Journal of Information Systems Education

Volume 34
Issue 1
Winter 2023

A Data-Driven Approach to Compare the Syntactic Difficulty of Programming Languages

Erno Lokkila, Athanasios Christopoulos,
and Mikko-Jussi Laakso

Recommended Citation: Lokkila, E., Christopoulos, A., & Laakso, M.-J. (2023). A Data-Driven Approach to Compare the Syntactic Difficulty of Programming Languages. *Journal of Information Systems Education*, 34(1), 84-93.

Article Link: https://jise.org/Volume34/n1/JISE2023v34n1pp84-93.html

Received: January 13, 2022
Revised: March 27, 2022
Accepted: April 29, 2022
Published: March 15, 2023

ISSN: 2574-3872 (Online) 1055-3096 (Print)

Journal of Information System Education, 34(1), 84-93, Winter 2023

A Data-Driven Approach to Compare the Syntactic Difficulty of Programming Languages

Erno Lokkila
Department of Computing
University of Turku
Turku, 20014, Finland
eolokk@utu.fi

Athanasios Christopoulos
Mikko-Jussi Laakso
Centre for Learning Analytics
University of Turku
Turku, 20014, Finland
atchri@utu.fi, milaak@utu.fi

ABSTRACT

Educators who teach programming subjects are often wondering "which programming language should I teach first?". The debate behind this question has a long history and coming up with a definite answer to this question would be farfetched. Nonetheless, several efforts can be identified in the literature wherein pros and cons of mainstream programming languages are examined, analysed, and discussed in view of their potential to facilitate the didactics of programming concepts especially to novice programmers. In line with these efforts, we explore the latter question by comparing the syntactic difficulty of two modern, but fundamentally different, programming languages: Java and Python. To achieve this objective, we introduce a standalone and purely data-driven method which stores the code submissions and clusters the errors occurred under the aid of a custom transition probability matrix. For the evaluation of this model a total of 219,454 submissions, made by 715 first-year undergraduate students, in 259 unique programming exercises were gathered and analysed. The results indicate that Python is an easier-to-grasp programming language and is, therefore, highly recommended as the steppingstone in introductory courses. Besides, the adoption of the described method enables educators to not only identify those students who struggle with coding (syntax-wise) but further paves the pathway for the adoption of personalised and adaptive learning practices.

Keywords: Computer programming, Computer science, Data analytics, Higher education, Information systems education, Program assessment & design

1. INTRODUCTION

Computer Science and Information Systems academic programs have many distinguishable goals but also share many elements in common. Therefore, considering the commonalities in these fields, it is important that educators share knowledge and good practices. To this end, an important concern for instructors from both fields is regarding the choice of the "first programming language" (Mahatanankoon & Wolf, 2021; Robins, 2019; Sharma et al., 2020; Sobral, 2021). Indeed, selecting a programming language that is easy to grasp—yet useful for novices—is critical (Wainer & Xavier, 2018) as it greatly influences students' future academic and professional development (Medeiros et al., 2018; Quille & Bergin, 2019; Smith & Jones, 2021).

Historically, the first programming languages that have been taught and recommended for introductory programming courses have been PL1, Ada, and Pascal (Sobral, 2021). Today, Java and Python are amongst the most popular languages (Perera et al., 2021), an outcome which naturally forces higher education professors to adopt and integrate them into the syllabus as early as possible (Zhang et al., 2020).

Despite the wide adoption of these languages, their fundamental structure and application differ considerably. Therefore, gauging their advantages and disadvantages, especially in view of the needs and capacity that novices have, has received great attention (Khoirom et al., 2020; Lukkarinen et al., 2021). There are several requirements for choosing a language for an introductory programming course. These include trends and changes, the pedagogical aspects, and the language popularity both in the "real world" and in other academic institutions (Sobral, 2021). Trends and changes relate to the fluctuations in popularity of programming paradigms; the paradigms of popular first languages have shifted from procedural programming to the more modern object-oriented programming. Pedagogic aspects should also not be forgotten,

as some languages are deemed "more difficult" on the grounds of more verbose syntax, or conceptually difficult topics in programming—such as pointer arithmetic—which novice programmers need not worry themselves with. These extraneous concepts are identified as noise and any programming language taught to novices should have as little of it as possible (Pellet et al., 2019). Educational institutions are also setting boundaries on which languages can be taught. For instance, if advanced courses at an institute are taught with language X, is it pedagogically justified to first teach students language Y? All things considered, motivating students to learn a language that has no future use is hard to justify.

