




Learning to Program: Mapping Errors and Misconceptions of Python Novices to Support the Design of Intelligent Programming Tutors

Lisa van der Heyden¹^a, Fatma Batur²^b and Irene-Angelica Chounta¹^c

¹Department of Human-centered Computing and Cognitive Science, University of Duisburg-Essen, Duisburg, Germany

²Computing Education Research Group, University of Duisburg-Essen, Essen, Germany

Keywords: Programming, Novices, Adaptation, Intelligent Tutors, Python, Errors, Misconceptions.

Abstract: Students often struggle with basic programming tasks after their first programming course. Adaptive tutoring systems can support students' practice by generating tasks, providing feedback, and evaluating students' progress in real-time. Here, we describe the first step for building such a system focusing on designing tasks that address common errors and misconceptions. To that end, we compiled a collection of Python tasks for novices. In particular, a) we identified errors occurring during introductory programming and mapped them to learning tasks; b) we conducted a survey to validate our mapping; c) we conducted semi-structured interviews with instructors to understand potential reasons for such errors and best practices for addressing them. Synthesizing our findings, we discuss the creation of a tasks' corpus to serve as a basis for adaptive tutors. This work contributes to the standardization and systematization of computing education and provides insights regarding the design of learning tasks tailored to addressing errors.


1 INTRODUCTION


Learning to program is considered challenging, and it is often associated with difficulties, errors, and misconceptions. In a systematic review, (Qian and Lehman, 2017) found that students' misconceptions and difficulties in syntactic, conceptual, and strategic knowledge were related to many factors, such as unfamiliarity with syntax, natural language, math knowledge, inaccurate mental models, lack of strategies, programming environments, teachers' knowledge, and instruction. They advocate that more research is needed on students' misconceptions in Computer Science (CS), especially on the development of (mis)conceptions, and that appropriate teaching strategies and tools need to be developed and disseminated to address them. As part of their review, the authors also concluded that there is no generally accepted definition of these problems in computer science education. Some frequently used terms are "misconceptions" (Sorva, 2012), "difficulties" (Du Boulay, 1986), and "errors" (Sleeman et al., 1986). Qian and Lehman suggest that "*misconcep-*


tions per se, are probably best defined as errors in conceptual understanding" (Qian and Lehman, 2017, p. 3). For our work, we prefer to use the general term "errors" most of the time.

One possibility for providing students with individualized learning and practice opportunities is using Intelligent Tutoring Systems (ITS). In particular, Intelligent Programming Tutors (IPTs) focus on teaching programming, also in the context of introductory programming (Crow et al., 2018). Related research on IPTs confirms the effectiveness of such systems (Rivers and Koedinger, 2017) in terms of promoting students' learning. Still, little is known about the specific learning tasks and materials these systems use to guide students' practice. Additionally, learning resources that are available online often are not associated with specific errors, misconceptions, or difficulty levels, and thus, it is not clear how to address potential issues that students may face when using them.

In this work, we aim to address this gap and contribute to the transparent development of learning materials for IPTs that can support novices when learning Python. In particular, our objective is to build a corpus of Python learning tasks that address different difficulty levels and target common errors and misconceptions. We argue that such tasks can be used as a basis for adaptive learning systems to

^a <https://orcid.org/0000-0003-0197-7473>

^b <https://orcid.org/0000-0002-7377-6585>

^c <https://orcid.org/0000-0001-9159-0664>

provide tailored feedback and scaffolding. To create this task corpus, we collected various errors and misconceptions as well as Python tasks. We assigned each task to a misconception or error and one of three difficulty levels. Then, we tested our assumptions about the mapping by asking Python teachers to validate our propositions. With this work, we aim to answer the following research questions (RQs):

RQ1. Do Python instructors agree to the mapping of the tasks and levels of difficulty?

RQ2. Do Python instructors agree to the mapping of the tasks, misconceptions and common errors?

RQ3. What are the concepts students struggle with?

We aim to get insights from Python instructors based on their professional experience regarding the concepts that students often have difficulties with. We use these insights to enrich the collection of programming tasks established and validated in RQ1 and RQ2 to address a wide range of students' needs.

RQ4. What practices do teachers use to support students with difficulties?

We aim to document the teaching practices that instructors recommend for efficiently and effectively supporting students who face difficulties with programming. These practices will accompany the task collection with the aim of guiding the design of automated, adaptive feedback.

