ASEE 2022 ANNUAL CONFERENCE
Excellence Through Diversity
MINNEAPOLIS, MINNESOTA, JUNE 26TH-29TH, 2022 ASEE
Paper ID #37608

Programming learners struggle as much in Python as in C++ or Java

Chelsea Gordon

Research Lead at zyBooks

Roman Lysecky (Professor)

Frank Vahid (Professor)

Frank Vahid is a Professor of Computer Science and Engineering at the University of California, Riverside, since 1994. He is co-founder and Chief Learning Officer of zyBooks, which creates web-native interactive learning content to replace college textbooks and homework serving 500,000 students anually. His research interests include learning methods to improve college student success especially for CS and STEM freshmen and sophomores, and also embedded systems software and hardware. He is also founder of the non-profit CollegeStudentAdvocates.org.

Programming learners struggle as much in Python as in C++ or Java

Chelsea Gordon* , Roman Lysecky*,†, Frank Vahid*,°

*zyBooks (www.zybooks.com)

†Dept. of Electrical and Computer Engineering, Univ. of Arizona

° Dept. of Computer Science and Engineering, Univ. of California, Riverside

Abstract

Many computer science instructors believe switching college-level introductory programming courses to Python, versus languages like C++, Java, or C, smooths the learning of programming, due in part to Python's syntax, dynamic types, and interpreted execution. Studies providing quantitative support for that belief are rare, in part due to the challenge of creating comparative studies without extensive confounding factors. However, today one of the top textbooks in C++, Java, and Python all happen to come from the same publisher and have similar style and content, have similar off-the-shelf programming assignments that instructors can use, incorporate a cloud-based development environment and auto-grader that records every run, and each is used by tens of thousands of students at hundreds of universities. As such, a unique opportunity exists to compare student performance data across those three languages. We obtained such data from the publisher for 56 courses that met specific criteria (18 Python, 16 C++, 22 Java) totaling 6,804 students. As a potential indicator of whether Python students had a smoother learning experience, we compared median time and number of attempts on nearly-identical off-the-shelf programming assignments across the three languages. Python students exhibited no reduction in time or attempts. Manual investigation into the students' programs suggest that problem understanding and programming logic, and not language features, may be the main hurdles to learning programming.

Introduction

College-level introductory programming courses (e.g., CS1) are the gateway to computing majors and as a result, the primary factor in attracting students to the major and determining which students proceed further in the field. Unfortunately, many students struggle in such courses, which exhibit low grades and high fail rates (30% or more) [1]. Research suggests that the high fail rate is a result of factors such as poor advising, poor math skills, poor lab design, poor time management, lack of feedback, and low comfort levels in the learning environment. [2],[3],[4],[5],[6],[7]. Even though programming language does not fall into one of these factors, many CS1 instructors have switched from using languages like C++ or Java to using Python in order to reduce drop out. One reason is the belief that there is a smoother learning curve in Python due to a simpler more-intuitive syntax, dynamic typing of variables, and simpler installation and execution (interpreted vs compile+run). Additional reasons include the availability of graphics and other libraries that quickly enable more powerful programs, increased motivation due to commercial popularity, and the usefulness of Python for data analysis in one's future career (especially for students only taking one CS course). [8]

Evidence supporting the smoother learning curve is scarce, with many supporting arguments being anecdotal. This scarcity is in part due to the many confounding factors that are typically introduced when switching from one language to another, including different textbooks, course organization, programming environments, and programming assignments. One small study surveying 45 CS students who had learned both C++ and Python found that the students ranked several aspects of programming as slightly easier in Python than in C++ [9], but these were 45 students who all learned from the same courses, resulting in numerous confounding variables.

However, a unique opportunity exists as of 2020: one of the top used textbooks for each of Python, Java, and C++ are from the same publisher, zyBooks. All three textbooks have similar content organization and authoring style, and each comes with the same simple built-in cloud-based integrated development environment (IDE) for developing and grading programs ("zyLabs"). Each textbook has a set of over 50 sample programming assignments that instructors can choose from, with many of those assignments being nearly identical across languages. Furthermore, each is used by many thousands of students across hundreds of universities.

In this paper, we provide results comparing student performance on nearly identical programming assignments across those three languages. We present data on the metrics of time spent and number of submissions, as potential indicators of how smoothly students are learning programming. We also present data on the metric of percentage of students who seem to be experiencing excessive struggle. As will be seen, the data for those metrics turn out to be roughly equal for Python, C++, and Java.

