Python Versus C++: An Analysis of Student Struggle on Small Coding Exercises in Introductory Programming Courses

Nabeel Alzahrani[1], Frank Vahid[1,3], Alex Edgcomb[1,3], Kevin Nguyen[1] and Roman Lysecky[2,3]
[1] Computer Science and Engineering, University of California, Riverside
[2] Electrical and Computer Engineering, University of Arizona
[3] zyBooks, Los Gatos, California
nalza001@ucr.edu, vahid@cs.ucr.edu, aedgcomb@cs.ucr.edu, knguy092@ucr.edu, rlysecky@ece.arizona.edu

ABSTRACT

Many teachers of CS 1 (introductory programming) have switched to Python rather than C, C++, or Java. One reason is the belief that Python's interpreted nature plus simpler syntax and semantics ease a student's learning, but data supporting that belief is scarce. This paper addresses the question: Do Python learners struggle less than C++ learners? We analyzed student submissions on small coding exercises in CS 1 courses at 20 different universities, 10 courses using Python, and 11 using C++. Each course used either the Python or C++ version of an online textbook from one publisher, each book having 100+ small coding exercises, expected to take 2-5 minutes each. We considered 11 exercises whose Python and C++ versions were nearly identical and that appeared in various chapters. We defined struggle rate for exercises, where struggle means a student spent excessive time or attempts on an exercise. Based on that rate, we found the learning for Python was not eased; in fact, Python students had significantly higher struggle rates than C++ students (26% vs. 13%). Higher rates were seen even when considering only classes with no prerequisites, classes for majors only, or classes for non-majors only. We encourage the community to do further analyses, to help guide teachers when choosing a CS 1 language.

ACM Reference format:
Nabeel Alzahrani, Frank Vahid, Alex Edgcomb, Kevin Nguyen and Roman Lysecky. 2018. Python Versus C++: An Analysis of Student Struggle on Small Coding Exercises in Introductory Programming Courses. In *SIGCSE '18: 49th ACM Technical Symposium on Computer Science Education, Feb. 21–24, 2018, Baltimore, MD, USA*. ACM, NY, NY, USA, 6 pages.DOI: 10.1145/3159450.3160586

1 INTRODUCTION

Python is growing in popularity in introductory programming classes (CS 1). Various factors are stated for switching from languages like C, C++, or Java. One is that Python is interpreted (also known as a scripting language), allowing students to interact immediately by typing print statements or simple calculations, and avoiding some of the complexities of compiling and then running. Another factor is that Python's syntax is simpler, thus preventing students from getting bogged down in syntax errors, and instead allowing students to focus on higher-level programming concepts. A third factor is that Python comes with graphics and other libraries that can make introductory programming courses more engaging for students, who can analyze real data, create graphics-based programs like video games, etc. Other reasons include increasing use of Python in industry, and studies showing fewer lines of code and/or increased productivity among experienced programmers. Key hopes by those who switch is to decrease attrition in CS 1 courses and to attract more people to computing degrees.

But, many teachers disagree, and continue to teach C, C++, or Java in CS 1. Reasons include a belief that new learners should think precisely about details like data types, that learners should not rely so heavily on library functions, that C/C++/Java (or variations) are widely used in industry especially in domains like mobile apps, and that learning Python after C/C++/Java is easier than the other way (strict to less strict being easier than less strict to strict). Some teachers indicate frustration that students who learn Python in CS 1 have trouble in later courses in C/C++/Java, not wanting to be bothered by details. Some engineers state new hires don't understand or respect underlying resources, which can lead to problems on commonly-constrained platforms. Furthermore, cloud-based coding systems for C/C++/Java, where students code in a web window and press "Run", reduce the interpreted/scripting benefit of Python for beginning students.

In previous work, Enbody [1, 2] used Python and C++ for groups in CS 1. They compared the Python-group and C++-group on three outcomes: final exam grade, programming projects scores, and course final grade, and found no significant differences. Using progression analysis, they also found that programming language features had no effect on students' performance in CS 2. For object-oriented programming (OOP), Goldwasser [3] reported that students in CS 1 were overwhelmed by the syntax and semantics of C++ and Java, and found Python provided a simple and consistent model to teach OOP. Prechelt [4] found programmers using a scripting language like Python can solve the same problem with less code and higher productivity vs. a system language such as C++. Zelle [5] found that scripting languages are simpler, safer, and more flexible vs. system languages like C and C++. Guo [6] stated that Python is the most used CS 1 language in top-ranked U.S. research universities. Such related research

SIGCSE '18, February 21–24, 2018, Baltimore, MD, USA.