We envision that our work contributes to the standardization and systematization of computing education. Additionally, we aim to provide insights regarding the design of learning tasks tailored to addressing common errors and misconceptions in introductory programming.

2 RELATED WORK

Learning to program is a popular and widely researched topic. Specifically, novice programmers are in the spotlight of CS education since research suggests that novices often suffer from a wide range of difficulties that involve acquiring complex new knowledge and related strategies and practical skills (Robins et al., 2003). (McCracken et al., 2001) conducted a multi-national study that examined novice students' programming competence at the end of introductory computer science courses. The results indicated that the majority of students performed poorly and suggested that the problems seemed to be independent of the country and educational system. Students appeared to have the most difficulties with abstracting the problem to be solved from the task description. (Luxton-Reilly et al., 2018) explored trends

and advances in learning introductory programming. Among other things, the authors pointed out that there was a lack of research concerning the specific concepts that were covered in the first few weeks of introductory programming courses.

It is challenging to reach a consensus on the most important and difficult concepts for an introductory computer science (CS1) course due to the diversity of programming languages, pedagogical paradigms, and programming environments that impact the perception of important and difficult topics (Goldman et al., 2008). Similarly, there is no consensus on the correct sequencing in which topics should be taught in a particular language (Robins, 2010). Our desk research revealed that the basic topics of variables, loops, and functions are often cited as challenging topics, which aligns with the work from (Lahtinen et al., 2005).

Learning to program is often accompanied by various difficulties, errors and misconceptions. Research on misconceptions in CS dates back to the 1980s (Du Boulay, 1986). Published misconception or error collections, such as <https://prog miscon.org> or <https://www.csteachingtips.org>, and related research (Sorva, 2012; Kohn, 2017) provide insightful overviews of common misconceptions and errors: Progmiscon offers a curated inventory of programming language misconceptions with a focus on syntactic and semantic knowledge (Chiodini et al., 2021); the CS teaching tips website contains more than 100 teaching tips that are tagged as "*content misconceptions*"; (Sorva, 2012) created an extensive collection of novice misconceptions about introductory programming content; (Kohn, 2017) presented a selection of misconceptions related to syntax, the nature of variables, and assignments. However, none of the aforementioned collections contains information about specific tasks or difficulty levels associated with the misconceptions. We aim to close this gap by a) linking tasks to misunderstandings and errors; and b) classifying tasks into different difficulty levels.

3 METHODOLOGY

To answer our research questions, we followed an iterative design process for designing a collection of Python learning tasks to address errors and misconceptions of novices. First, we conducted a desk review to identify important concepts and common errors in introductory programming. In parallel, we collected Python tasks related to the concepts and the errors identified in the review. Next, we established the mapping of the Python tasks, different levels of difficulty, and common errors. Following this, we de-

signed and conducted a survey study for Python instructors to validate the mapping of the tasks with levels of difficulty (RQ1) and the common errors (RQ2). Additionally, we conducted semi-structured interviews with Python instructors to get insights into further errors, misconceptions, and difficulties of students (RQ3) and the teaching approaches to support struggling students (RQ4). The mapping of tasks, difficulty levels and common errors was adjusted to reflect the results of the surveys and interviews.

3.1 Task Collection

Here, we focus on three programming concepts: variables, loops, and functions since they appear as challenging concepts for novices. For these concepts, we conducted a desk review to document common errors: for variables, we focused on errors related to the assignment and reassignment of values (Kohn, 2017; Du Boulay, 1986); for loops, we focused on specific keywords such as break or else, the understanding of while and for loops, and errors related to off-by-one errors and the assumption that a loop halts as soon as the condition is false rather than finishing the loop's body first (Rigby et al., 2020); for functions, we selected common errors related to the "general" understanding of defining and calling functions, nesting and parameter passing (Kallia and Sentance, 2019).