Methods

Textbooks and homework

We collected data for three introductory programming "textbooks", in Python, Java, and C++, intended for use in college-level CS1 and CS2 courses. The books were authored for the web and thus primarily exist online only (they can be printed, but are not optimized for paper). Each book was written in a similar style in terms of organization, coverage, depth, and style. The books use an interactive approach in which each topic is taught using minimal text, animations, and learning questions. The key observation we make here is that, although some variability exists among the three books, that variability is substantially less than for nearly any other collection of top-selling books in those three languages.

In addition to the core material, these textbooks each include integrated coding homework problems. In these problems, students complete code in a programming window or indicate the output of given code, and each submission is immediately auto-graded. Many of these problems are very similar in quantity, difficulty, and style.

Programming environment and sample assignments

Beyond the textbook and homework features mentioned above, all three online textbooks come with an optional add-on tool that supports the programming assignment ("lab") component of a course. The lab tool includes a simple integrated development environment (IDE) in which

students can write and run programs, in any of various languages (Python, Java, C++, C, Javascript, MATLAB, etc.). This built-in cloud IDE feature eliminates the differences of Python versus C++/Java with respect to installation and execution, as all three operate identically -- students type code in the IDE window, and then press "Run". For C++/Java, pressing Run does both compilation and execution, whereas for Python, Run simply starts interpreted execution, but the experience from the student's perspective is the same. This similarity in running all three languages is true of many cloud-based IDEs today, where users simply press Run. Students can provide input and observe the program's output.

The lab tool includes an auto-grader that can automatically test a submitted program for input/output correctness against various test cases, or can run unit tests (calling a student's function/method directly and checking the returned results). Students are immediately told their current score, and can reattempt the program multiple times. Figures 1 and 2 show an example lab assignment, part of a solution, and example test results.

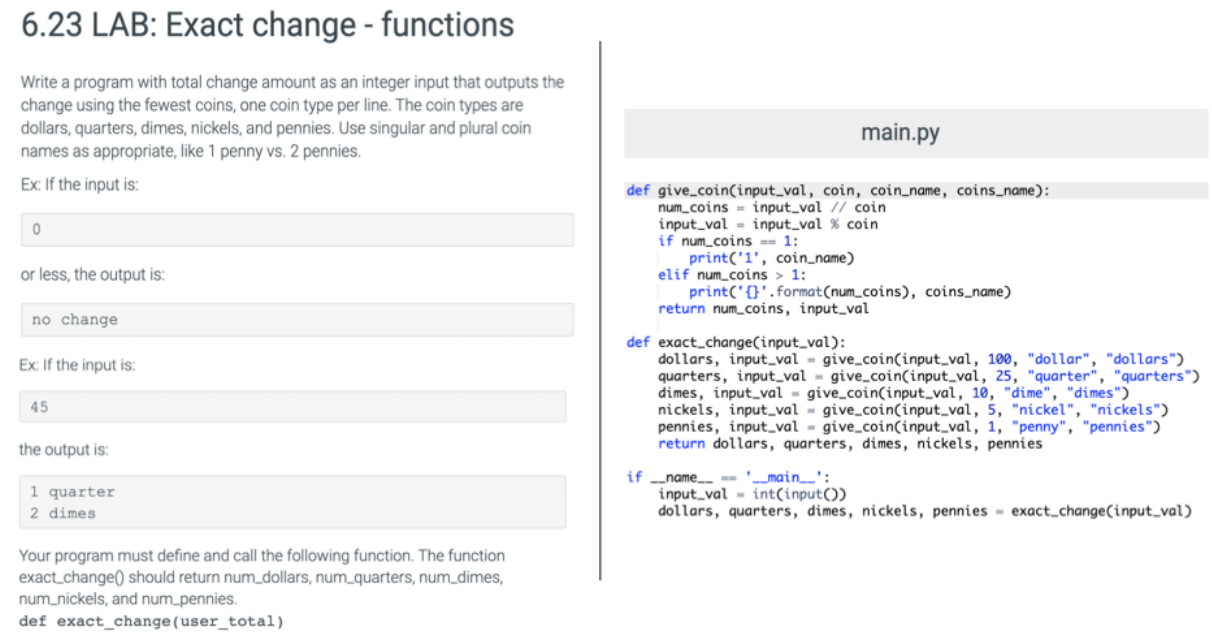

Figure 1: Example zyLab assignment. In the figure, the assignment instructions are shown on the left, and submitted code on the right.

