et al. [16]	2010	●	●		●		◐				●	●		●
Sirkia and Sorva [32]	2012	●	●	●	●		●		●		●			
Altadmri et al. [1]	2015	●	●				◐				●	◐		
Swidan et al. [39]	2018	●	●	●	●						●	●	◐	
Caceffo et al. [3]	2019	●	●		●		◐			●	●	◐		●
progmiscon.org (our work)	2021	●	●	●	●	●	◐	◐	◐	◐	●	◐	◐	◐

visual form). Moreover, our inventory includes those characteristics we believe are most valuable for educators: connection with chapters of textbooks (*Textbooks*), possible sources (*Origins*), and how misconceptions can be employed during teaching practice (*Value*).

6 Conclusions

We propose a *definition* of programming language misconception, describe a way to *organize* a collection of such misconceptions, and introduce progmiscon.org, an inventory based on the above definition and organization. Our definition focuses on misconceptions about PL syntax and semantics. We do believe that the difficulties students have with acquiring pragmatics and strategies are important as well. However, definitions broader than ours make it more difficult to clearly delineate the boundary between misconceptions and difficulties in general. As Sorva states in introducing his unified collection of misconceptions, such a broad body ends up being "a list of not only apples and oranges, but also of tomatoes and the odd dried plum." [37] Our more focused definition provides a solid foundation for deciding whether or not an issue represents a programming language misconception. Moreover, our definition's tight focus on language syntax and semantics enables the unambiguous connection of each misconception to the underlying conceptual structures of the PL, and the definitions in the authoritative language specification. In cases where these specifications are formalized, this connection paves the way for future work on automatic misconception detection by relating student statements to formal specifications of the actual languages.

progmiscon.org is the result of several years of effort collecting, organizing, and documenting programming language misconceptions, including two semester-long video-recorded observational studies targeting Java and JavaScript. It is a living inventory that continues to evolve along several dimensions. We plan to add further evidence and observations of misconceptions encountered while learning to program and references to studies carried out by researchers in the area. With help from the community, we hope to expand progmiscon.org with misconceptions across a richer set of PLs, possibly leveraging attempts to generalize misconceptions across PLs [3] and considering languages targeted at novices such as Scratch [39]. Ultimately, we hope that progmiscon.org will help to *turn misconceptions into learning opportunities*.

Acknowledgments

This work was partially funded by the Swiss National Science Foundation project 200021_184689. We are grateful to Andrea Adamoli for his assistance and contributions to progmiscon.org.

References

[1] Amjad Altadmri and Neil C.C. Brown. 2015. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 522–527. https://doi.org/10.1145/2676723.2677258

[2] Piraye Bayman and Richard E. Mayer. 1983. A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements. *Commun. ACM* 26, 9 (Sept. 1983), 677–679. https://doi.org/10.1145/358172.358408

[3] Ricardo Caceffo, Pablo Frank-Bolton, Renan Souza, and Rodolfo Azevedo. 2019. Identifying and Validating Java Misconceptions Toward a CS1 Concept Inventory. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, Aberdeen Scotland Uk, 23–29. https://doi.org/10.1145/3304221.3319771

[4] Françoise Détienne. 1997. Assessing the Cognitive Consequences of the Object-Oriented Approach: A Survey of Empirical Research on Object-Oriented Design by Individuals and Teams. *Interacting with Computers* 9, 1 (Aug. 1997), 47–72. https://doi.org/10.1016/S0953-5438(97)00006-4

[5] Andrea A. diSessa. 2018. A Friendly Introduction to "Knowledge in Pieces": Modeling Types of Knowledge and Their Roles in Learning. In *Invited Lectures from the 13th International Congress on Mathematical Education (ICME-13 Monographs)*, Gabriele Kaiser, Helen Forgasz, Mellony Graven, Alain Kuzniak, Elaine Simmt, and Binyan Xu (Eds.). Springer International Publishing, 65–84.