In parallel, we collected tasks that could be assigned to these errors. We aimed at combining different tasks with expected errors and potential misconceptions. Our task collection can be found in the digital appendix (<https://zenodo.org/records/14712398>). Since the difficulty level of concepts and tasks plays a vital role in this context, we aimed at collecting tasks of different difficulty levels. To that end, we used the taxonomy of (Le et al., 2013; Le and Pinkwart, 2014). The authors proposed a categorization of the degree of ill-definedness of educational problems based on the existence of solution strategies (Le et al., 2013). Reviewing existing intelligent learning environments for programming exercises, (Le and Pinkwart, 2014) suggested that the exercises can be classified into three classes: (1) exercises with one single solution, (2) exercises with different implementation variants, and (3) exercises with different solution strategies. The authors also compared the proposed classification with the PISA proficiency levels for Mathematics, where "the proficiency scale represents an empirical measure of the cognitive demand for each question/exercise" (Le and Pinkwart, 2014, p. 57). They connected class 1 problems with proficiency level 1, class 2 problems with proficiency level 2, and class 3 problems with proficiency level 3 and 4. Hence, we

Create a function named "my_function" and fill in the gap:

```
1 
2 print("Hello from a function")
```

Figure 1: Task F1C1.

Convert the following into code that uses a for loop:

```
2
4
6
8
10
Goodbye!
```

Figure 2: Task L2C2.

argue that the three classes of programming exercises can also be seen as "difficulty levels". We mapped the tasks to three different levels of difficulty, so-called "classes":

- **Class 1 Tasks:** have the lowest level of difficulty and have one single correct solution. An example is a task that describes the problem and asks the user to input one specific value in a specified gap (Figure 1). We reviewed our collected tasks and assigned all tasks that fit to the "one solution strategy, one implementation" approach to class 1.
- **Class 2 Tasks:** are more complicated and have one solution strategy but can be solved by different implementation variants, such as modifying program statements to make an incorrect code snippet work. All of our tasks that followed the "one solution strategy, alternative implementation variant" approach were assigned to class 2. An example is task L2C2 (Figure 2).
- **Class 3 Tasks:** are the most difficult tasks that enable students to implement different solution strategies in different variants. We only found four tasks that had a known number of typical solution strategies and were mapped to class 3. One example is task V1C3 (Figure 3).

This resulted in 33 programming tasks: 11 for variables, 11 for functions and 11 for loops. For the difficulty levels, we assigned 16 tasks to class 1, 13 tasks to class 2 and four tasks to class 3.

Write a loop that sums the values 1 through "end", inclusive. "end" is a variable that we will define later. So for example, if we define "end" to be 6, your code should print out the result "21" (which is 1+2+3+4+5+6). Do not include input statements and do not define "end" to a specific value. We will do this later.

Figure 3: Task V1C3.

3.2 Survey

To validate the mapping of learning tasks and errors (see section 3.1), we designed an online survey to collect input from Python instructors. Specifically, we wanted to gather feedback on whether the Python instructors agree to the mapping of the tasks with the levels of difficulty (RQ1) and the errors (RQ2). The study was conducted with the approval of the Department's Ethics Committee (number 2410CMvL7106). To keep the survey short regarding the completion time, we created three questionnaires: one for variables, one for loops, and one for functions. Each questionnaire was composed of three parts: A short introduction, the main part – where the participants were asked to rate the appropriateness of the difficulty level and whether they thought the tasks appropriately addressed the expected errors – and finally, demographic questions related to the participants' teaching experience. In addition, we included free text fields where participants could provide additional insights. Participation in the survey was anonymous, and we did not collect any sensitive data. We contacted Python instructors from our home institution and via public calls for participation that were distributed via email and social media channels. Upon positive response, we followed up by sending a short description of the research idea, a link to the online survey, and an "instructions" sheet (see digital appendix) that should have been read before the survey was completed. The instructions sheet provided more information about the distinction of the different levels ("classes") and explained the structure of the survey. The questions from the survey can also be found in the digital appendix.

3.3 Interviews

After the survey, we conducted follow-up semi-structured interviews with three individuals who participated in the survey study. We used interview input to gain deep insights into the concepts students struggle with (RQ3) and how the instructors support students in resolving these struggles (RQ4). An interview protocol was created to lead the interviewer through the process. All interviews were conducted online and lasted between 23 and 37 minutes. The first question referred to the survey, and participants were asked whether they wanted to follow up on a topic from the survey in more depth. Then, participants were queried about common errors, potential misconceptions, and difficult concepts beyond the ones that were mentioned in the survey. Participants were also asked about their programming teaching ex-

perience, which concepts they thought students struggled with, and what they did to support students. All participants gave informed consent for their participation. The interviews were analyzed using thematic analysis (Braun and Clarke, 2006) and affinity diagrams (Beyer and Holtzblatt, 1989).