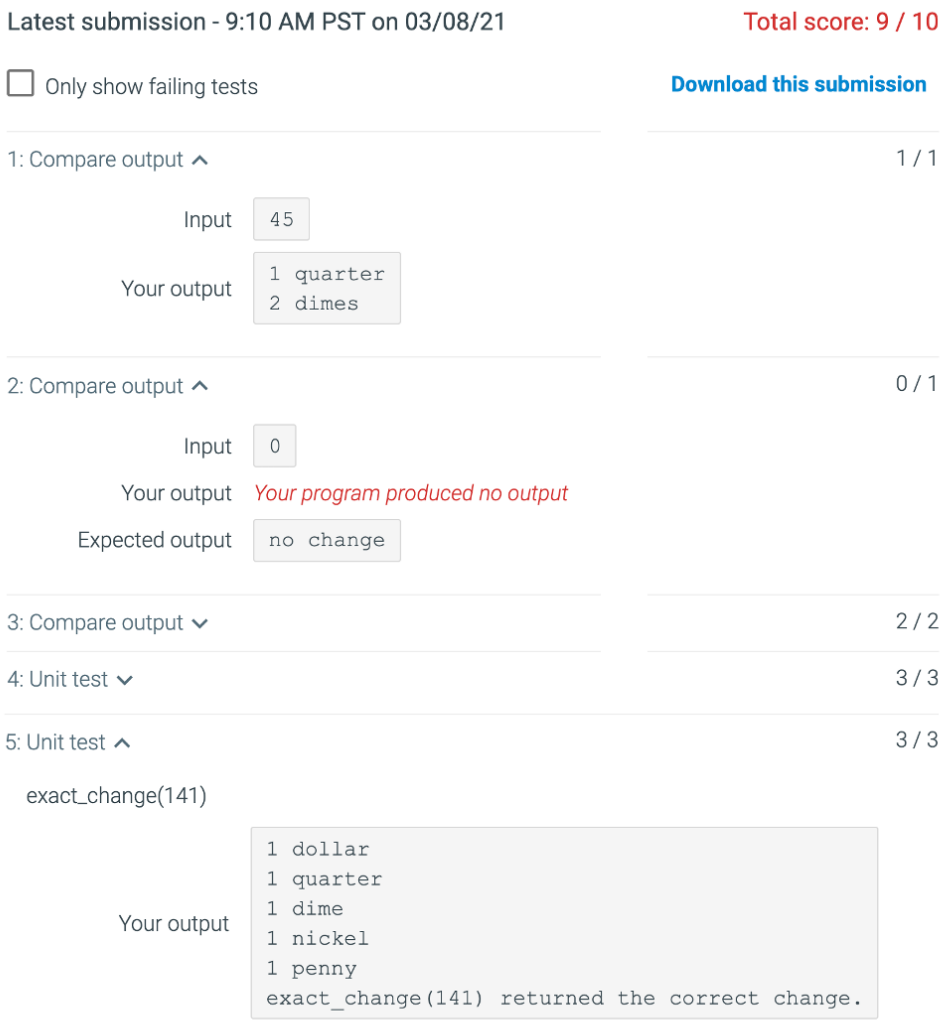

Figure 2: Test results for the example assignment. The submitted code results in a grade of 9/10, losing one point for not returning "no change" when the input is 0.

Course and sample lab selection

We included courses in this analysis that met the following criteria:

- Semester course offered in in Spring/Fall of 2020 or 2021
- 40 or more students
- Made use of numerous sample labs that are used across the three languages
- Syllabus specified that the course was for both computing majors and non-majors

The final criterion was included because many Python courses target non-majors only, and thus one might expect those courses would have students spending more time and having more struggle on those sample labs.

Among those courses, we found numerous cases of sample labs used by the courses. We selected sample labs in particular courses for analysis according to the following criteria:

- For each course, a lab was included if (1) the lab was configured to the built-in zyBooks IDE (rather than allowing the files to be uploaded), and (2) the lab had been completed by at least 50% of students (suggesting it was a required lab, not optional, because students completing optional labs might not be representative of the typical student population)
- A lab was included in the overall analysis if that lab was used in at least 3 total courses in that language (by at least 50% of students, as specified above)

We found 9 labs that met the above criteria in the 18 Python courses, 22 Java courses, and 16 C++ courses that were included in our analysis, involving 2,076 Python students, 2,592 Java students, and 2,136 C++ students. Some of the labs were not used frequently enough in every language, so those are shown only for the languages where it was used. The 9 labs, the chapter in which they appear, and our student sample size are shown in Table 1.

