Learnersourcing at Scale for Introductory Programming: Longitudinal Data Collection on the Python Tutor Website

Philip J. Guo
UC San Diego
La Jolla, CA, USA

Julia M. Markel
UC San Diego
La Jolla, CA, USA

Xiong Zhang
University of Rochester
Rochester, NY, USA

ABSTRACT

The Python Tutor website (pythontutor.com) currently gets over 10,000 daily active users executing around 100,000 pieces of code daily. We have been experimenting with collecting large-scale data about learners' thought processes while coding. For instance, we created a learnersourcing system that elicits explanations of potential misconceptions from learners as they fix errors. We have deployed this system for the past three years to the Python Tutor website and collected 16,791 learner-written explanations. By inspecting this dataset, we found surprising insights that we did not originally think of due to our own expert blind spots as programming instructors. We are now using these insights to improve compiler and run-time error messages to explain common novice misconceptions.

INTRODUCTION

(This is a 2-page lightning paper that summarizes our Learning@Scale 2020 work-in-progress paper [2].)

Novices suffer from a large variety of misconceptions when learning computer programming, ranging from misunderstandings about syntax to incorrect mental models of code execution [4]. To help novices out, experts often write explanations for common misconceptions, but a fundamental shortcoming of this approach is that experts can suffer from the *expert blind spot* [3], also known as the *curse of knowledge*: They forget what it was like to be a novice, so they have a hard time empathizing with novice struggles. As a result, their explanations might be incomplete or inadvertently use jargon that is too advanced for novices to properly comprehend.

To help overcome these expert blind spots, we took a learnersourcing approach [5] where we built a system to collect crowdsourced explanations of programming misconceptions by using learners as the crowd. We deployed this system within Python Tutor [1], an online code editor and visual debugger with tens of thousands of daily active users and over ten million total users so far (Figure 1). Our intuition is that by prompting learners there to provide explanations *at the exact moment they overcome a particular coding struggle*, we can collect a corpus of written explanations that can augment educational materials to overcome common expert blind spots.

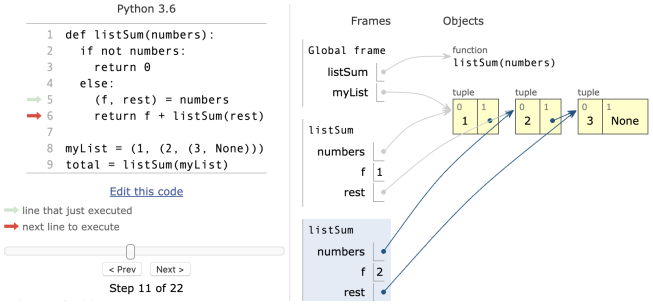

Figure 1. Python Tutor [1] lets users write code (left) and see how it executes step-by-step with visualizations of its run-time state (right).

We deployed this system online for three years (2017–2020) and collected 16,791 learner-written explanations spanning a variety of novice coding misconceptions. By inspecting these explanations, we (as experienced programming instructors) have found surprising insights that we did not originally think of due to our own expert blind spots. We can use these insights to augment tools with more novice-friendly explanations.

LEARNERSOURCING SYSTEM PROTOTYPE

Figure 2 shows our system prototype embedded within the Python Tutor website: a) The user can directly write code on the site or copy-paste in code examples and problems they are working on from online tutorials. They will inevitably encounter errors when they are coding, which are either syntax errors or run-time errors. Figure 2a shows a run-time error when line 3 is executed, with Python reporting "UnboundLocalError: local variable 'y' referenced before assignment." These message are hard for novices to understand [4] since they use technical jargon and do not indicate *why* someone might have made that error (i.e., what their misconception was). Syntax errors in Python show even less helpful messages, most commonly "SyntaxError: invalid syntax." b) After they eventually fix the error, here by adding '`global y`' on line 3, they run the code again and the error is gone. c) At that moment, our system pops up a dialog showing the user's previous code, its error message, and the question "What misunderstanding do you think caused this error?" d) The user can write a response or dismiss the dialog. Hopefully they write an explanation using terms that fellow learners can empathize with better than Python's built-in error messages; in this example, the user wrote "'global' needed when both global & local variable have same name." Their text is saved to our corpus, along with the history of their coding session, which includes both the erroneous and fixed code. This way, we collect the learner's thoughts ***right at the moment when they just fixed an error*** so that their misconception is at the top of their mind.

Sixth SPLICE Workshop at L@S 2020 "Building an Infrastructure for Computer Science Education Research and Practice at Scale", SPLICE'20, August 12, 2020, Virtual Event