3.4 Participants

Nine Python instructors with different teaching experience participated in the survey (three participants per concept – see table 1).

In total, three participants volunteered to take part in a follow-up interview after filling in the survey. We can't identify which survey participants (table 1) participated in the interview as the surveys were anonymous. Nevertheless, we found out that each interview participant had received another concept so that we could conduct one interview with a person who filled in the survey for variables, one for loops, and one for functions. This allowed us to gain complementary insights.

4 RESULTS

4.1 Do Python Instructors Agree to the Mapping of the Tasks and Levels of Difficulty? (RQ1)

In total, for 21 out of 33 tasks (64%), the participants agreed that the difficulty level was appropriate for the tasks. For 11 tasks, two out of three participants agreed that the mapping was appropriate, while for 1 task, only one out of three participants agreed to the mapping. Figure 4 shows the agreement among the participants for all tasks and concepts. The tasks that were rated with an agreement of 3/3 covered all concepts and all levels of difficulty. Nevertheless, we saw that agreement was established for 9/11 tasks relating to variables, compared to 7/11 tasks for loops and 5/11 tasks for functions. Since we did not have the same number of tasks for each level of difficulty (see Section 3.1), we calculated an agreement rate – this was done by dividing the number of tasks for which participants gave their full agreement by the respective number of available tasks for each class. Since we had a small sample size and some missing values, we decided to use this metric. Class 3 scored the highest agreement rate (75%), followed by class 1 (69%) and class 2 (54%).

Qualitative analysis of participants' input suggests that the main points of participants' disagreement

Table 1: Study Participants.

Participant	Professional Background	Years of Teaching Experience	Questionnaire
P1	Undergraduate/college student	1-3	Loops
P2	Research (staff)	10+	Loops
P3	Graduate/Postgraduate student/PhD candidate	1-3	Loops
P4	Professor	6-10	Functions
P5	Undergraduate/ college student	1-3	Functions
P6	Graduate/Postgraduate student/PhD candidate	1-3	Functions
P7	Graduate/postgraduate student/PhD candidate	1-3	Variables
P8	Research (Staff)	4-5	Variables
P9	Graduate/postgraduate student/PhD candidate	1-3	Variables

concerned the wording of the tasks and the idea that some tasks could be solved differently from the proposed solutions.

4.2 Do Python Instructors Agree to the Mapping of the Tasks and Misconceptions and Common Errors? (RQ2)

For the examination of the mapping of tasks and errors, we had 29 tasks because we did not map specific errors to the four tasks of class 3. For 19 out of 29 tasks (66 %), participants fully agreed on the mapping. For 9 tasks, two out of three participants agreed on the mapping, and for 1 task, only one participant approved the mapping. Figure 4 illustrates the agreement among the participants for all tasks and concepts. The tasks rated with an agreement of 3/3 covered all concepts and both difficulty levels (that is, class 1 and class 2). Similar to the results for RQ1 (Section 4.1), we calculated two agreement rates, one for the concepts of loops, functions, and variables and one for the difficulty levels class 1 and class 2. We observed the highest agreement rate for the concept of loops (90 %), followed by functions (56 %) and variables (50 %). Comparing the difficulty levels, the agreement rate for the mapping of class 1 was slightly higher (69 %) than for class 2 (62 %). The main points on which the participants disagreed were the wording of the tasks and the related errors. Furthermore, they also made assumptions about possible other solutions to the tasks, which could then lead to potential new errors.

4.3 What Are Concepts Students Struggle with? (RQ3)

To further investigate the concepts that students struggle with potentially leading to errors, we interviewed 3 (I1, I2, and I3) out of the 9 participants. Through the interviews, participants gave several indications

of potential errors that they considered relevant and that were not included in our original task collection. We documented these errors and divided them into six error categories. Three categories referred to variables, loops, and functions, as in the original scheme. In addition, we identified three more error categories for (a) syntax errors (*“use of methods without ()”* (I3)); (b) data structures/flow (*“using libraries”* (I1), *“lists and tuples”* (I3), *“how data flows in the program”* (I3)); and (c) other that included errors that were related to the code quality (*“no sufficient documentation”* (I2)) or concepts and errors that were not mentioned multiple times (e.g. *“conditions, the if/else”* (I1), *“confusing the assignment and comparison signs”* (I3)).