		Chapter			**Students**		
	zyLab description	Python	Java	C++	Python	Java	C++
Convert to binary	Student reads an integer, and outputs a string of 0s and 1s representing that number in binary.	5	4	4	713	612	324
Check integer string	Student reads in a string representing an integer, and outputs yes if every character is a digit 0-9.	7	4	4	492	235	633
Middle item	Student reads a sorted list of integers and outputs the middle number, assuming the list length is always odd.	8	5	5	463	416	161
Swapping variables	Student reads in two integers out outputs an array of those integers, which their positions swapped.	6	6	6	383	353	160
Step counter	Student creates a function that takes a double as a parameter (number of feet walked) and returns an integer (number of steps walked).	6	6	6	0	662	545
Driving costs - functions	Student creates a function that takes parameters miles, mpg, and dollars/gallon, and returns the cost to drive those miles.	6	6	6	550	891	680
Convert to binary - functions	Student creates two new functions in a program to read an integer and output the integer in binary. One function returns a string of 1's and 0's representing the integer in binary. The second function returns the input string in reverse.	6	6	6	584	135	158
Exact change - functions	Student creates a function that takes in the total change amount in cents, calculates the change using the fewest coins, and outputs the change, one coin per line.	6	6	6	303	217	83
Car value	Student completes a class "Car" by creating an attribute purchase_price and the method print_info() that outputs the car's information	9	7	7	181	253	160

Table 1: Labs used in this study, the chapter of the lab, and number of observed students.

Results

Figure 3 shows the median time spent on each lab, for each of C++, Java, and Python (the error bars represent 95% confidence intervals). The labs are sorted by the order that they typically appeared in the textbook. If a particular language had an easier learning curve, we would expect that language's students to be spending less time on average. However, the data shows that in none of the languages were students spending significantly less time. Figure 4 depicts the distribution of time spent for each lab in all three languages. This gives a picture of the variability in the data, and the large upper skew, which supports using the median as the measure of central tendency, rather than the mean.

As another indicator, Figure 5 shows the median total number of runs made by the student, including both develop and submit runs. Number of runs is another potential indicator of the learning curve, but again, no language's students have significantly fewer runs.

Another potential indicator of a smoother learning curve is the percentage of struggling students. A struggling student is a student who is spending an excessive amount of time or effort completing a lab. Even if a language has a higher average time, it could be that other languages have a higher percentage of students struggling: Most students are programming successfully (even if slightly slower), but a non-trivial number are terribly lost. We defined a struggling student as a student who either (1) spent twice as much time on the lab as the average or had twice the number of runs as the average, or (2) attempted the lab but did not achieve the lab's maximum score. Figure 6 shows that none of the languages had a significantly lower percentage of struggling students. Figure 7 shows the proportion of struggling students who did not eventually receive full points on the lab. The labs with higher proportions here may be indicative of struggle on programming logic, in particular, whereas those with lower proportions might indicate struggle with syntax.

Thus, none of the three indicators listed above support a smoother learning curve for any of the three languages of Python, C++, and Java.

