Performance of Python CS1 Students in Mid-level non-Python CS Courses

Richard J. Enbody and William F. Punch
Department of Computer Science and Engineering
Michigan State University
East Lansing, Michigan
[enbody,punch]@cse.msu.edu

ABSTRACT
If you change the CS1 language to Python, what is the impact on the rest of the curriculum? In earlier work we examined the impact of changing CS1 from C++ to Python while leaving CS2 in C++. We found that Python-prepared CS1 students fared no differently in CS2 than students whose CS1 course was in C++, even though CS2 was taught in C++ and covered the same topics as in previous years. Was that an anomaly? What happens in the next tier of courses? When our CS1 was first changed to Python there were many students who had taken CS1 in C++ still in the system. The result is that there is a cadre of students with either CS1 in Python or CS1 in C++ moving together through our curriculum. This one-time occurrence is an opportunity to study the students with many variables fixed. Our next tier of courses is a C-based computer organization course, a C++ based object-oriented software design course, and a data structures course. We found that the students who started with Python fared as well as the CS1 C++ students. As before, the best predictor of performance was their college GPA. Python versus C++ CS1 preparation was not a predictor of performance in any course. We conclude again that in our C++ based curriculum changing CS1 to Python had no negative impact on student performance and did not require any significant change in those subsequent courses.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education—computer science education, curriculum

General Terms
Measurement, Experimentation

Keywords
CS1, Python, curriculum, intro. to programming

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE '10, March 10–13, 2010, Milwaukee, Wisconsin, USA.
Copyright 2010 ACM 978-1-60558-885-8/10/03 ...$10.00.

1http://www.slashdot.org

1. INTRODUCTION
If you want to start a vigorous debate (fight) in the field of computer science, ask "What is the best first programming language?" Not only does the question generate passionate debate, but it is also a recurring question on the Internet. For example, during the summer of 2009 there were at least four discussions on Slashdot¹, a popular website for computer folk, that started or devolved to that topic.

As teachers of first-time programmers, it is both confusing and understandable as to why this debate persists. Confusing because the programming language is typically not the biggest problem the first-time programmer faces. Rather, most first-time programmers struggle with problem solving, turning a problem into a series of steps to effectively solve it. Understandable because the obvious, novel element of a first programming class is the learning of a new topic, a programming language.

The question before us here is: what is the impact on the curriculum of changing the language in the first programming course? In particular, what effect(s) does it have on the students and what changes must be made in subsequent courses, if any?

2. IT'S PROBLEM SOLVING
We embarked on this path because we believe that an important goal of a first course in computer science should be the ability to use programming as a useful problem-solving tool. After much discussion, we have decided that our goal can be rather simply stated. We hope that, subsequent to our course, a student would have the following thought: "Hey, I'll just write a program to solve that!" That is, after completing an introductory course a student should have a new tool in their kit, and one that they will actually use in school and later.

After fifteen years of teaching C++ in CS1 it was apparent to us that one semester of C++ does not provide students with that kind of problem solving tool, one which would foster the above thought. C++ requires more exposure than a single semester to be effective in this way. This need for effectiveness in a single semester becomes more important given that we have seen non-majors become the majority in our CS1 course. A typical CS1 class at MSU has about 25% CS majors, and about 20% other engineering students leaving more than 55% as majors from all walks of the university. As we have not been able to define course outcomes, that would be different for non-majors, we have resisted splitting
the class. In addition, since most non-majors only want one course in programming we would serve them best by providing them with a useful tool when they were done. Wouldn’t that also be good for majors?

Redefining course goals in this way opens up a variety of possibilities for the first language. However, the question is: what is the impact of changing the CS1 language on the rest of the CS curriculum? In our case, the curriculum has been C++ based for fifteen years.

We had multiple goals:

- Students should have a useful tool after one semester (and actually use it outside of this class).
- CS majors should be prepared for a C++ based curriculum.

In our SIGCSE 2009 paper [2], we showed that the Python-prepared students did as well as the non-Python prepared students in a C++ based CS2 course. What happens after that? Oldham [5] noted that, after adopting Python in CS1, they observed no difference in ETS exam scores in their graduates, but provided no measurements. Here we present objective measures to the question: what happens next?