4.4 What Practices Do Teachers Use to Support Students with Difficulties? (RQ4)

When asked what practices teachers apply to address the students’ struggles, I2 and I3 set the focus on understanding (*“promoting the understanding”* (I2)) so that students a) learn how to write code, b) understand *“why to write code in a certain way”* (I3), and c) are enabled to implement code in different ways instead of *“bare remembering”* (I3). Both I2 and I3 also talked about the benefit of learning with problems (*“I guess the most useful skill in terms of teaching is to break down problems into pieces”* (I2)). Furthermore, I2 and I3 mentioned collaboration (for example, working in groups and discussing the results to find a solution (I2)) and guidance (having long lectures and the time for discussions with students (I3); staying with the students in the lab and looking things up with them (I2)) as important factors in the classroom, and general recommendations like teaching how to write good code (recommendations to being more organized in terms of naming (I2) and use comments for the code (I2)) or provide information about the software setup (I2).

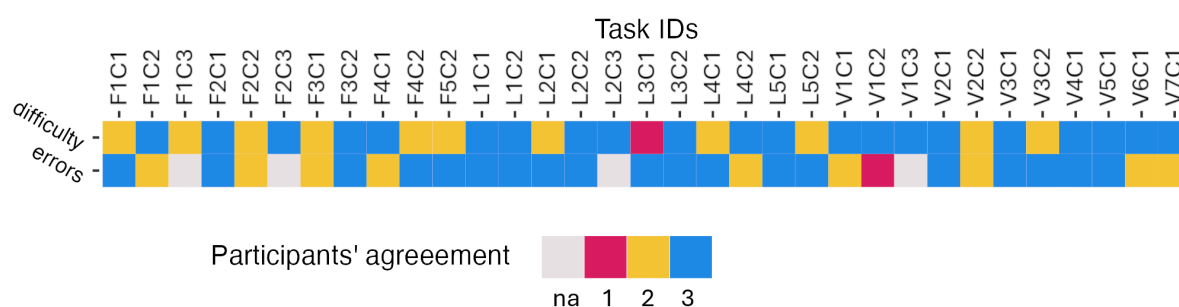


Figure 4: Agreement rates from participants' survey responses for the levels of difficulty and the related errors. A value of 3 means that 3 out of 3 participants expressed their agreement, a value of 2 means that 2 out of 3 participants expressed their agreement, and so on.

When discussing teaching practices (section 4.3), the participants also mentioned future perspectives for learning and teaching programming. I2 introduced this topic with the statement *“things will change, but the problem-solving skill is something you can train”* pointing out that it is important to teach students to break down problems into pieces. I2 also mentioned *“in a broader scope of programming”* that it is important to teach and learn how to translate problems into a symbolic language. I1 said that focusing on concepts would be a *“good idea”* as it would help *“this attitude of computational thinking”*. Following, I1 elaborated that the programming concepts were more about computational thinking and a mindset of debugging, going step by step, thinking about the concepts in a broader way, and applying them to other fields. To talk about the importance of teaching programming, I3 pointed out an important difference between (1) teaching how to write instructions in the correct order and with the correct syntax and (2) teaching in terms of thinking about structures of code, how to optimize code, why certain things are implemented and so on. I3 said that Large Language Models (LLMs) were good for the former but didn't perform well on the latter. In general, none of the participants seemed to be opposed to the use of LLMs, provided they were used to support students.

5 DISCUSSION

5.1 Mapping of Tasks with Levels of Difficulty, Errors and Misconceptions (RQ1, RQ2)

Our findings suggested that the lowest level of agreement for the mapping of tasks with difficulty levels was established for tasks related to functions or tasks that are assigned to class 2. This may reflect that when

dividing tasks into three difficulty levels, it is more difficult to evaluate a task in the middle (class 2) than a task that represents one extreme or the other (class 1 or class 3).

Concerning the mapping of tasks with errors, there was a high agreement for tasks related to loops. The task descriptions were specific and did not leave room for interpretation. Participants' feedback often focused on reflecting about the learning task instead of the related errors. This may suggest that the classification of the programming exercises (Le and Pinkwart, 2014) can meaningfully reflect “levels of difficulty”.