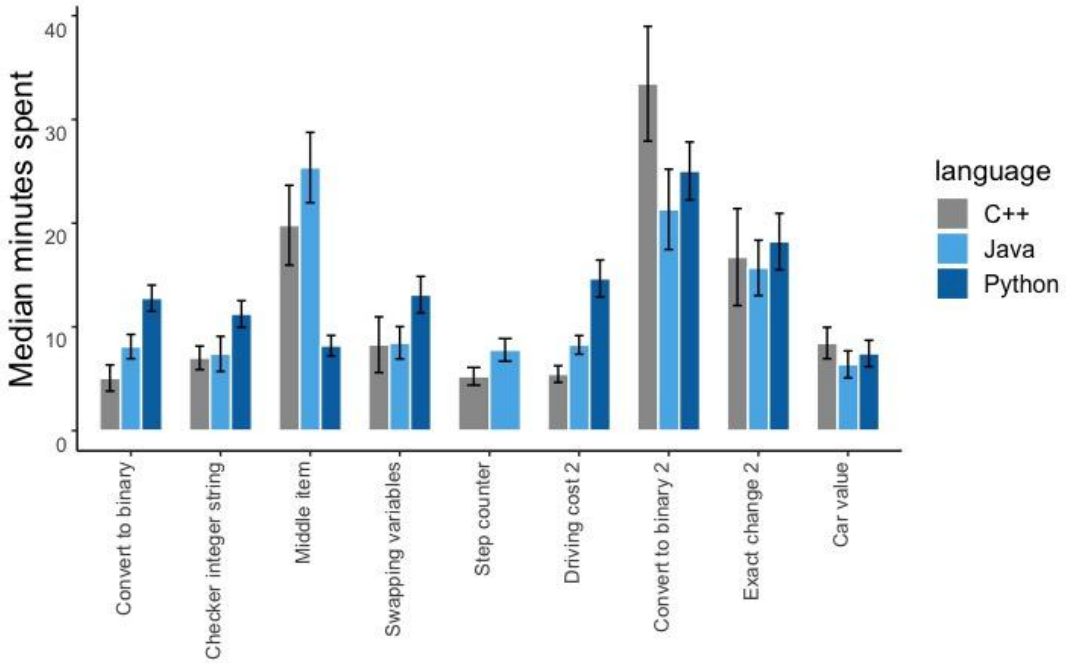

Figure 3: Median time spent completing 9 different zyLabs. Error bars represent 95% confidence intervals.

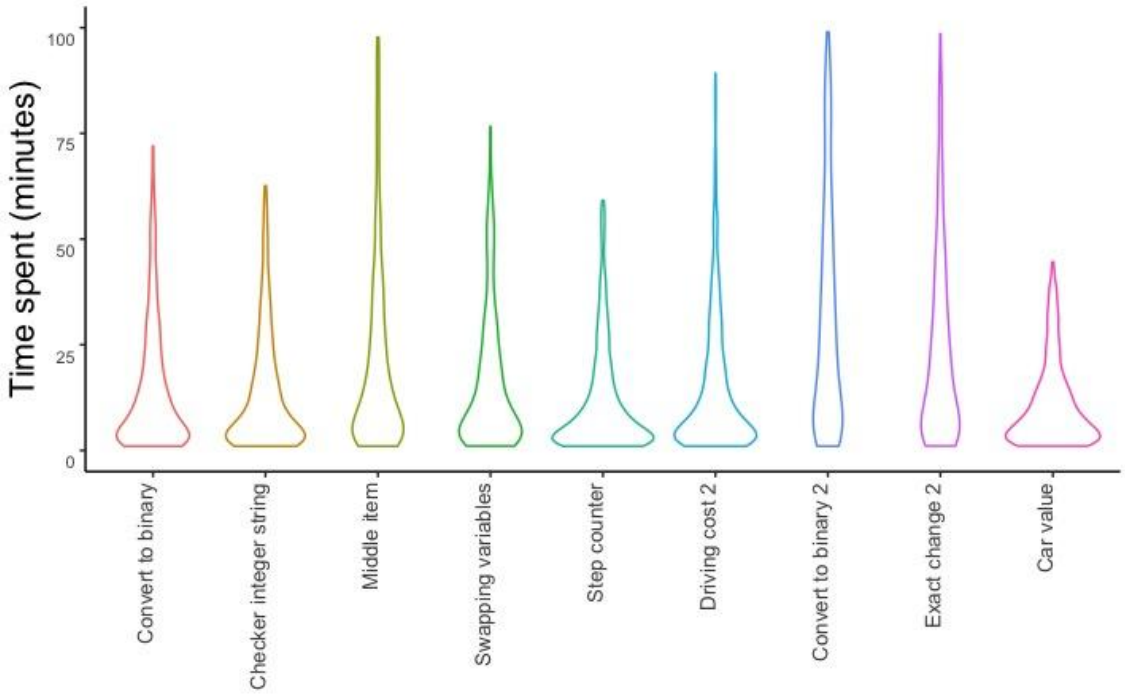

Figure 4: Distribution of total time spent for each lab across all three languages.