3. USEFUL TOOL?

Because we raised the point, let’s briefly digress to look at the first goal: students should have a useful tool. We have not yet found a hard and fast measure for this goal. However, anecdotal evidence indicates that it is being met to some extent. In fifteen years of teaching CS1 with C++ neither of us ever had a non-major ask for help in applying programming to some problem they were solving outside of class. In the two years of teaching Python in CS1 there have been many such requests. Here are a few from the most recent semester:

- wrote a slot-machine game.
- wrote a program to calculate and keep track of his golf handicap.
- wrote a program for a science lab.
- wrote a program to rearrange a spreadsheet for work.

In each case, the student came by for some guidance. In the spreadsheet example, the student was having difficulty with the CSV file. She had found Python’s csv module, but forgot the “U” option when opening the Excel-generated CSV file. This semester has brought students to us who had the problem mostly solved, but needed a little help. Of course, we didn’t hear from students who did not need help. Maybe the students with C++ never needed any help, but that doesn’t seem to be a reasonable explanation for what we are seeing.

3.1 Python

What about Python helps students have a useful tool after one semester of study? We believe that Python has lower “cognitive load.” By this, we mean that the features made available by Python tend to directly help in solving common problems. Furthermore, the focus of Python is typically “One way to do it,” avoiding overload of the introductory student with too many options. With a lower load for learning the language, students can focus more on problem solving. We cannot yet prove that is the case, but that is what we have observed.

For example, in Python you can iterate through nearly every data structure, either built-in or downloaded, using a for statement. We introduce strings early as a first data structure and instruct the students on how for iterates through each element of a string, that is, each character. When later faced with new data structures such as lists, tuples, dictionaries, and sets, the students already know how to examine all the elements in that structure: they use the for as before. The concept even extends beyond traditional data structures. A file can be iterated through in the same way, by default grabbing a line at at time using the for iterator. Learning “one thing” that applies so broadly makes the student’s job easier, focusing less on new language issues for every problem and focusing more on applying what they already know to a new problem.

Another aspect is the direct availability of useful data structures and their associated methods such as dictionaries, lists, and sets. These are not “advanced” topics to be taught later, but are integral to Python’s approach and, as such, allow one to crack difficult problems earlier. Python’s string handling facilities are one example. They allow one to process text easily, especially text files. With tools like these, it is easier to tackle problems early on using real world data from publicly available sources.

In the tone of [9] we’ve had students work with real data such as stock market data analysis, building a classifier for breast cancer data, digging through sports data to figure out who was the “best” player, “six degrees of Kevin Bacon,” etc. In addition, Python provides powerful tools for both drawing and graphing.

And we can do all that in one semester using Python!

We have summarized our experience by writing a CS1 textbook using Python [7].

3.1.1 Other benefits for CS majors

Obviously, one benefit of Python in CS1 is that CS majors get experience in a second programming language—something that is generally agreed to be good [4, 1]. In that tenor, Python’s ability to work as a high-level scripting language can be useful in subsequent courses, especially in how well it plays with operating systems (e.g. using the os module) and with other languages.

4. IMPACT ON CURRICULUM

How do CS majors fare, especially in subsequent courses? The sequence in our CS curriculum begins with CS1 and CS2—the courses we considered in our earlier study. The third course is discrete structures, but that is unrelated to programming so we have not included it. The next three courses: computer organization, object-oriented software design, and algorithms & data structures complete the six-course core of our curriculum. Here are their official descriptions:
### Table 1: Results by course. Python students are shown in parentheses, non-Python students without parentheses.

<table>
<thead>
<tr>
<th>Course</th>
<th>Count</th>
<th>Grade Mean (Grade SD)</th>
<th>GPA Mean (GPA SD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSE 320</td>
<td>20 (18)</td>
<td>2.75 (0.94)</td>
<td>3.24 (3.31)</td>
</tr>
<tr>
<td>CSE 331</td>
<td>17 (10)</td>
<td>2.77 (0.85)</td>
<td>3.26 (3.30)</td>
</tr>
<tr>
<td>CSE 335</td>
<td>30 (9)</td>
<td>3.15 (0.06)</td>
<td>3.31 (3.29)</td>
</tr>
<tr>
<td>Total</td>
<td>51 (23)</td>
<td>3.15 (0.06)</td>
<td>3.31 (3.29)</td>
</tr>
</tbody>
</table>

### 4.1 Analysis

To answer the question of how Python-prepared students fared in subsequent courses we collected data on the students. We got approval for the study of students from our Institutional Review Board (IRB), and using the approved process we received permission from 74 students who represented most of the 98 individuals across the three courses. There were 23 students who took CS1 with Python and 51 other students, most of whom took CS1 with C++, but a couple transferred in CS1 courses based on Java. The breakdown is shown in Table 1 with the number of participants in each course and their associated mean course grade (four-point scale), with the value in parentheses used to indicate the Python group. Many of the students were enrolled in more than one course so the total’s column is not the sum of the individual courses. The semester was Fall 2008—when the students from the Fall 2007 Python-based CS1 course reached that level.

