Volume 15, No. 3
May 2017
ISSN: 1545-679X

INFORMATION SYSTEMS EDUCATION JOURNAL

In this issue:

The **Information Systems Education Journal** (ISEDJ) is a double-blind peer-reviewed academic journal published by **EDSIG**, the Education Special Interest Group of AITP, the Association of Information Technology Professionals (Chicago, Illinois). Publishing frequency is six times per year. The first year of publication was 2003.

ISEDJ is published online (http://isedj.org). Our sister publication, the Proceedings of EDSIGCon (http://www.edsigcon.org) features all papers, panels, workshops, and presentations from the conference.

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the conference. At that point papers are divided into award papers (top 15%), other journal papers (top 30%), unsettled papers, and non-journal papers. The unsettled papers are subjected to a second round of blind peer review to establish whether they will be accepted to the journal or not. Those papers that are deemed of sufficient quality are accepted for publication in the ISEDJ journal. Currently the target acceptance rate for the journal is under 40%.

Information Systems Education Journal is pleased to be listed in the 1st Edition of Cabell's Directory of Publishing Opportunities in Educational Technology and Library Science, in both the electronic and printed editions. Questions should be addressed to the editor at editor@isedj.org or the publisher at publisher@isedj.org. Special thanks to members of AITP-EDSIG who perform the editorial and review processes for ISEDJ.

INFORMATION SYSTEMS EDUCATION JOURNAL

Editors

2017 ISEDJ Editorial Board

Java vs. Python Coverage of Introductory Programming Concepts: A Textbook Analysis

Kirby McMaster
kmcmaster@weber.edu
Computer Science - retired
Weber State University
Ogden, UT 84408 USA

Samuel Sambasivam
ssambasivam@apu.edu
Computer Science
Azusa Pacific University
Azusa, CA 91702 USA

Brian Rague
brague@weber.edu
Computer Science
Weber State University
Ogden, UT USA 84408

Stuart Wolthuis
stuartlw@byuh.edu
Computer and Information Sciences
Brigham Young University-Hawaii
Laie, HI 96762 USA

Abstract

In this research, we compare two languages, Java and Python, by performing a content analysis of words in textbooks that describe important programming concepts. Our goal is to determine which language has better textbook support for teaching introductory programming courses. We used the TextSTAT program to count how often our list of concept words appear in a sample of Java and Python textbooks. We summarize and compare the results, leading to several conclusions that relate to the choice of language for a CS0 or CS1 course.

Keywords: programming concepts, Java, Python, textbooks.

1. INTRODUCTION

In the early years of computing, the choice of a first language for programmers was often decided by the work environment, typically Information Technology divisions with specialized needs. Assembly language for a specific hardware system was the usual situation. Programming in a higher-level language such as Fortran or Cobol became common over time as more versatile computing platforms and elaborate computing problems emerged.

When Computer Science programs at universities began to develop, the choice of an introductory programming language was determined primarily by the curriculum designers, with an emphasis on the pedagogical value of the language rather than its popularity or practicality in developing real-world applications. As might be expected in the academic world, there was and still is a diversity of opinion on what the first language should be (Siegfried, Chays, & Herbert, 2008).

The most recent Computer Science Curriculum Guidelines (2013) published by ACM/IEEE state that "...advances in the field have led to an even more diverse set of approaches in introductory courses [and these] approaches employed in introductory courses are in a greater state of flux." Moreover, the report observes "...that rather than a particular paradigm or language coming to be favored over time, the past decade has only broadened the list of programming languages now successfully used in introductory courses".

In the 1970s and 1980s, Pascal became the language taught most often in introductory programming courses. Eventually, many schools moved to C for practical reasons, since graduates rarely used Pascal in their employment. As the benefits of object-oriented programming became evident, the first language evolved to C++ and later to Java, which provides a more managed development environment (de Raadt, Watson, & Tolman, 2002).

The tradeoffs of an object-first approach versus an imperative-first approach in introductory courses have been extensively and hotly debated (Lister, 2006). This decision about which programming paradigm to teach beginning students strongly influences the choice of introductory language. Alternatively, some early courses in CS emphasized broader computing concepts rather than the subtleties of programming syntax (Sooriamurthis, 2010). The paramount question regarding the delivery of an effective introductory CS course remains "What to teach?", followed immediately by "Which language best supports the concepts to be taught?".