Within the last decade, Python has risen steadily in the popularity ranks of programming languages (Kruglyk & Lvov, 2012). Another measure of language popularity, which also considers introductory programming courses, is the "Richard Reid's List of First Programming Languages" (Siegfried et al., 2012). In Reid's 2016 List, Python overtook the second place that C++ was holding, while Java has steadily held the first position (Siegfried et al., 2016).

Python is a multi-paradigm language praised for its short and readable syntax. Learning the syntax of Python is like learning pseudocode (i.e., it removes the need to "unlearn" syntactic peculiarities when learning other languages) (Siegfried et al., 2016). Python is also a multi-paradigm language meaning that it can be used in teaching the very basics of imperative programming, object-oriented programming, and even functional programming if chosen so by the educator. Python also features dynamic typing which, on the one hand, decreases the cognitive load of learners. On the other hand, it "hides" the types of variables that may introduce bugs or, in the worst case, even faulty mental models of programming (Donaldson, 2003; Jain et al., 2020). Python also has indentation as an intrinsic part of its syntax: blocks are indented by whitespace, making the "art" of indenting code part of learning the language (Dierbach, 2014).

Java is a purely object-oriented language. It is attractive as a first language because of its popularity, the existence of broad and complete libraries, as well as its strong static typing (Ren & Ji, 2021). Researchers (Khoirom et al., 2020; Sabharwal, 1998) praise Java for its data abstraction capabilities through abstract data types—such as lists, hashmaps, and most of all, custom classes—while also pointing out several pitfalls such as the inconsistent naming conventions and confusing design decisions (e.g., the *Date* class).

Ultimately, the choice of first programming language should be based on curricular issues and the educational context, whereas strong emphasis should also be paid to the prevalence of the language in the industry. Given that both Python and Java are prevalent in the industry, the remaining factors to consider are the advanced courses that are taught in an institute. This is naturally affected by the choices of the introductory programming languages and the pedagogical merits such as the syntax of the language.

In the context of this work, we focus on the pedagogical aspects of those languages and seek to answer the following two research questions (RQs):

RQ1: *How do students' errors differ when programming in Java versus when programming in Python?*

RQ2: *Which language (Python, Java) is more appropriate for introductory programming courses in view of syntactic difficulty?*

2. RELATED WORK

Pears et al. (2007) conducted a systematic literature review on empirical studies that described findings from introductory programming courses. For the analysis of the included works, the authors categorised the manuscripts in the following broad categories: (a) studies that discuss the language choice, (b) studies that compare different languages, and (c) studies that discuss the language selection criteria. In view of this classification, we discuss the key findings and implications that emerge from more recent works related to introductory programming courses.

The comparative study that McMaster et al. (2017) performed contends that educators often follow a textbook, thus arguing that the choice of the textbook has a profound effect on the contents of the course. The authors computed how often important introductory topics are mentioned in Java and Python textbooks by measuring the repetition rates of specific programming terms. The end-goal was to shed light on the views that the authors of the textbooks have with regard to "what is considered to be important for the language." Despite the efforts made, no clear conclusions were reached in view of the question put forward (i.e., which language would be more suitable to be taught as the first programming language?).

Khoirom et al. (2020) performed an in-depth feature comparison of Java and Python based on diverse tests. Their key findings suggest that while Java executes faster, Python has smaller file sizes and less lines of code. They also list several advantages and disadvantages of both languages. The main advantages of Python are the learner-friendly syntax of the language, its open-source nature, and its multi-paradigm nature. On the other hand, the advantages of Java lie in its strict structure to guide learners' computational thinking, its *Security Manager* which offers run-time security assurance, and the strict static typing that the language inherently presents. In view of these conclusions, they recommend educators to choose the first programming language on the grounds of the projects that the students will be working on in the future. For instance, future courses in developing applications should emphasise teaching Java first whereas institutes with courses in Machine Learning or Artificial Intelligence, should explore Python.