5.2 What Concepts Do Students Struggle with and How Can Teachers Support? (RQ3, RQ4)

From our interviews, it emerged that participants put more emphasis on concepts than on syntax. Understanding was more important than writing code correctly, as this could also be outsourced to tools like LLMs or accompanying materials. This idea is in line with current research on LLMs, which shows promising results in areas such as solving introductory programming tasks (Finnie-Ansley et al., 2022) or using LLMs for assessment and feedback generation for a given input (Hellas et al., 2023). One participant suggested including hyperlinks or similar information about syntax when using our task corpus so that students could completely focus on the concepts. This idea complements the proposal of (Crow et al., 2018) to link additional resources to the intelligent and adaptive component of an IPT after they had identified the gap that only a few IPTs were equipped with comprehensive reference material. It is important to point out the importance of context: Even if we saw some agreement among our participants, related research shows that educators only form a weak consensus about which mistakes are most frequent and

that educators' perceptions of students' errors and the errors that students make are different (Brown and Al-tadmri, 2014). Taking the feedback of the surveys and the interviews into account, we adjusted our task collection twice. A detailed overview of the adjustments and the reasoning is provided in the digital appendix.

5.3 Theoretical and Practical Implications

In this work, we built on existing tasks and mapped these tasks to different levels of difficulty and potential errors. By asking Python instructors to review this initial task collection and conducting follow-up interviews, we adjusted and refined our enriched task collection. We envision that our process of creating and adjusting the task collection can help inform future research and provide an opportunity for practical application of the task collection. On the one hand, our tasks can be used for practicing specific concepts in introductory programming courses. On the other hand, our results reveal which tasks are generally suitable for investigating errors of novices. Therefore, our results can be used to create tasks that follow the same approaches and concepts to investigate common errors of novices. For the future, we also see potential in connecting misconception collections with other taxonomies - such as Bloom's taxonomy (Bloom et al., 1956) or SOLO taxonomy (Biggs and Collis, 1982) - and thus achieving even more granular differentiations than we have with our three levels of difficulty. We plan to integrate our existing collection of tasks with errors and misconceptions as base for an ITS for programming, that focuses specifically on detecting misconceptions and helping students overcome them. As mentioned by (Crow et al., 2018), we see a chance to equip ITS with comprehensive reference material, especially to deal with students' misconceptions. The levels of difficulty can be used to adjust the tutor to the knowledge state of the student and to provide customized feedback.

6 CONCLUSION

In this paper, we explored the alignment between Python tasks for the concepts of variables, loops and functions, and the mapping of these tasks with three levels of difficulty and related errors. We created a task collection that was refined based on our survey results for the mapping of tasks with difficulty levels (RQ1) and with related errors (RQ2). Gaining additional insights from Python instructors for concepts that students struggle with (RQ3) and informa-

tion on how the instructors support students (RQ4) helped us to refine the task collection once again. After the two phases of the refinement of the mapping, we ended with 30 tasks (33 tasks initially). During the first refinement, we removed 1 task and adjusted the mapping for 10 tasks. During the second refinement, we removed 2 more tasks and adjusted the mapping for 9 tasks (2 of these 9 tasks were adjusted for the second time). As a result, we provide a validated collection of tasks mapped to three difficulty levels and common errors. The digital appendix provides all reference materials for further use of the tasks (<https://zenodo.org/records/14712398>).

We acknowledge that this study has several limitations, such as the small sample size, the choice of programming language, and the initial concepts that may limit generalizability and transferability. We acknowledge that our focus on novices might have neglected relevant misconceptions and errors about programming. Concerning the idea of creating learning tasks for an adaptive tutor, we based our mapping of the difficulty levels on the work of (Le et al., 2013) and (Le and Pinkwart, 2014) and did not use a cognitive taxonomy like Bloom's taxonomy (Bloom et al., 1956) or SOLO taxonomy (Biggs and Collis, 1982). Possibly, instructors are more familiar with cognitive taxonomies and categorizing on this basis could have led to different results.

To investigate whether our findings apply to "real world" classroom settings, studies are required that make use of the actual task collection. If our task collection proves to be valid for teaching Python and addressing related errors for novices, one could also try to investigate whether the task collection can be transferred to different programming languages. As future work for ourselves, we will design an adaptive tutor for Python programming that uses the validated task collection and conceptually similar tasks. We intend to set up an ITS that adapts the tasks and, thus, the different levels of difficulty to the learner's level of knowledge. The planned system will allow us to provide tailored support to novices who encounter misconceptions and errors when learning Python programming.