ACM ISBN 978-1-4503-5103-4/18/02…$15.00
https://doi.org/10.1145/3159450.3160586

suggests how Python can be helpful to teach programming skills. We sought however to determine from the students' experiences whether Python led to less student struggle.

This paper describes our analysis of student struggle rates on identical small coding exercises used in Python and C++ classes across 20 universities. Struggle means spending excessive time or making excessive attempts on such exercises. Some teachers (like ourselves) believe struggle is an important metric because struggle can lead to frustration, and excessive/repeated frustration can lead to students giving up. Struggle is one way (but not the only way) of estimating the learning curve of a language. This paper introduces the small coding exercises, defines a struggle metric, provides data comparing struggle rates for Python and C++ in CS 1 courses showing that struggle rates for Python are not lower (and are actually higher), summarizes a manual investigation into the student submissions to better understand those rates, discusses possible reasons that Python's struggle rates are not lower and are actually higher, and provides conclusions.

2 SMALL CODING EXERCISES

zyBooks [8] is a publisher that creates online textbooks for introductory C, C++, Java, and Python, with those textbooks being very similar but adapted to each language. Over 100 small coding exercises are embedded at the end of sections throughout those books. The exercises are cloud-based, meaning students code directly in a web window and press "Run" to execute their code, so no substantial difference exists between C++'s compile/execute approach and Python's interpreted approach. We abbreviate those exercises in this paper as CA (coding activity). Figure 1 provides some examples.

Each exercise has a coding window with a partial program, and the student is instructed to complete the program to carry out a particular task, like printing numbers by dividing a variable of value 40 until reaching 1 as shown in Figure 1. The instructions usually include a first test case (sample input and output). The student can only edit the relevant portion of the existing program. When the student presses "Run", the program is executed with various test cases that test for proper program output, and the student is shown which test cases passed and which failed (showing the difference in output for failed cases) as shown in Figure 2. Each exercise is worth 2 points: 1 point for passing any test case, and 1 point for passing all test cases. Students can attempt such exercises as many times as desired. Most universities give students some homework points for completing those coding exercises.

The CA in Figure 1, titled "Basic while loop expression", asks the students to print userNum divided by 2 until reaching 1 (given userNum = 40). The shown code is the template. Students change the comment "/* Your solution goes here */" with their code.

Our institution has taught both Python and C++ in our CS 1, so we wished to compare those languages. We noticed 11 of the CA's were nearly identical in the Python and C++ versions. Those

Figure 1: Each coding activity (CA) includes instructions, an example, a coding area, and a Run button. For this activity, only lines 8 - 10 are editable.

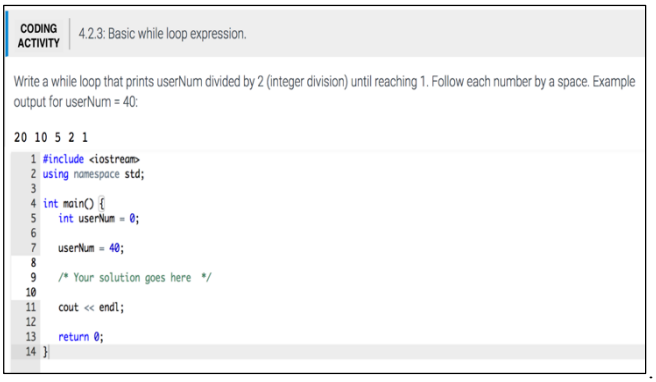
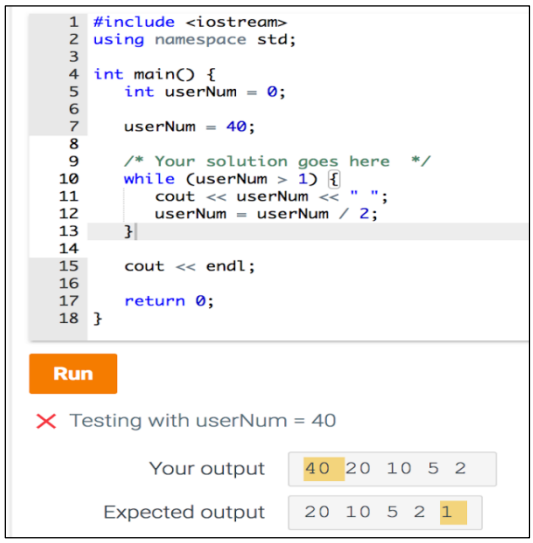


Figure 2: A wrong student solution to the Figure 1 CA. Differences between student and expected output are highlighted.