In a language-choice paper, Uysal (2012) reports using Java as their introductory programming language. Since Java is at heart object-oriented, it forces institutions to decide early on whether to teach programming algorithms-first or objects-first. The author further argues that students who were taught Java objects-first, using the BlueJ IDE along with all its visualisations, performed better than a group of students who were taught objects-late, using text-based tools. An open question which is recommended for further examination concerns the impact that visualisations have when compared to text-based solutions. A more recent study performed by Rubiano et al. (2015) also concluded that teaching objects-first produces better results, albeit they concede that the matter of objects-first or algorithms-first is still under active discussion.

Goldman et al. (2008) created a list of difficult programming concepts using a Delphi process in which experts first individually came up with a number of concepts and rated them according to their difficulty. Subsequently, the experts were provided with statistics describing how others ranked the same concepts. The experts then provided their final ranking based on this new data. The three most difficult concepts in

programming, starting from most difficult, were identified as: (1) Inheritance, (2) Recursion, tracing, and designing, and (3) Procedure design.

Xinogalos et al. (2006) identified and divided the most frequent errors that novice Java programmers make into 10 categories and urged educators to pay special attention to these when teaching Object-Oriented Programming. The categories include the "typical" difficulties related to syntax and general computational thinking, but also the difficulties like instantiating objects, declaring constructors, and specifying inheritance relationships. It is notable that all the provided categories were still pertinent and as challenging in the more recent study that Goldman et al. (2008) performed.

Jadud (2005) analyzed 1,926 errors generated by undergraduate students using the BlueJ Java IDE. The key element that was considered for the analysis was the rate of moving between executable and non-executable code. The findings of this work were later incorporated into the Error Quotient metric (Jadud, 2006). According to Jadub (2005), "the five most common errors account for 58% of all errors generated by students while programming: missing semicolons (18%), unknown symbol: variable (12%), bracket expected (12%), illegal start of expression (9%), and unknown symbol: class (7%)" (p. 30). A surprising statement by Jadud (2005) is that for any given failed compilation event, the next event will also be a failed event with a probability of 44%. Kohn (2019) analyzed 4,091 secondary school students' Python errors using a custom-made parser. Three of the most frequent error messages were: (a) name error (35.7%), (b) indentation error (14.9%), and (c) type error (8.5%). These types together already add up to 59.1% of all student errors in Python.

The work of analysing student behaviour statistically has been taken to the next level, as machine learning models have been brought to bear on the problem. Pereira et al. (2021) describes one such machine learning method, which they prove significantly outperforms the traditional measures. They use multiple different variables, such as mid-term grades, different patterns in keystrokes, and copy-paste events. The model presented in this study is simpler, as it only uses student code submissions.

An interesting comparison is to see how Kohn's (2019) ratios of Python errors compare to Jadud's (2005) Java error ratios. The Java version of "name error" (i.e., unknown symbol) can be found in Jadud's (2005) list of novice errors with 19% of errors being of this type. The Java version of "type error" (i.e., incompatible types) is 4.3% compared to 8.5% in Python. Indentation error from Python is not so simple to transform to strict Java-errors, but generally related to it are the error classes "Bracket expected" (especially those related to { *or* }), "illegal start of expression" (which occurs when students write code outside a method), and "class of interface expected" (which occurs when students write code outside a class). Summing these three error classes gives us a sum of 24.61%. Since all these errors certainly are not block-related, we can conservatively estimate a third of these being block-related. This gives us an 8.2% error in indentation in Java versus 14.9% in Python.

Although there are several studies (Khoirom et al., 2020; Pellet et al., 2019) that compare programming languages in general or the syntactic features of Java and Python in particular, we were unable to identify any study that measures the difficulty of the syntax of a given language using a data-driven approach. Based on this inadequacy of the literature, in the present work, we compare Python and Java in terms of the errors novice students make with the language. Accordingly, we determine which language is easier for students to program in using a data-driven approach.

3. MATERIALS AND METHODS

The data was collected using a learning management system called ViLLE (Laakso et al., 2018). The platform enables teachers to distribute automatically assessed exercises, such as programming assignments or multiple-choice questions, to students. For the needs of this study, we chose two Python and two Java courses. In total, we collected code submissions (n=219,454) from 259 programming exercises made by 715 undergraduate Computer Science students (first-year college students). Although we do not control for demographics explicitly, the sample consisted mostly of Caucasian adults (18-22 years of age). The dataset contained two students who took both the Java and the Python course.