REFERENCES

- Beyer, H. and Holtzblatt, K. (1989). *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann, San Francisco, CA.
- Biggs, J. B. and Collis, K. F. (1982). *Evaluating the quality of learning: The SOLO taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, New York.

- Bloom, B. S., Engelhart, M. D., Furst, E. J., Hill, W. H., and Krathwohl, D. R. (1956). *Taxonomy of Educational Objectives: Handbook 1 Cognitive Domain*. Longmans, London.
- Braun, V. and Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101.
- Brown, N. C. and Altadmri, A. (2014). Investigating novice programming mistakes: educator beliefs vs. student data. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, ICER '14, page 43–50, New York, NY, USA. Association for Computing Machinery.
- Chiodini, L., Moreno Santos, I., Gallidabino, A., Tafliovich, A., Santos, A. L., and Hauswirth, M. (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, page 380–386, New York, NY, USA. Association for Computing Machinery.
- Crow, T., Luxton-Reilly, A., and Wuensche, B. (2018). Intelligent tutoring systems for programming education: a systematic review. In *Proceedings of the 20th Australasian Computing Education Conference*, ACE '18, page 53–62, New York, NY, USA. Association for Computing Machinery.
- Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1):57–73.
- Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., and Prather, J. (2022). The robots are coming: Exploring the implications of openai codex on introductory programming. In *Proceedings of the 24th Australasian Computing Education Conference*, ACE '22, page 10–19. Association for Computing Machinery.
- Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., and Zilles, C. (2008). Identifying important and difficult concepts in introductory computing courses using a delphi process. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '08, page 256–260, New York, NY, USA. Association for Computing Machinery.
- Hellas, A., Leinonen, J., Sarsa, S., Koutchme, C., Kujanpää, L., and Sorva, J. (2023). Exploring the responses of large language models to beginner programmers' help requests. In *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, ICER '23, page 93–105, New York, NY, USA. Association for Computing Machinery.
- Kallia, M. and Sentance, S. (2019). Learning to use functions: The relationship between misconceptions and self-efficacy. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 752–758, New York, NY, USA. Association for Computing Machinery.
- Kohn, T. (2017). *Teaching Python Programming to Novices: Addressing Misconceptions and Creating a Development Environment*. ETH Zürich.
- Lahtinen, E., Ala-Mutka, K., and Järvinen, H.-M. (2005). A study of the difficulties of novice programmers. *SIGCSE Bull.*, 37(3):14–18.
- Le, N.-T., Loll, F., and Pinkwart, N. (2013). Operationalizing the continuum between well-defined and ill-defined problems for educational technology. *IEEE Transactions on Learning Technologies*, 6(3):258–270.
- Le, N.-T. and Pinkwart, N. (2014). Towards a classification for programming exercises. In Trausan-Matu, S., Boyer, K., Crosby, M., and Panourgia, K., editors, *Proceedings of the 2nd Workshop on AI-supported Education for Computer Science at the 12th International Conference on Intelligent Tutoring, Systems (ITS)*, Berlin, Heidelberg. Springer-Verlag.
- Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., and Szabo, C. (2018). Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, page 55–106, New York, NY, USA. Association for Computing Machinery.
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students. *SIGCSE Bull.*, 33(4):125–180.
- Qian, Y. and Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ.*, 18(1).
- Rigby, L., Denny, P., and Luxton-Reilly, A. (2020). A miss is as good as a mile: Off-by-one errors and arrays in an introductory programming course. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*, ACE'20, page 31–38, New York, NY, USA. Association for Computing Machinery.
- Rivers, K. and Koedinger, K. R. (2017). Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *International Journal of Artificial Intelligence in Education*, 27:37–64.
- Robins, A. (2010). Learning edge momentum: a new account of outcomes in cs1. *Computer Science Education*, 20(01):37–71.
- Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(02):137–172.
- Sleeman, D., Putnam, R. T., Baxter, J., and Kuspa, L. (1986). Pascal and high school students: A study of errors. *Journal of Educational Computing Research*, 2(1):5–23.
- Sorva, J. (2012). *Visual program simulation in introductory programming education*. School of Science, Aalto University.