In recent years, the increased demand for programming courses for liberal arts students has led to the development of what are termed CS0 courses (with CS1 courses aimed for CS majors). The preferred programming language for a CS0 course is often different from the language taught in CS1. CS0 languages trend toward predominantly visual environments such as Alice, or more dynamic popular choices such as Python.

Purpose of this Research

Much research has been performed over the last few decades on which language is best for an introductory programming course (Brilliant & Wiseman, 1996). In an effort to contribute to this discussion, our research focuses on two languages--Java and Python. These languages are increasing in popularity for introductory courses, especially Python (Guo, 2014). Rather than evaluate the usability or suitability of the languages within an introductory context, we performed a content analysis (Krippendorff, 2012) of Java and Python textbooks to determine how well they cover important CS0/CS1 programming concepts such as *class* and *algorithm*.

We developed a list of basic programming concepts that might be taught in an introductory course. Initial sources used for developing this concepts list were drawn from various instructional assessments, curriculum resources, and introductory course content that we designed ourselves or researched. We then counted how often each textbook mentioned each concept. We did not study the order in which the concepts were presented, nor did we judge how well the concepts were explained. We simply summarized frequencies for the words that represented each concept.

An instructor in a programming course usually chooses a textbook to guide how she/he will organize and present the material. Our main research assumption is that the framework of the author is reflected by the words used most often in the textbook. The framework we are evaluating is one that is appropriate for introductory programming. From the author's choice of words, we can judge how suitable the textbook will be for teaching the main concepts of the programming course.

2. METHODOLOGY

This section of the paper describes the methodology used to collect word frequency data from selected Java and Python textbooks. The words we examine represent important concepts for an introductory programming course.

Programming Concepts

We created a list of important programming concepts from several sources. We started with an initial list of programming terms taken from quizzes and exams we have given to CS1

students to measure their understanding of course topics. In earlier research, we performed a word frequency analysis of object-oriented programming (OOP) textbooks (representing a variety of languages) to empirically reveal frequent OOP concepts. We used the results of that study to form a list of OOP words.
In the current study, we created a list consisting of programming concepts mentioned in the Programming Fundamentals (PF) section of the Computing Curricula 2001 Computer Science Final Report (2001). We created an additional list of concepts based on the Software Development Fundamentals (SDF) section of the Computer Science Curricula 2013 Final Report.

In constructing our list, we attempted to avoid keywords from specific languages, such as *float* and *while*. However, a few keywords, such as *class*, were difficult to omit. From the above sources, we formed a combined list that grew to 100 programming concepts. This larger list evolved as we performed the actual word analysis in the selected Java and Python books.

Sample of Textbooks
We collected a sample of 10 Java textbooks and 10 Python textbooks. We wanted our sample to include popular books in both languages. Due to budget constraints (i.e. no budget), we chose textbooks that were available on the Internet and could be downloaded as PDF files. We obtained a reasonably diverse sample of books (see References), but some were older editions (e.g Zelle, 2002).

We later observed that the Java books tended to be larger (i.e. contained more words). The average size of the Java books was 222,953 words, whereas the average size for the Python books was 144,039 words. As a quick check to confirm that the sizes of our Java and Python books were representative, we compared 10 Java books and 9 Python books (not including very short books) listed on Amazon. For the Amazon books, the total number of words was not available, but the number of pages was given. The Amazon sample averages were 690 pages for the Java books and 514 pages for the Python books. So on Amazon, the Java books tend to be larger, which is consistent with our downloaded sample.

Convert PDF files to Text Files
Textbooks in PDF file format are not convenient for performing repeated word searching and counting. Fortunately, Adobe Reader has a "File/save As" menu choice to convert the contents of a PDF file to a text file. We used Adobe Reader to create a text file for each of the 20 textbooks in our study.

We noticed that the text file versions of the books included many character strings that contained digits, punctuation, and other non-alphabetic symbols. To simplify our counting of concept words, we wrote a short program (in Python) that removed all non-letter symbols and replaced them with blank characters. This program also converted all letters to lower-case. We used this program to obtain a filtered set of 20 text files which consisted of only letters and blanks. Note that none of the targeted word groups contains a numeric or special character.