Each submission contained the following data points: (a) user, (b) timestamp, (c) code, and (d) score. The courses were given in one of the following Higher Education institutions: (a) University of Turku, (b) Open University of Turku, or (c) Turku University of Applied Sciences. It should be noted that the Open University of Turku offers undergraduate courses from the University of Turku, to the general public for a fee. Thus, courses C1 and C2 used the same course materials but C3 and C4 different. Additionally, courses C2 and C3 were taught by the same instructor. A more detailed description of the data can be found in Table 1.

4. PROPOSED METHOD

To analyse learners' programming behavior, we propose a novel method which—for simplicity purposes—we call "the transition matrix." Computing the transition matrix is possible given the following data points: (a) the submission history of a

Course	Code	Year	Language	Institution	Students	Assignments	Submissions
Algorithms & Programming	C1	2017	Java	University of Turku	294	86	124,697
Algorithms & Programming	C2	2018	Java	Open University of Turku	58	73	24,291
Introduction to Programming	C3	2020	Python	University of Turku	92	55	21,164
Introduction to Programming	C4	2020	Python	Turku University of Applied Sciences	273	45	49,302

Table 1. Overview of the Primary Data

student and (b) the timestamp for each submission. We use the term "submission" for the generated program code via which a student attempts to solve the given assignment. Further data points can be inferred from these given two data points, such as whether executing the code ended in an error or an infinite loop.

To compute the transition matrix, we introduced a state machine based on the following states (S_x):

- S_1 ***Met success***, when the student first succeeds in compiling the code after one or more unsuccessful compilation attempts.
- S_2 ***Met error***, when the student first encounters an error that prevents execution of code.
- S_3 ***Confused***, when the student's previous attempt was not successful and the current attempt also gave an error.
- S_4 ***Runtime error***, when the student's attempt ended with a runtime error.
- S_5 ***Unmodified error submission***, when the student resubmits an erroneous code with no modifications.
- S_6 ***Unmodified success submission***, when the student resubmits an executable code with no modifications.
- S_7 ***Repeated success***, when the student submits compiling code after one or more successful attempts.
- S_8 ***Success on first attempt***, when the student's first submission is correct and solves the given assignment with full marks.

The adjacency matrix for the state machine is shown in Table 2. The algorithm to compute the transition matrix is as follows:

1) Create consecutive pairs from all compilation events in a session: $(e_1, e_2), (e_2, e_3), \ldots, (e_{n-1}, e_n)$ based on the event timestamp.
2) For each event pair, mark which state transition occurred in the transition matrix.
3) For each cell in the transition matrix, divide the value of the cell by the sum of all events on the same row.

This results in a matrix of the probabilities which describes the probability of each state transition for a given student. This matrix can then be used for further analysis to, for instance, cluster students based on their transition matrices or to derive a single value corresponding to the skill level of the student.

	S1	S2	S3	S4	S5	S6	S7
S1		X		X	X		X
S2	X		X		X	X	
S3	X		X		X	X	
S4		X		X	X		X
S5	X		X		X	X	
S6	X		X		X	X	
S7		X		X	X		X

Table 2. The State Machine Adjacency Matrix (rows are start states and columns are the end states)

5. RESULTS

All submissions were first compiled and any compiler error messages were collected. The Java code was compiled with JDK13 and the Python code using the inbuilt compile method. The successfully compiled submissions were then executed and any run-time errors were collected. If program execution took longer than 2 seconds, the program was terminated and the submission was marked as "infinite loop." The submission details are presented in Table 3.

As mentioned earlier, Python is marketed as an easy-to-learn language with simple syntax, whereas Java is more difficult due to the inherently verbose syntax (Khoirom et al., 2020). This conclusion is verified as, nearly 60% of all student Java submissions, could not be executed successfully, whereas the error rate for Python is only slightly over 50% (Table 3). To determine whether this difference is statistically significant, we performed a chi-square (χ^2) test of independence. The test confirmed the significance of the result with $p<0.001$ ($\chi^2(1, 219{,}454)=1070.2719$). This leads us to conclude that novice programmers are, indeed, more likely to create working and executable code in Python. Interestingly, C4 has noticeably more compilation errors but fewer run time errors than C3, even though both are Python courses. As the courses were given in different universities, we hypothesised that this may be attributed to the pedagogical focus each course had. For instance, the quality of course materials, the delivery method, or students' differences in pre-existing knowledge of programming may also explain the difference in part. Nevertheless, we do not possess any data to explore these hypotheses any further and thus, this observation is recommended for future research (i.e., *How much do each of these differences contribute to overall error counts?*). A startling observation is that more than half of the student attempts ended in an error. This naturally raises the following question: *What were the errors that students received?* In view of this question, we list the most frequent errors that learners received for both Python and Java in the following sections.