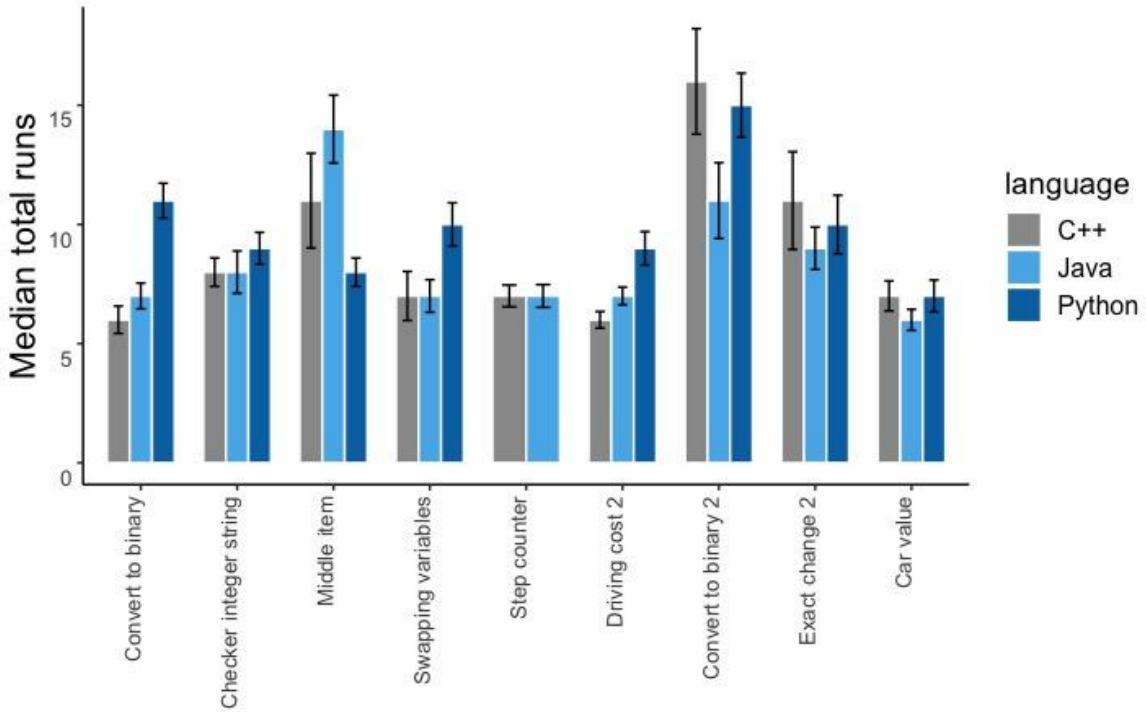

Figure 5: Median number of runs attempted on 9 different zyLabs. Error bars represent 95% confidence intervals.

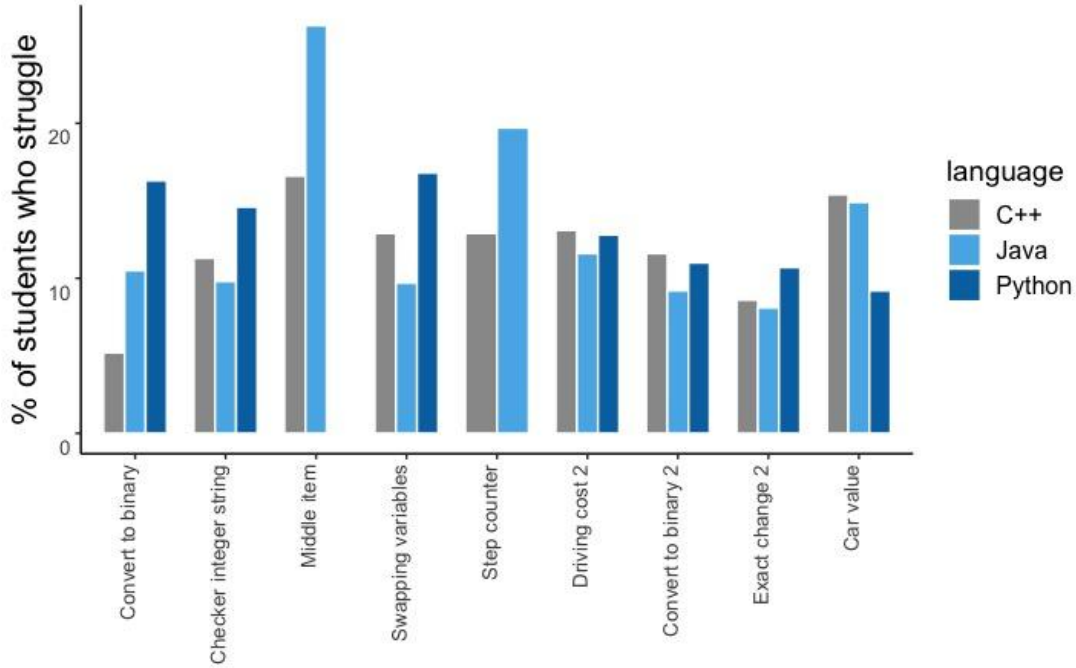

Figure 6: Percent of students who struggled on 9 different zyLabs.