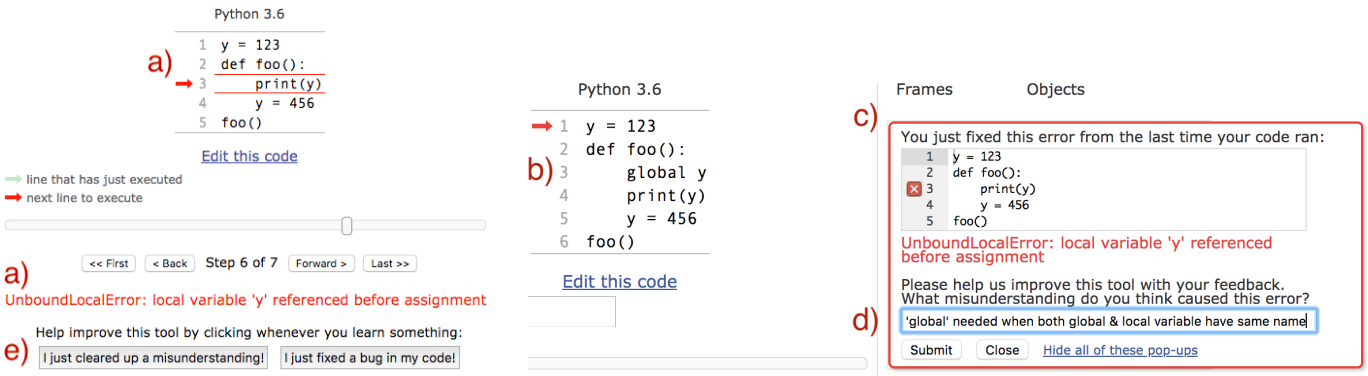

Figure 2. a) The user encounters an error while coding in Python Tutor, b) fixes their code and re-runs, c) sees a pop-up box showing the error they just fixed, and d) writes an explanation about what misconception led them to make that error. e) The user can also write freeform explanations at any time.

Our prototype automatically pops up a dialog whenever the user fixes a syntax or run-time error. Note that since it only detects that an error message has disappeared, it is possible to get false positives when the user, say, completely changes their code in between attempted executions. In that case, the original error may be gone, but they simply moved on without trying to fix it. In the future, we could heuristically suppress false positives by taking diffs between code executions and not popping up a dialog box if the code diff is too large.

This prototype can collect explanations for a variety of errors that novices encounter, which include both compile-time (syntax) and run-time errors that Python automatically flags. However, it is possible for code to execute to completion without Python issuing any errors, but it still produces incorrect results. These are known as semantic or logic errors [4], and they are impossible for a system to detect without a test suite. Since Python Tutor users mostly write freeform code without a test suite, we wanted to provide a way to collect learner-sourced explanations for semantic errors. Thus, we added two buttons to the bottom of the interface, shown in Figure 2e. The user can click either "I just cleared up a misunderstanding!" or "I just fixed a bug in my code!" at any time, which then prompts for a written explanation. These buttons give users a way to share *freeform explanations* for moments when they get a sudden insight about what is wrong with their code

PRELIMINARY FINDINGS

We deployed this feature to Python Tutor in 2017, so we have collected over three years of data so far. We only looked at data for Python 3, which is the most common language that learners write on the website. (Other supported languages include Python 2, Java, JavaScript, C, and C++.) Learners submitted 16,791 explanations that were at least 10 characters long. That is an average of around 10 submissions per day.

We discovered a variety of misconceptions including: 1) *Linguistic*: These reveal learners' troubles mapping between the syntax of natural languages (e.g., English) and code syntax. Our own expert blind spots caused us to never think about many of these since we had been programming for so long that code syntax came "naturally" to us. But learners wrote insightful explanations for errors including capitalization, singular/plural forms, and implicit pronoun references in code. 2) *Mathematical*: Since many introductory programming problems deal with numbers and math, we discovered some surprising misconceptions from mapping between the syntax of math and code, such as uses of equals and assignment statements. 3) *Polyglot*: Finally, many learners came to Python from other languages like Java, C, C++, or JavaScript. As polyglot programmers, they wrote explanations for some common cross-language errors such as API naming mismatches and syntactic differences. See our paper for more details [2].

This preliminary analysis revealed some of our own expert blind spots as programming instructors. While many misconceptions seem apparent in retrospect, we never even considered many of them before reading these learner-submitted explanations. Some of these have been reported in prior studies mostly from classroom settings [4], but the novel contributions of our scalable technique and large data set are: 1) We can automatically collect data that tells us exactly which Python error messages map to specific misconceptions, and how frequently those occur in the wild. 2) We logged all of the code that led to those errors, which we can later distill into error-inducing code snippets. 3) We have explanations written in the learners' own words, which can help us create custom error messages using terminology that learners can better relate to.

Acknowledgments: This material is based upon work supported by the National Science Foundation under Grant No. NSF IIS-1845900.

REFERENCES

[1] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-based Program Visualization for CS Education *(SIGCSE '13)*. ACM.

[2] Philip J. Guo, Julia M. Markel, and Xiong Zhang. 2020. Learnersourcing at Scale to Overcome Expert Blind Spots for Introductory Programming: A Three-Year Deployment Study on the Python Tutor Website. In *Conference on Learning @ Scale (L@S '20)*.

[3] Mitchell J Nathan, Kenneth R Koedinger, and Martha W Alibali. 2001. Expert blind spot: When content knowledge eclipses pedagogical content knowledge. In *Proceedings of the third international conference on cognitive science*. Beijing: University of Science and Technology of China Press, 644–648.

[4] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017).

[5] Sarah Weir, Juho Kim, Krzysztof Z. Gajos, and Robert C. Miller. 2015. Learnersourcing Subgoal Labels for How-to Videos *(CSCW '15)*. ACM.