[6] Dimitrios Doukakis, Maria Grigoriadou, and Grammatiki Tsaganou. 2007. Understanding the Programming Variable Concept with Animated Interactive Analogies. In *Proceedings of the The 8th Hellenic European Research on Computer Mathematics & Its Applications Conference (HERCMA'07)*. Economical University, Athens, Greece.

[7] 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

[8] Anna Eckerdal and Michael Thuné. 2005. Novice Java Programmers' Conceptions of "Object" and "Class", and Variation Theory. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*. ACM, New York, NY, USA, 89–93. https://doi.org/10.1145/1067445.1067473

[9] Ann E. Fleury. 1991. Parameter Passing: The Rules the Students Construct. *ACM SIGCSE Bulletin* 23, 1 (1991), 283–286.

[10] Ann E. Fleury. 2000. Programming in Java: Student-Constructed Rules. In *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education (SIGCSE '00)*. Association for Computing Machinery, New York, NY, USA, 197–201. https://doi.org/10.1145/330908.331854

[11] Matthias Hauswirth and Andrea Adamoli. 2017. Identifying Misconceptions with Active Recall in a Blended Learning System. In *Data Driven Approaches in Digital Education (Lecture Notes in Computer Science)*, Élise Lavoué, Hendrik Drachsler, Katrien Verbert, Julien Broisin, and Mar Pérez-Sanagustín (Eds.). Springer International Publishing, Cham, 416–421. https://doi.org/10.1007/978-3-319-66610-5_36

[12] Matthias Hauswirth and Andrea Adamoli. 2017. Metacognitive Calibration When Learning to Program. In *Proceedings of the 17th Koli Calling International Conference on Computing Education Research (Koli Calling '17)*. Association for Computing Machinery, New York, NY, USA, 50–59. https://doi.org/10.1145/3141880.3141904

[13] Simon Holland, Robert Griffiths, and Mark Woodman. 1997. Avoiding Object Misconceptions. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '97)*. ACM, New York, NY, USA, 131–134. https://doi.org/10.1145/268084.268132

[14] 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

[15] J. Jackson, M. Cobb, and C. Carver. 2005. Identifying Top Java Errors for Novice Programmers. In *Proceedings Frontiers in Education 35th Annual Conference*. IEEE, USA, 24–27. https://doi.org/10.1109/FIE.2005.1611967

[16] 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

[17] Antti-Juhani Kaijanaho. 2015. Evidence-Based Programming Language Design : A Philosophical and Methodological Exploration. *Jyväskylä studies in computing* 222 (2015).

[18] Brenda Keogh and Stuart Naylor. 1999. Concept Cartoons, Teaching and Learning in Science: An Evaluation. *International Journal of Science Education* 21, 4 (April 1999), 431–446. https://doi.org/10.1080/095006999290642

[19] Colleen Lewis. 2021. Computer Science Teaching Tips. https://www.csteachingtips.org/ [Accessed March 29, 2021].

[20] Linxiao Ma. 2007. *Investigating and Improving Novice Programmers' Mental Models of Programming Concepts*. PhD Thesis. University of Strathclyde, Glasgow, Scotland.

[21] Tara Madhyastha and Steven Tanimoto. 2009. Faring with Facets: Building and Using Databases of Student Misconceptions. *Journal of Interactive Media in Education* 2009, 1 (Feb. 2009), Art. 5. https://doi.org/10.5334/2009-1

[22] Sandra Madison and James Gifford. 1997. *Parameter Passing: The Conceptions Novices Construct*. RIE ED406211. ERIC. 30 pages.