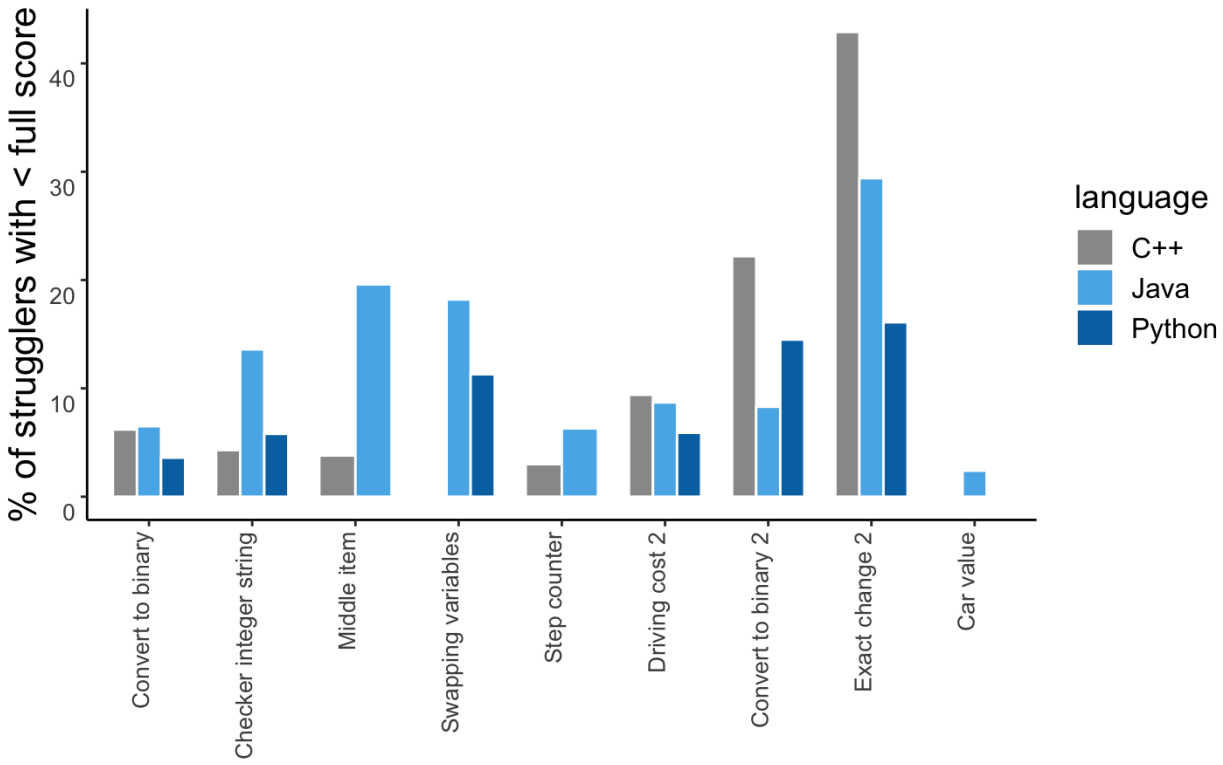

Figure 7: Percent of struggling students who did not eventually get full points for the lab.

Discussion

Our analyses of student programs suggest that the language differences between Python, C++, and Java are not a key barrier to learning programming. These results are consistent with earlier work by Alzahrani [11], who performed a comparison between C++ and Python, but on smaller homework problems. They too found no evidence of reduced Python struggle, and in fact found that Python students struggled more on those small homework problems, even when controlling for classes targeting majors or non-majors, and for university type.

Given that our results indicate that student struggle may not be driven by the programming language, we propose two other factors that contribute to a large share of student struggle, based on our observations. The first is students struggling to achieve correct program logic. For example, to find the minimum of three numbers a, b, and c, one approach with correct program logic is to write:

```
max = a
```

```
if b > max
   max = b
if c > max
   max = c
```

In contrast, an approach with incorrect program logic that we have informally observed in many submissions is:

```
if  a > b and a > c
   max = a
else if b > a and b > c
   max = b
else
   max = c
```

That logic is wrong when some values are equal: a=9, b=9, c=2 will yield max=2. The differences in language syntax between C++, Java, and Python wanes in comparison to detecting or understanding that the above logic is wrong.

Another common source of struggle is misunderstanding the problem itself. If asked to return the value with the smallest magnitude among three given numbers, such as returning -3 if given 5 -8 -3, a misunderstanding of the problem would be to instead believe the correct value to return is the smallest value or -8.