The question we wished to ask was whether the difference in course grade mean was significant between the two populations in the three courses.

#### 4.2 t-test

For the analysis we used the R statistical language [6]. By default, the R two-sample t-test uses the null hypothesis that the means of the two populations are equal. We look for a \( p \) value at the level of 0.05 or less to reject that hypothesis and accept the alternate hypothesis that there is a difference in the means. The results are shown below for the three classes:

1. A significant difference of the mean course grade for students who took Python in CS1 versus students who did not take Python in CS1.
2. A significant difference of the mean student GPA for students who took Python in CS1 versus students who did not take Python in CS1.

Table 2 lists the \( p \) values. Because the \( p \) values are greater than 0.05 for all three grade means and GPA means, we do not reject the hypothesis that the means are different, so we conclude that the means of Python students and non-Python students are not significantly different.

Therefore, the t-test indicates that there was no significant difference in the mean grades of Python versus non-Python students in any of the three courses. Maybe the Python students were smarter. However, there was also no significant difference in the mean GPAs associated with Python versus non-Python students, meaning that (measured by GPA, at least) the Python students were not “smarter” than the other students.

#### 4.3 Linear regression analysis

We continued with our statistical analysis of the three groups. Using R, we wanted to see if any of the following factors were useful as a linear predictor of performance for the class grade:

1. ACT-Math score
2. GPA
3. Python in CS1 vs. non-Python in CS1

<table>
<thead>
<tr>
<th>Mean Tested</th>
<th>CSE 320</th>
<th>CSE 331</th>
<th>CSE 335</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p ) for course-grade mean</td>
<td>0.57</td>
<td>0.83</td>
<td>0.82</td>
</tr>
<tr>
<td>( p ) for GPA mean</td>
<td>0.56</td>
<td>0.85</td>
<td>0.88</td>
</tr>
</tbody>
</table>

Table 2: \( p \) values for the indicated means for the Python versus the non-Python samples.
Using the linear modeling package \textit{lm}, we created the following linear model used \textit{R} to approximate the shown $\beta$ factors:

\[
\text{CourseGrade} = \beta_0 + \beta_1 \text{ACT-Math} + \beta_2 \text{GPA} + \beta_3 \text{Python vs nonPython}
\]

What the linear model results report are the $\beta$ factor approximations, plus a $p$ value for each factor. The $p$ values are again used to reject the null hypothesis, which in this case is that the particular $\beta$ value is 0. The $p$ value results are listed in Table 3. For the ACT-Math and Python factors, the $p$ values are all greater than 0.05 so we do not reject the hypothesis that they are 0. That is, we conclude that they are 0, so they do not contribute significantly to the course grade. However, for the GPA factor the $p$ values are all less than 0.05 so we reject the hypothesis that they are 0. That is, we conclude that they are different than 0 so they contribute significantly to the course grade.

Therefore, the linear regression analysis listed in Table 3 indicates that only GPA was a significant predictor of course performance. The other factors, Python vs. non-Python and ACT-Math, were not deemed significantly different from 0, meaning that they were not useful for predication.

### Table 3: $p$ values for the three $\beta$ factors in each of the three courses.

<table>
<thead>
<tr>
<th>Factor</th>
<th>CSE 320</th>
<th>CSE 331</th>
<th>CSE 335</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT-Math</td>
<td>0.31</td>
<td>0.44</td>
<td>0.09</td>
</tr>
<tr>
<td>Python</td>
<td>0.60</td>
<td>0.99</td>
<td>0.60</td>
</tr>
<tr>
<td>GPA</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

5. CONCLUSION

When we proposed replacing C++ with Python in CS1, the question was raised about the impact of that change on the rest of the curriculum. Because students with C++ CS1 preparation remained in the system, this moment provided an opportunity to compare the Python vs. non-Python students. In our previous work we looked at their performance in a C++ CS2 course and found no difference. In this study we looked at the next level of courses, and found again that the Python-prepared students performed as well as the non-Python students in spite of the fact that these courses were based on C and C++.

We conclude that starting students with Python has not had a negative impact within a C++ based curriculum. It would have been nice to conclude that the Python students did better in these courses, but the evidence does not appear to support that hypothesis.

6. REFERENCES