Perform Word Counts
We used a popular program called TextSTAT (Huning, 2007) to obtain word counts for all words on our programming concept list. With TextSTAT, you first define a "Corpus", which holds a list of text files. We defined a corpus for each textbook and linked the corpus to the transformed textfile containing the textbook.

To perform a word search, a separate TextSTAT screen allows the user to specify search options. Most of the time, we used the option to include all words, with the words and frequencies presented in alphabetical order. We would then go through the concept list (also in alphabetical order) and record/total the frequencies for each word group. This was the most labor-intensive part of our methodology. Occasionally, we would enter a short string (e.g. *iterat*) to search for all words that contain the string (e.g. *iterate*, *iteration*, *iterator*).

3. ANALYSIS OF DATA

The number of programming concepts on our evolving list reached 100 by the end of our data analysis. Alphabetically, the concepts ranged from *abstraction* to *variable*. As mentioned in the methodology section, each concept was represented by a group of one or more words. For example, the word group for the OOP concept *object* contained two words--*object* (singular) and *objects* (plural).

For every concept, we counted the number of occurrences of each word group member in the Java and Python textbooks. As an example, in the Java book by Schildt (2007), the word *object* appears 1674 times, and the word *objects* appears 380 times. The total word count for the concept is 2054.

Convert Word Counts to Word Rates
Because each textbook contains a different number of words, the actual word counts for concepts are not comparable across books. Larger books tend to have larger word counts. To standardize the counts, we converted each word count for a concept to a *word rate*. The rate we chose was "per 100,000 words". That is, we divided the concept word count by the total number of words in the book and multiplied by 100,000.

For example, Schildt's book mentioned above contains a total of 325,991 words. The word count for the *object* concept is 2054. This count is rescaled to a word rate as shown below:

word rate = (2054/325,991)*100,000 = 630.1

This means that the object concept is mentioned 630.1 times per 100,000 words in Schildt's book. Word rates were calculated for each concept in each book.

Calculate Trimmed Means
After concept word rates were obtained in all Java and Python textbooks, averages were calculated separately for the Java and Python values. Because the word rates for concepts (Java or Python) often varied widely from book to book, we calculated trimmed means (instead of the usual untrimmed versions) to diminish the effect of outliers. To provide a conservative treatment for these outliers, our trimmed means include only the middle 6 out of 10 word rates. The top two and bottom two word rates are dropped.

For example, word rates for the *object* concept in all 10 Java textbooks are:

522.4 561.7 630.1 334.5 843.3
684.9 703.5 767.2 863.5 488.4

Removing the two highest rates (863.5 and 843.3) and two lowest rates (334.5 and 488.4), the trimmed mean for *object* in the Java books is 645.0. Two trimmed means were calculated for each concept, one for Java and the other for Python.

Distributions of Trimmed Means
Each set of books (Java and Python) provided a sample of 100 trimmed means, representing word rates for the 100 concepts. A statistical description of the Java and Python distributions is summarized in Table 1.

Many of the statistics are larger for the Java distribution than the Python distribution. The central tendency measures (mean and median) are higher, and the dispersion measure (IQR) is larger. This is primarily due to the greater number of concept words in the Java books.

Statistic	Java	Python
Sample N	100	100
Minimum	0.34	0.00
Centile 25	18.92	10.50
Median	58.00	38.05
Centile 75	134.27	116.68
Maximum	987.40	601.93
IQR	115.35	106.18
Mean	109.95	90.59

Table 1: Distributions of Trimmed Means

For the Java distribution, the *maximum* word rate is for the concept *class*, and the *minimum* word rate is for *decomposition*. For Python, the *maximum* word rate is for *function*, while the *minimum* word rate is (again) for *decomposition*. The Java *median* word rate is the midpoint between the word rates of the two middle concepts *stream* and *block*. For Python, the two middle concepts are *block* and *event*.

The *mean* of the Java word rates is almost twice the size of the median. This indicates that the distribution is positively *skewed*, mainly due to the presence of several high word rates (including the maximum value). The mean of the Python word rates is more than twice the size of the median, indicating another positively *skewed* distribution.