CA's are summarized in Table 1, numbered as CA 1, 2, …, 11 (which differs from the numbering in the textbooks).

3 STRUGGLE RATE AS A METRIC

To calculate the struggle rate, we first needed to find the number of struggling students for a particular CA. We define struggling students using two parameters: number of attempts (number of submissions), and time spent trying to solve the CA. Students may submit multiple submissions before achieving the correct solution to a CA. We define a struggling student for a particular CA as a student who has spent more than 5 minutes and spent more than double the Baseline time and attempted more than 3 times and attempted more than double the Baseline attempts, or spent more than 15 minutes. The Baseline time is the time spent by the top 20% students in that class for that CA and the Baseline attempts is the number of attempts by the top 20% students in that class for

Table 1: List of the 11 CA's with their titles and chapters.

CA #	CA title	Ch #	Chapter Title
1	Tree Height	2	Vars / Assgnmt
2	Basic while loop expression	4	Loops
3	Simon says	4	Loops
4	Vector iteration: Sum excess	5	Arrays / Vectors
5	Function call in expression	6	User-Def Fcts
6	Function errors: Copying one function to create another	6	User-Def Fcts
7	Function with loop: Shampoo.	6	User-Def Fcts
8	Constructor overloading	7	Objs & Classes
9	Basic inheritance	10	Inheritance
10	Derived class membr override	10	Inheritance
11	Recursive function: Writing the base case	12	Recursion

that CA. We defined a dynamic struggle rate (by referring to top 20% students) rather than a static struggle rate to account for the class (students) background level in programming. A CA's struggle rate is defined as the # of struggling students divided by the # of students in that class. The following formulas summarize a struggling student and the struggle rate.

Struggling student = ((time > 5 min.) AND (time > 2 * Baseline time) AND (# attempts > 3) AND (# attempts > 2 * Baseline attempts)) OR (time > 15 min.)

Struggle rate = # struggling students / # students

Changing the parameters increases or decreases the struggle rate; we based these numbers on teaching experiences. Other struggle rate metrics are possible. For our purposes, the raw % is less important than is the comparison of rates for different languages.

As shown in Table 2, students can make multiple submissions for each CA. Each submission consists of a timestamp, user id #, correctness, and the submitted code. To identify a struggling student for a CA, we do the following: (1) get the student total time spent and the student total number of attempts to solve that CA; (2) calculate the top 20% student average time to solve that CA, called "Baseline time", and the top 20% student average

Table 2: A snapshot of the student 0xxx submissions for CA2: Basic while loop expression.

Time of submission	User #	Answer correct	Submitted solution

3/10/2017 1:20:44 PM	0xxx	No	while (userNum != 1){
			cout << userNum << " ";
			userNum = userNum / 2;
			}

...	0xxx	No	...

3/10/2017 1:24:51 PM	0xxx	No	while (userNum !=0){
			while (userNum != 1){
			cout << userNum << " ";
			userNum = userNum / 2;
			}
			}
			cout << userNum << " ";

...	0xxx	No	...

3/10/2017 1:30:18 PM	0xxx	Yes	while (userNum >= 1){
			cout << userNum << " ";
			userNum = userNum / 2;
			}

number of attempts to solve that CA, called "Baseline attempts"; (3) if the student total time spent is greater than 15 minutes, then the student is a struggling student; or if the student total time spent is greater than 5 minutes and the student total time spent is greater than double the Baseline time and the student total number of attempts is greater than 3 and the student total number of attempts is greater than double the Baseline attempts, then the student is a struggling student. To calculate the struggle rate for a CA, we divide the total number of struggling students for a CA by the total number of students for that CA.