[23] Tanya J. McGill and Simone E. Volet. 1997. A Conceptual Framework for Analyzing Students' Knowledge of Programming. *Journal of Research on Computing in Education* 29, 3 (March 1997), 276–297. https://doi.org/10.1080/08886504.1997.10782199

[24] Jim Minstrell. 2000. Student Thinking and Related Assessment: Creating a Facet-Based Learning Environment. In *Grading the Nation's Report Card: Research from the Evaluation of NAEP*. National Academies Press, Washington, D.C. https://doi.org/10.17226/9751

[25] Roy D. Pea. 1986. Language-Independent Conceptual "Bugs" in Novice Programming. *Journal of educational computing research* 2, 1 (1986), 25–36.

[26] Ralph T. Putnam, D. Sleeman, Juliet A. Baxter, and Laiani K. Kuspa. 1986. A Summary of Misconceptions of High School Basic Programmers. *Journal of Educational Computing Research* 2, 4 (Nov. 1986), 459–472. https://doi.org/10.2190/FGN9-DJ2F-86V8-3FAU

[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] Noa Ragonis and Mordechai Ben-Ari. 2005. A Long-Term Investigation of the Comprehension of OOP Concepts by Novices. *Computer Science Education* 15, 3 (Sept. 2005), 203–221. https://doi.org/10.1080/08993400500224310

[29] Philip M. Sadler and Gerhard Sonnert. 2016. Understanding Misconceptions: Teaching and Learning in Middle School Physical Science. *American Educator* 40, 1 (2016), 26–32.

[30] Jorma Sajaniemi, Marja Kuittinen, and Taina Tikansalo. 2008. A Study of the Development of Students' Visualizations of Program State during an Elementary Object-Oriented Programming Course. *Journal on Educational Resources in Computing* 7, 4 (Jan. 2008), 1–31. https://doi.org/10.1145/1316450.1316453

[31] 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

[32] 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

[33] D. Sleeman, Ralph T. Putnam, Juliet Baxter, and Laiani Kuspa. 1986. Pascal and High School Students: A Study of Errors. *Journal of Educational Computing Research* 2, 1 (Feb. 1986), 5–23. https://doi.org/10.2190/2XPP-LTYH-98NQ-BU77

[34] John P. Smith III, Andrea A. diSessa, and Jeremy Roschelle. 1994. Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *Journal of the Learning Sciences* 3, 2 (April 1994), 115–163. https://doi.org/10.1207/s15327809jls0302_1

[35] Juha Sorva. 2007. Students' Understandings of Storing Objects. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88 (Koli Calling '07)*. Australian Computer Society, Inc., Koli, Finland, 127–135.

[36] Juha Sorva. 2008. The Same but Different Students' Understandings of Primitive and Object Variables. In *Proceedings of the 8th International Conference on Computing Education Research - Koli '08*. ACM Press, Koli, Finland, 5. https://doi.org/10.1145/1595356.1595360

[37] Juha Sorva. 2012. *Visual Program Simulation in Introductory Programming Education*. PhD Thesis. Aalto University, Espoo, Finland.

[38] Juha Sorva. 2013. Notional Machines and Introductory Programming Education. *ACM Transactions on Computing Education* 13, 2 (June 2013), 1–31. https://doi.org/10.1145/2483710.2483713

[39] Alaaeddin Swidan, Felienne Hermans, and Marileen Smit. 2018. Programming Misconceptions for School Students. In *Proceedings of the 2018 ACM Conference on International Computing Education Research - ICER '18*. ACM Press, Espoo, Finland, 151–159. https://doi.org/10.1145/3230977.3230995

[40] Mariana Teif and Orit Hazzan. 2006. Partonomy and Taxonomy in Object-Oriented Thinking: Junior High School Students' Perceptions of Object-Oriented Basic Concepts. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '06)*. Association for Computing Machinery, New York, NY, USA, 55–60. https://doi.org/10.1145/1189215.1189170

[41] 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/doi:10.1007/s10763-010-9203-x

[42] Tobias Wrigstad and Elias Castegren. 2019. Mastery Learning-Like Teaching with Achievements. *CoRR* abs/1906.03510 (2019). arXiv:1906.03510