The variability of scores in a distribution is usually described by the *standard deviation*. However, this statistic is inflated when outliers are present. A more stable measure of variation is the interquartile range IQR (Upton & Cook, 1996), which is the difference between the 75th centile value and the 25th centile value. For Java, the 75th centile concept is *definition*, and the 25th centile concept is *link*. The corresponding concepts for Python are *set* (75th centile) and *literal* (25th centile).

The word rates for programming concepts tend to be higher in the Java books. Overall, 62 of the 100 concepts have a higher word rate in the Java books than in the Python books. The remaining 38 concepts appear more often in the Python books. Additional details and comparisons of these two word rate distributions are presented in the following sections.

Most Frequent Concepts

The fifteen programming concepts with the highest word rates for Java and Python are listed in Table 2.

Java Concept	Rate	Python Concept	Rate
class	987.4	**function**	601.9
method	949.8	**list**	487.0
object	645.0	program	462.1
value	477.5	value	451.1
program	460.6	string	410.4
string	399.8	file	372.0
type	369.5	object	336.7
variable	288.6	number	319.7
array	272.2	code	300.6
system	253.7	method	298.9
number	251.4	class	297.0
file	216.9	**line**	263.7
code	213.2	**module**	235.8
statement	212.1	type	204.0
thread	188.2	statement	203.1

Table 2: Most Frequent Concepts
(Differences in **bold**)

Eleven of the concepts appear on both lists, but in different ranked positions. This demonstrates substantial agreement by authors on which concepts are *most important* in both languages. Four concepts are on the Java list only, and four others are confined to the Python list. The concepts that are not on both lists are shown in **bold**.

Among the Java concepts, the top three--class, method, and object--describe features of object-oriented programming (OOP). These concepts are also on the Python list, but with lower word rates. Six of the Java concepts--value, string, type, variable, array, and number--describe data types and data structures. The Python list contains four of these concepts, but replaces array with list and excludes variable.

The I/O concept *file* is on both lists, but has a higher word rate in the Python books. The Java concept *thread* is rarely mentioned in the Python texts. *Function* and *module* are older terms used to describe modular programming. Python retains these terms, whereas the Java books prefer the OOP concepts *method* and *class*.

Least Frequent Concepts

The fifteen programming concepts with the lowest word rates for Java and Python are listed in Table 3. Again, eleven of the concepts appear on both lists, but in different ranked positions. This shows agreement by Java and Python authors on concepts they perceive to be *unimportant* in both languages. Concepts that appear on only one list are shown in **bold**.

Java Concept	Rate	Python Concept	Rate
encapsulation	9.3	**constant**	6.6
debug	8.1	maintainable	5.8
signature	7.9	**stream**	5.1
record	7.9	encapsulation	4.0
maintainable	7.1	reserved	3.9
abstraction	5.9	branch	3.1
polymorphism	5.5	pointer	2.8
relation	5.4	polymorphism	2.5
reserved	5.1	procedure	1.6
procedure	4.7	signature	1.5
pointer	4.2	quality	1.5
branch	3.3	**queue**	0.6
module	1.3	**thread**	0.6
quality	0.6	abstraction	0.6
decomposition	0.3	decomposition	0.0

Table 3: Least Frequent Concepts
(Differences in **bold**)

The concepts that appear on both least-frequent lists include a few surprises. Some of these concepts are often considered important by programming instructors. Certainly *abstraction* is a key programming topic. Of the three pillars of OOP (*encapsulation*, *inheritance*, and *polymorphism*), two are on both least-frequent lists. Thankfully, these textbooks spare *inheritance* from such neglect. The *signature* concept, relevant to *polymorphism*, is rarely mentioned.

Function and *procedure* were once distinct concepts in modular programming. Perhaps due to compromises made in the design of the C language (and perpetuated in C++ and Java), the *procedure* word has been replaced with "void" functions.

From the Software Engineering (SE) vocabulary, *quality* and *maintainable* are held in low regard by both Java and Python textbooks. The concept of *pointer* has low word rates, although the substitute term *reference* does appear more often in both sets of books. *Keyword* is more popular than *reserved* word. Finally, almost none of the books contain *decomposition*, which is the least frequent word on both lists. This concept embodies a core strategy in modular programming.