4 RESULTS

We obtained anonymized student submission data for the 11 above-mentioned nearly-identical CAs, for C++ and Python courses at dozens of universities. We chose 11 C++ courses and 10 Python courses at 20 universities to represent a variety of institutions, including 4-year research institutions (none were schools typically ranked in the top 20), non-research 4-year institutions, and 2-year institutions (community colleges). To obtain roughly equal samples from both languages, we generally sought to match each C++ course with a Python course from an institution of the same type and roughly the same numbers of students. Table 3 shows the number of students in the 20 courses per language (C++ and Python). Obviously such matching can't be perfect, but by attempting such matching, coupled with the large numbers of students, we can have more confidence that the two sample populations' statistics can be meaningfully compared.

Table 4 shows the struggle rates for the 11 coding exercises, summarized for all 11 C++ courses and all 10 Python courses. For example, the first data row is for CA 1 (Coding Activity 1). 787

Table 3: Each row is two similar schools using different languages. A Python-match for row 11 does not exist, but we kept the C++ offering to have data for no-prerequisite and non-majors for C++ CA's.

School	Total # students in C++	Total # students in Python
1 (Research universities)	153	105
2 (Community colleges)	13	33
3 (Teaching universities)	34	23
4 (Research universities)	277	176
5 (Same community college)	21	29
6 (Teaching universities)	48	35
7 (Research universities)	121	92
8 (Community colleges)	14	165
9 (Community colleges)	15	17
10 (Research universities)	167	195
11 (Teaching university)	194	N/A
Total number of students	1057	870

C++ students across the 11 C++ courses attempted that CA, while 434 Python students did. Among those students, 5% of the C++ students struggled (39 out of 787), while 9% of the Python students struggled (41 out of 434). Such data is shown for each of the 11 CA's. On average, the C++ struggle rate was 13%, while the Python struggle rate 26%. The average was computed by dividing the total number of students for all the 11 CA's by the total number of struggling students for all the 11 CA's. We did the struggle rate analysis for one community college with similar numbers of students in both C++ (21) and Python (29), and we found nearly identical results with Python students struggling more than C++ students.

The data surprised us, so we further sought to see if perhaps the effect was due to differences in the student populations. We did not have access to information about individual students, so we examined the course descriptions. We considered that perhaps the C++ courses were a second course, following a simpler programming course or some other computing-related course. Thus, we excluded any courses that had a computing-related prerequisite, reducing the set to 4 C++ courses and 7 Python courses. Table 5 shows results. The difference in struggle rates continued: 14% for C++, 28% for Python.

Table 4: A comparison of struggle rates on 11 nearly-identical coding exercises for 11 C++ and 10 Python courses.

	C++			Python		
CA	# students	# struggle	struggle %	# students	# struggle	struggle %
1	787	39	5	434	41	9
2	700	131	19	493	159	32
3	489	130	27	362	179	49
4	513	28	5	259	43	17
5	658	51	8	469	46	10
6	630	14	2	420	28	7
7	482	128	27	414	183	44
8	467	19	4	115	30	26
9	193	38	20	77	19	25
10	181	32	18	69	8	12
11	400	103	26	233	118	51
Total	5500	713	13	3345	854	26

Table 5: Struggle rates considering only courses having no computing-related prerequisites, meaning 4 C++ courses and 7 Python courses.

	C++			Python		
CA	# students	# struggle	struggle %	# students	# struggle	struggle %
1	619	33	5	323	33	10
2	602	121	20	388	142	37
3	410	123	30	266	132	50
4	402	25	6	192	34	18
5	564	46	8	365	40	11
6	544	13	2	326	26	8
7	400	115	28	320	153	48
8	247	14	6	32	5	16
9	10	1	10	7	1	14
10	7	0	0	5	1	20
11	220	56	25	165	93	56
Total	4025	547	14	2389	660	28

We considered that perhaps the C++ courses were taken by majors and Python by non-majors. We thus divided the courses into those intended for majors (per their course descriptions), and those for non-majors. The effect was still seen: 10% vs. 26% struggle for majors (2242 C++ and 383 Python students), and 18% vs. 26% for non-majors (1783 C++ and 1834 Python students).