We have observed that each language has some unique pitfalls that may become sources of struggle. For example, C++ students may have difficulty discovering the bug of writing "if (a = b)" instead of "if (a == b)" (the former assigns a with b rather than comparing for equality), whereas Python students may struggle wondering why the following program, for inputs 3 4, outputs 34 instead of 7 (reason: input() needs to be cast to an integer, else Python's dynamic typing creates a and b as strings):

```
a  = input()
b = input()
print(a + b)
```

Noting such pitfalls of profession-targeted languages like Python, C++, and Java, some instructors have experienced good results starting with a learner-focused language to learn basic programming logic, then transitioning to a commercial language [12].

There are many reasons for choosing among Python, C++, or Java for an introductory programming language, but the data presented suggests that a smoother learning curve may not be one of those reasons. This point is especially relevant in light of the trend of schools switching intro classes to Python, given the challenge of transitioning from Python to a more strongly-typed language like Java or C++ in subsequent CS coursework, as done in many computer science departments. In fact, we have observed some schools switch their intro courses from Python back to Java or C++, due to the difficulty of such later transition coupled with the non-manifestation of substantial learning curve improvement. Lewis [13] summarizes a list of factors that should go into choosing a programming language that could be a useful starting point.

Conclusion

We examined a unique data set involving thousands of students across dozens of courses attempting nearly identical programming assignments in introductory courses. Upon examining time spent, number of runs, and rates of potential struggle, the data showed no evidence of any of Python, C++, or Java being significantly better or worse in terms of a smoother learning curve. Of course, other reasons may exist for choosing a language like Python for an intro course, such as graphics libraries, data science focus, or usefulness for scripting by non-majors. Absent such other reasons, however, instructors seeking to improve their CS1 courses may wish to focus more on course features rather than language, such as teacher quality, active learning techniques inside and outside class, pair programming, mastery grading, a more comfortable learning environment, and more.

References

[1] J. Bennedsen and M. E. Caspersen. "Failure rates in introductory programming: 12 years later," *ACM Inroads* 10, 2, 30–36. June, 2019.

[2] T. Beaubouef and J. Mason. "Why the high attrition rate for computer science students: Some thoughts and observations," *SIGCSE Bulletin*, vol 37. 103-106. 2005

[3] L. J. Barker and K. Garvin-Doxas. "Making visible the behaviors that influence learning environment: A qualitative exploration of computer science classrooms," *Computer Science Education*, 14(2):119–145, 2004.

[4] R. Boyle, J. Carter, and M. Clark. "What makes them succeed? entry, progression and graduation in computer science," *Journal of Further and Higher Education*, 26(1):3–18, 2002.

[5] K. Garvin-Doxas and L. J. Barker. "Communication in computer science classrooms: Understanding defensive climates as a means of creating supportive behaviors," *ACM Journal of Educational Resources in Computing*, 4(1):1–18, April 2005.

[6] J. Philip and R. Ventura. "Identifying predictors of success for an objects-first CS1," *Computer Science Education*, 15(3):223–243, 2005.

[7] P. Kinnunen and L. Malmi. "Why students drop out CS1 course?" *In Proceedings of the second international workshop on computing education research* (ICER '06). *Association for Computing Machinery*, New York, NY, USA, 97–108. 2006

[8] K. Agarwal, and A. Agarwal. "Python for CS1, CS2 and beyond," *Journal of Computing Sciences in Colleges*, vol. 20. 262-270. 2005

[9] M. Ateeq, H. Habib, A. Umer and M. Rehman. "C++ or Python? Which One to Begin with: A Learner's Perspective," *Proceedings - 2014 International Conference on Teaching and Learning in Computing and Engineering*, LATICE 2014. 64-69.

[11] N. Alzahrani, F. Vahid, A. Edgcomb, K. Nguyen, and R. Lysecky. "Python versus C++: An analysis of student struggle on small coding exercises in introductory programming courses," In *SIGCSE 2018 - Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Vol. 2018-January, pp. 86-91). Association for Computing Machinery, Inc. https://doi.org/10.1145/3159450.3160586

[12] J.M. Allen and F. Vahid. "Teaching Coral before C++ in a CS1 Course," In ASEE Virtual Annual Conference, 2020.

[13] M. Lewis. Picking a Language for Introductory CS — The Argument Against Python *Medium*. (2021, February 3). https://itnext.io/picking-a-languages-for-introductory-cs-the-argument-againstpython-4331cca26cfa