Middle Frequency Concepts

We have presented word rates for the top 15 and bottom 15 programming concepts, and now turn our attention to the 70 concepts with middle-level usage rates. This list of concepts is too long to include in a single table in the paper. Instead, in Table 4 we present 10 Software Engineering concepts that have middle-level word rates in the programming textbooks.

	Java	Python
Concept	**Rate**	**Rate**
problem	63.9	57.9
solution	32.1	48.1
requirement	29.9	42.8
specification	55.5	39.5
model	25.1	13.6
algorithm	34.9	22.5
design	49.2	12.3
test	85.5	138.2
style	21.1	17.7
document	40.5	44.0

Table 4: Middle Frequency Concepts
Software Engineering Words

For Java books, the SE word rates range from 21.1 (for *style*) to 85.5 (for *test*). The word rates in Python books range from 12.3 (for *design*) to 138.2 (again for *test*).

Concepts on the list include *problem* (Java/Python rates 63.9/57.9) and *solution* (Java/Python rates 32.1/48.1), reflecting the problem-solving focus in SE. The words *requirement*, *specification*, *model*, *algorithm*, *design*, and *document* are life cycle development activities. *Style* is a consideration to ensure source code is readable and maintainable. The relatively low word rates for *style* (Java/Python rates 21.1/17.7) and for *model* (Java/Python rates 25.1/13.6) are unfortunate.

As Table 4 indicates, all of these concepts appear with moderate word rates in both the Java and Python textbooks. Six of the concepts appear more often in Java books, while the other four are more frequent in Python books. There is no obvious single criterion for determining which language favors which SE concepts.

Word Rate Correlation

In this section, instead of examining the Java and Python word rate distributions separately, we consider the joint distribution of the two rates. If the focus on key introductory concepts is consistent across all examined textbooks, we would expect to find a positive relationship between the Java and Python word rates. For most programming concepts, a higher word rate in the Java books should suggest a higher word rate in the Python books, and vice versa.

To measure the degree of linearity in the relationship, we calculated the Pearson correlation coefficient. The correlation value we obtained for our 100 pairs of scores was 0.601, which is positive but far from 1.0.

We do not claim that the relationship should be linear, but it should be monotonic. A better statistic for monotonic relationships is the Spearman rank-order correlation (Maritz, 1995). Our result for the Spearman statistic was 0.726, which describes a fairly strong *increasing* relationship between Java and Python word *ranks*.

A scatter diagram of the word rate pairs, converted to ranks from 1 (highest rank) to 100 (lowest rank), is displayed as Figure 1.

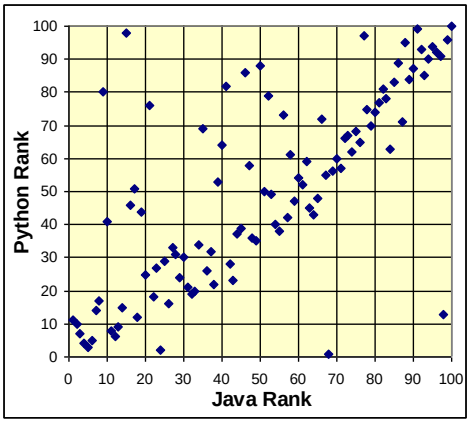

Figure 1: Java vs. Python Concept Ranks

In this figure, we can see that most of the pairs of ranks fall approximately along a line that runs from pair (1,1) to pair (100,100). Below the implied line, two obvious outliers are the pairs (98,13) for *module* and (68,1) for *function*. In these pairs, the Python rank is much higher (closer to 1) than the Java rank. Above the line, the two most noticeable outliers are (15,98) for *thread* and (9,80) for *array*. These concepts have a much higher Java rank (closer to 1).

A more complete list of outliers is presented in Table 5.