Because students are anonymized and we have no data on the students themselves, the above analyses should not be considered as perfectly representative of the student populations. For example, the courses intended for non-majors may very well have had some majors. However, the analysis was intended merely to determine if the hypothesis is correct that Python students have a substantially easier learning curve than C++ students. For that purpose, the data seems to suggest that belief is false (and in fact the opposite may be true)

5 ANALYSIS

Table 6: % likelihood that any given student would struggle more with Python than with C++

Table	Students population	p-value	Struggle %
Table 4	All students. 11 C++ and 10 Python courses and each course's submissions has 11 CA's.	< 0.0001	12%
Table 5	No prerequisite. with 4 C++ and 7 Python courses.	< 0.0001	12%
Not shown	No prerequisite and CS majors. 2 C++ and 4 Python courses.	< 0.0001	18%
Not shown	No prerequisite and non CS majors. 2 C++ and 3 Python courses.	< 0.0001	6%

Tables 4 and 5 show Python students struggle more than C++ students. We want to account for the relative number of students per CA because the struggle % is not consistent per CA. For example, in Table 5, CA 8 had 247 C++ students but only 32 Python students. We thus converted the difference of the average C++ and Python Z-scores to a % as follows. For each table, we:

1. Calculated the Z-score per CA: We used the mean and standard deviation of C++ and Python struggle combined per CA.
2. Calculated the p-value for the whole table: We used a Student's t-test to compare the C++ Z-scored struggle to Python Z-scored struggle
3. Calculated the percentage of the average difference: We averaged the C++ Z-scores, and separately averaged the Python Z-scores, then used a Z-score to percentile calculator [7] to convert the difference in average to a percentile.

The final step gives the % likelihood that a given student would struggle more with Python than C++. Table 6 shows that a student is 12% more likely to struggle with Python than C++.

6 MANUAL INVESTIGATION

The analysis above suggests that the Python learning curve, based on the metric of struggle rate on small coding exercises, is not easier than the C++ learning curve. In fact, the analysis suggests (perhaps surprisingly) that the learning curve is actually harder. To better understand, we manually examined student submissions to many of the CA's. Because manual examination is very time consuming, we examined CA's 1, 2, 7, and 10, seeking to spread out the CA's examined.

Table 7 illustrates our findings. For example, for CA 1, which involved converting a math equation into an assignment statement, Python students struggled more with basic assignment concepts, such as missing an = operator, confusing left and right sides, or assigning to an expression rather than a variable. For CA 2, which involved writing a while loop to output a number halved until reaching 0, Python students struggled more on all aspects of the problem, including writing the loop condition, updating the loop counter variable, or placing the output in the correct location.

7 DISCUSSION

Limitations: Other struggle metrics exist, such as measuring struggle on weekly programming assignments, surveying students, measuring performance on exams, etc.

Our analysis involved 1,927 students at courses across 20 universities. While those large numbers and the diversity of populations are strengths of the data and likely minimizes the impact of one particular course's policies or instructor's teaching style, also useful would be a controlled study at one university (which is hard to carry out, since such random assignment is rarely acceptable), or where the university switched from one language to another across semesters (but other factors like teacher and student population may confound results).

The Python/C++ textbooks use a standard approach. Other approaches, such as a media-based approach or objects-first ordering, may yield different results.

The perceived easier learning curve is just one reason some teachers have switched to Python. Other reasons exist, such as built-in libraries. Thus, the above data relates to just one factor among many that influence a CS 1 language decision.

Possible reasons: This study analyzed struggle rate, not the reasons. One possible reason for Python's struggle rate not being lower than for C++ is that learning core programming concepts may overshadow syntax issues. The manual investigation of student submissions seemed to support this reason; few students struggled with syntax in either language. Instead, struggle was due to programming concepts like creating a proper loop to solve a task. The case may be that college students can master the basic syntax of C++ nearly as quickly as they master the slightly-easier syntax of Python. Also, C++ teachers can choose whether or not to dwell on C++ syntax. The textbook in this study avoids potentially-problematic aspects of C++, such as branches/loops without braces (the book always uses braces), assignments in branch/loop expressions (the book avoids those), use of prefix/postfix increment operators (the book avoids except in a for-loop header), etc., instead teaching a common and safer subset of C++.

Python's struggle rate was surprisingly higher. One possible reason relates to programming requiring precision. From the beginning, C++ requires precise thought about variable declarations, variable types, data types resulting from expressions, use of braces, use of = vs. ==, etc. This precision may prime students to think more precisely about language-independent problem-solving as well, like writing loop expressions that iterate exactly as desired. Python's forgivingness might breed a more cavalier attitude that extends beyond syntax/semantics into problem solving as well. This of course is just conjecture; future work may seek to test the idea.