Concept	Java Rank	Python Rank	Diff
module	98	13	-85
function	68	1	-67
interface	16	46	30
system	10	41	31
component	35	69	34
event	17	51	34
stream	50	88	38
constant	46	86	40
declaration	41	82	41
constructor	21	76	55
array	9	80	71
thread	15	98	83

Table 5: Largest Differences in Ranks
("Highest" rank is 1)

The choice of how large the difference in ranks should be to consider a concept an outlier is subjective. In this table, we include all pairs in which the difference in ranks is 30 or larger. A negative difference occurs when Python has a higher rank. A positive difference favors Java. Note that all but two of the concepts in Table 5 have a higher Java rank.

We noted earlier that *function* and *module* are among the top fifteen concepts in word frequency in Python books. This table indicates that these two popular Python concepts appear much less often in Java books. Three OOP concepts--*constructor*, *component*, and *interface*--are favored by Java books.

The data concepts *array*, *declaration*, and *constant* appear less often in Python books for various reasons. Python prefers *lists* over *arrays*. Variables are not overtly *declared* in Python. *Stream* I/O, as a generalization of file I/O, is implemented in Java as stream classes. Real-time *events* and *threads* are common Java features, but not Python.

4. SUMMARY AND CONCLUSIONS

The choice of programming language for introductory Computer Science courses is a strong indicator of the concepts emphasized during course instruction. Ongoing discussion about what to teach and which language tool best supports learning objectives for introductory programming courses continues unabated among instructors, administrators, and accreditation organizations. A definitive "best practices" approach in this area remains unresolved. Our current work further informs this debate by correlating core programming concepts with specific textbooks that promote either Java or Python as the coding language.

The primary purpose of this study was to compare how well Java and Python textbooks provide coverage of important introductory programming topics. We developed a list of 100 programming concepts, and we collected a sample of 10 Java books and 10 Python books. We then counted how often words that represent the concepts appeared in the books. After standardizing the data, we computed trimmed means of word rates for all 100 concepts, with separate rates for Java and Python. From this data, we draw the following conclusions.

First, words that describe our 100 programming concepts have a greater density (higher word rates) in the Java books in our study. The word rate distribution for Java has a mean of 109.25, with a maximum value of 987.40. For Python, the mean is 90.59, with a maximum of 601.93.

Second, there is remarkable agreement between the programming concepts mentioned most often in the Java and Python books. Eleven of the top 15 Java concepts are also included in the top 15 Python concepts. Highly-used concepts for both languages include *class*, *object*, and *method*, each representing OOP.

Third, there is also agreement on which concepts are rarely mentioned in both sets of books. Eleven of the bottom 15 Java concepts are also in the list of 15 least-used Python concepts. Common neglected concepts include *encapsulation* and *polymorphism* for OOP, plus SE concepts *quality* and *maintainable*. It is disappointing that *abstraction* is on both bottom 15 lists.

Fourth, several concepts appear on only one of the top 15 or bottom 15 word lists for Java and Python. The top 15 Java-only concepts include *array* and *variable*. Among the top 15 Python-only concepts, *array* is replaced by *list*, and other concepts are added. The bottom 15 Java concepts include *module*, which is a top 15 concept for Python. The bottom 15 Python list includes *thread*, which is a top 15 concept for Java.

Fifth, a fairly strong increasing relationship exists between concept ranks for Java vs. Python, as indicated by a rank-order correlation of 0.726. There are a few clear exceptions to this relationship. *Thread*, *constructor*, and *declaration* have much higher Java ranks. *Module* and *function* have much higher Python ranks.

Sixth, Java and Python textbooks devote substantial time on practical concepts that describe how to write code. Discussion of Software Engineering concepts that deal with how to *think* like a programmer and write efficient, maintainable code receive less attention. This learning goal may be less important in an introductory programming course, but it becomes a major focus as students progress through a Computer Science degree program.

Overall, both Java and Python books provide reasonable levels of support for most of the programming concepts we considered. The choice of Java or Python (or other language) for an introductory class should be based on considerations beyond textbook support for important concepts. Whatever language and textbook are chosen, instructors must be prepared to provide additional material to achieve their desired course objectives.

Future Research
Planned future research activities include:

1. Perform a similar study comparing Java and C++ textbooks to determine how well they support important CS1 concepts.

2. Perform a similar study comparing textbooks for Python and another language (e.g. Ruby) to determine how well they support important CS0 concepts.