Table 7: CA1, 2, 7, and 10 for all data and the reasons why students struggled, as determined by manual investigation.

CA	C++	Python
	Reasons	Reasons
1	1-Using / instead of * 2-Missing tan() for the angleElevation variable 3-Mistyping variable names	1-Using tan() instead of math.tan() 2-Using / instead of * 3-Missing tan() for the angleElevation variable 3-Mistyping variable names 4-Wrong assignment (using two = symbols, assign to the wrong variable, reverse assignment, etc.)
2	1-Wrong loop condition 2-Wrong/missing loop counter update 3-Missing/wrong location output stmt	1-Wrong loop condition 2-Wrong/missing loop counter update 3-Missing/wrong location output statement 4-Indentation (few: just 5 students)
7	1-Missing for-loop 2-Missing counter inside the for-loop 3-Wrong for-loop counter initial value 4-Wrong for-loop condition 5-Wrong for-loop location 6-Wrong cout() arg inside for-loop	1-Missing for-loop 2-Wrong while-loop update (when using while-loop) 3-Wrong for-loop condition 4-Wrong print() argument inside the for-loop 5-Missing for-loop condition variable inside the for-loop 6-Wrong for-loop location
10	1-Missing ; 2-Wrong cout() argument 3-Not complete cout() arguments 4-Wrong location to call a member function 5-Missing to call a member function 6-Wrong format to call a member function 7-Missing function def	1-Missing function definition 2-Missing call to member function 3-Missing argument to call a member function 4-Extra space when calling print() 5-Wrong call to a member function 6-Wrong print() statement 7-Wrong function argument

We note that cloud-based programming is reducing the difference between languages, eliminating (or postponing) the need to install or even use an IDE.

8 CONCLUSIONS

One factor leading teachers to use Python in CS 1 courses is the belief that Python has an easier learning curve. We analyzed struggle rates for 11 nearly-identical short coding exercises in 11 C++ and 10 Python courses, involving about 1,000 students in each language at 20 universities. We found the Python struggle rate was not lower than C++. One possible reason is that the languages' syntax differences are eclipsed by the difficulty of learning language-independent programming concepts, especially if C++ teachers don't dwell on C++'s complex syntax options.

In fact, our analysis showed Python's struggle rate to be significantly higher than C++. One possible reason is that C++'s focus on precision translates to a more precise approach to programming. As for attrition, at our institution, we have found that a caring talented instructor with good class design, policies, and assignments -- appropriate homework/assignment points ratio, various help resources, encouragement of collaboration, flipped lectures, interesting/relevant assignments -- seem far more important than the language choice. In fact, in our most recent offering of CS 1 in C++, students provided evaluations in the 95'th percentile for all courses in the university of 30,000 students, while performing strongly on programming assignments and exams.

In any case, the analysis might help CS 1 teachers predict whether switching from C++ (or C or Java) to Python might yield the desired benefit of an easier learning curve. We encourage the community to perform more such analyses, so that teachers can be guided by data in making language decisions for CS 1 courses.

REFERENCES

[1] Richard J. Enbody, William F. Punch, and Mark McCullen. 2009. Python CS1 as preparation for C++ CS2. *ACM SIGCSE* Bulletin 41, no. 1 (2009): 116-120.

[2] Richard J. Enbody, and William F. Punch. 2010. Performance of python CS1 students in mid-level non-python CS courses. *In Proceedings of the 41st ACM technical symposium on Computer science education*, pp. 520-523. ACM, 2010.

[3] Michael H. Goldwasser, and David Letscher. 2008. Teaching an object-oriented CS1-: with Python. *ACM SIGCSE* bulletin, vol. 40, no. 3, pp. 42-46. ACM, 2008.

[4] Lutz Prechelt. 2003. Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. *Advances in Computers 57* (2003): 205-270.

[5] John M. Zelle. 1999. Python as a first language. *In Proceedings of 13th Annual Midwest Computer Conference*, vol. 2, p. 145. 1999.

[6] Philip Guo. 2014. Python is now the most popular introductory teaching language at top us universities. *BLOG@ CACM*, July (2014): 47.

[7] Measuring U. Z-Score to Percentile Calculator. https://measuringu.com/pcalcz/, accessed Aug, 2017.

[8] zyBooks. https://www.zybooks.com/, accessed Aug, 2017.