3. Perform research to provide empirical support to improve our list of important programming concepts. This is not a trivial task, in light of previous research by Hertz (2010) and Tew & Guzdial (2010).

Note: A complete list of our 100 programming concepts, along with Java and Python trimmed mean word rates, are presented in Table 6 in the APPENDIX.

5. REFERENCES

Brilliant, S. S., and Wiseman, T., "The First Programming Paradigm and Language Dilemma", ACM SIGCSE Bulletin Vol. 28, No. 1 (1996), p. 338-342.

Computing Curricula 2001 Computer Science Final Report, Joint Task Force on Computing Curricula, Association of Computing Machinery, IEEE Computer Society, 2001.

Computer Science Curricula 2013, Joint Task Force on Computing Curricula, Association of Computing Machinery, IEEE Computer Society, 2013.

deRaadt, Michael, Watson, Richard, and Toleman, Mark, "Language Trends in Introductory Programming Courses," InSITE, June 2002. proceedings.informingscience.org /IS2002Proceedings/papers /deRaa136Langu.pdf

Guo, Philip, "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities." Communications of the ACM, Blogs, 2014.

Hertz, Matthew, "What do 'CS1' and 'CS2' Mean? Investigating Differences in the Early Courses." SIGCSE Proceesings, Milwaukee, 2010.

Huning, M, TextSTAT 2.7 User's Guide. TextSTAT, created by Gena Bennett, 2007.

Krippendorff, Klaus H., Content Analysis: An Introduction to Its Methodology, 3rd Ed. SAGE Publications, 2012.

Lister, Raymond, E. A. Research perspectives on the objects-early debate. In ITiCSE proceedings (2006), pp. 146--165.

Maritz, J. S., Distribution-Free Statistical Methods (2nd ed). Chapman and Hall, 1995.

Siegfried, Robert M., Chays, David, and Herbert, Katherine G., "Will There Ever be Consensus on CS1?" In Proceedings of FECS. 2008, 18-23. home.adelphi.edu/~siegfried /Consensus.pdf

Sooriamurthi, Raja, The Essence Of Object Orientation For CS0: Concepts Without Code. Journal of Computing Sciences in Colleges, Vol. 25 (3), p 67-74, January, 2010.

Tew, Allison Elliott, & Guzdial, M., Developing a Validated Assessment of Fundamental CS1Concepts, SIGCSE Proceedings, Milwaukee, 2010.

Upton, Graham, and Cook, Ian, Understanding Statistics. Oxford University Press, 1996, p.55.

Java Textbooks:

Arnold, Ken, James Gosling, and David Holmes, THE Java Programming Language (4th ed).Addison Wesley Professional, 2005.

Deitel, Harvey, and Paul Deitel, Java How to Program (4th ed). Prentice Hall, 2002.

Downey, Allen B., Think Java: How to Think Like a Computer Scientist. Allen Downey, 2012.

Eck, David J., Introduction to Programming Using Java (Version 6.0.3). Hobart and William College, 2014 (PDF version of on-line book).

Lemay, Laura, and Charles L. Perkins, Teach Yourself JAVA in 21 Days. Sams.net Publishing, 1996.

Roberts, Eric. S., The Art and Science of Java (Preliminary Draft). Stanford University, 2006.

Schildt, Herbert, Java: The Complete Reference (7th ed). McGraw-Hill, 2007.

Sierra, Kathy, and Bert Bates, Head First Java (2nd ed). O'Reilly.

Stein, Lynn Andrea, Interactive Programming in Java. Lynn Andrea Stein, 1999.

Wu, C. Thomas, An Introduction to Object-Oriented Programming with Java (5th ed). McGraw-Hill, 2010.

Python Textbooks:

Downey, Allen, Think Python: How to Think Like a Computer Scientist (Version 2.0.15). Green Tea Press, 2015.

Halterman, Richard L., Learning to Program with Python. Richard L. Halterman, 2011.

Heinold, Brian, Introduction to Programming Using Python. Brian Heinold, 2012.

Jackson, Cody, Learning to Program Using Python. Cody Jackson, 2011.

Kuhlman, Dave, A Python Book: Beginning Python, Advanced Python, and Python Exercises. Dave Kuhlman, 2009.

Lutz, Mark, Programming Python (4th ed). O'Reilly, 2011.

Maruch, Stef, and Aahz Maruch, Python for Dummies. Wiley, 2006.

Payne, James, Beginning Python: Using Python 2.6 and Python 3.1. Wiley Publishing, 2010.

Pilgrim, Mark, Dive Into Python. Mark Pilgrim, 2004.

Zelle, John M., Python Programming: An Introduction to Computer Science (Version 1). Wartburg College, 2002.

APPENDIX
Table 6: Concept Word Rate Trimmed Means for Java and Python

	Concept	Java Rate	Python Rate			Concept	Java Rate	Python Rate
1	abstraction	5.9	0.6		51	literal	14.0	10.5
2	algorithm	34.9	22.5		52	local	36.2	36.0
3	argument	114.4	142.7		53	loop/looping	112.6	152.5
4	array	272.2	7.8		54	maintain/maintainable	7.1	5.8
5	assignment/assign	53.7	55.8		55	method	949.8	298.9
6	block	56.9	38.4		56	model/modeling	25.1	13.6
7	boolean	82.0	19.8		57	module	1.3	235.8
8	branch/branching	3.3	3.1		58	nest/nested	23.0	22.4
9	case	127.0	81.0		59	number/numeric	251.4	319.7
10	character	120.0	119.6		60	object	645.0	336.7
11	class	987.4	297.0		61	operation/operator	139.1	157.7
12	code	213.2	300.6		62	output	106.8	80.0
13	component	100.4	17.2		63	parameter	92.7	84.0
14	condition/conditional	49.1	53.1		64	pattern	37.1	32.5
15	constant	63.1	6.6		65	pointer	4.2	2.8
16	constructor	141.1	9.9		66	polymorphism	5.5	2.5
17	control	61.7	22.7		67	problem	63.9	57.9
18	correct/correctness	21.2	18.1		68	procedure	4.7	1.6
19	data	133.5	175.5		69	process/processing	61.7	74.0
20	debug/debugging	8.1	15.0		70	program	460.6	462.1
21	declaration/declare	80.9	7.6		71	quality	0.6	1.5
22	decomposition/decompose	0.3	0.0		72	queue	16.1	0.6
23	definition/define	134.3	95.1		73	record	7.9	6.9
24	design	49.2	12.3		74	recursion/recursive	25.0	28.0
25	development/develop	23.9	27.5		75	reference	84.2	34.4
26	documentation/document	40.5	44.0		76	relation/relational	5.4	6.6
27	dynamic/dynamically	9.3	7.6		77	requirement/require	29.9	42.8
28	efficient/efficiency	12.7	9.9		78	reserved	5.1	3.9
29	encapsulation/encapsulate	9.3	4.0		79	scope	12.5	7.7
30	error	77.9	102.9		80	selection	13.1	10.9
31	event	152.8	37.7		81	sequence	50.3	67.2
32	exception	125.3	89.7		82	set	142.4	116.7
33	expression	98.1	111.0		83	signature	7.9	1.5
34	file	216.9	372.0		84	software	20.2	21.1
35	floating/floating-point	13.5	16.7		85	solution/solve/solving	32.1	48.1
36	function	24.8	601.9		86	specification/specify	55.5	39.5
37	identifier	11.8	9.8		87	stack	56.2	9.7
38	implementation/implement	144.4	45.2		88	statement	212.1	203.1
39	index	60.5	74.2		89	stream	59.1	5.1
40	information	68.4	72.2		90	string	399.8	410.4
41	inheritance/inherit	44.1	21.1		91	structure	33.5	44.7
42	input	74.6	128.9		92	style	21.1	17.7
43	instance	137.3	110.4		93	system	253.7	55.5
44	integer	116.0	94.0		94	test/testing	85.5	138.2
45	interface	161.0	44.4		95	thread	188.2	0.6
46	iteration/iterate	11.7	20.5		96	tree	16.8	19.6
47	keyword	21.4	23.1		97	type	369.5	204.0
48	line	146.4	263.7		98	user	110.9	151.7
49	link/linked	18.9	17.4		99	value	477.5	451.1
50	list	137.1	487.0		100	variable	288.6	164.8