5.1 Java Errors

The five most common errors in Java comprise 50% of all the errors students encounter and are as follows (Table 4): (1) "Missing semicolon" (11.19%), (2) Unknown variable (10.9%), (3) Missing bracket (10.13%), (4) Type errors (9.15%), and (5) Unknown method (8.68%).

5.2 Python Errors

The two most frequent errors in Python account for 52% of all errors students encountered (Table 5). These errors are: (a) "Invalid syntax" (30.32%) and (b) "Using undeclared variables" (21.68%). The "Invalid syntax" error is encountered when compiling (using the inbuilt compile-method) Python code and is given when students, for instance, forget colons, parentheses, or commas. In order to make a reasonable comparison to Java, the Python error "Invalid syntax" roughly corresponds to a combination of the Java errors at ranks 1, 3, 6, 8, 15, and 16 for a combined error ratio of 31.26%, which is not that different from the rate of syntax errors for Python.

5.3 Cluster Analysis of the Courses

After generating the transition matrix for each student, an unsupervised machine learning technique was utilised to identify inferences from the gathered data. For the needs of this work, the *k*-means clustering algorithm was utilised. When

Course	Language	Successes	Compile errors	Run-time errors	Timeouts	Error rate
C1	Java	51,250	67,207	5,280	960	58.9%
C2	Java	9,780	13,211	1,026	274	59.7%
C3	Python	10,075	3,418	7,492	179	52.4%
C4	Python	24,013	13,402	11,398	489	51.3%

Table 3. Overview of the Submissions Made across the Four Courses

clustering students, we ignored all transitions which ended in either S_5 (unmodified error submission) or S_6 (unmodified success submission), as these states do not give any meaningful information on the skill of the student (e.g., a student may resubmit code to verify an error message or to reread the output).

Rank	Error	Rate	Rank	Error	Rate
1	Semicolon expected	11.19 %	11	Reached end of file while parsing	2.53 %
2	Unknown variable	10.90 %	12	op application error	2.45 %
3	Bracket expected	10.13 %	13	Method application error	2.34 %
4	Incompatible types	9.15 %	14	String index out of bounds	2.28 %
5	Unknown method	8.68 %	15	Not a statement	2.18 %
6	Illegal start of expression	5.65 %	16	Illegal start of type	2.11 %
7	Missing return statement	4.16 %	17	Unknown class	2.08 %
8	<identifier> expected	3.82 %	18	Variable already defined	1.56 %
9	Array index out of bounds	3.07 %	19	invalid method signature; return type required	1.13 %
10	class, interface, or enum expected	2.57 %	20	'else' without 'if'	1.04 %

Table 4. Most Common Errors in Java

Rank	Error	Ratio	Rank	Error	Ratio
1	Invalid syntax	30.32 %	11	Using variable before assignment	1.59 %
2	Using undeclared variable	21.68 %	12	Wrong number of parameters passed	1.42 %
3	Invalid literal for base conversion	5.33 %	13	Object not callable	1.10 %
4	Unsupported operand type(s)	4.51 %	14	Non-integer index	1.10 %
5	Using missing attribute	4.37 %	15	Not all args converted in string formatting	0.84 %
6	Expected an indented block	4.22 %	16	Object not subscriptable	0.81 %
7	Unexpected indent	2.89 %	17	Wrong type parameter passed	0.81 %
8	Index out of range	2.37 %	18	Wrong types for concatenation	0.74 %
9	Object to iterable	2.31 %	19	'return' outside function	0.70 %
10	Unindent does not match outer indentation	1.90 %	20	Using missing object attribute	0.60 %

Table 5. Most Common Errors in Python

	Cluster	N	Avg. submissions	Avg. time on task	Avg. error rate %
Java	G1	131	596.74	1187.74	61.82
	G2	178	370.81	807.31	48.49
	G3	43	111.84	538.06	32.18
Python	G1	159	304.02	1739.41	49.06
	G2	151	121.85	972.04	48.21
	G3	55	67.76	565.0	35.82

Table 6. Naïve Metrics of Skill for Clusters' Creation

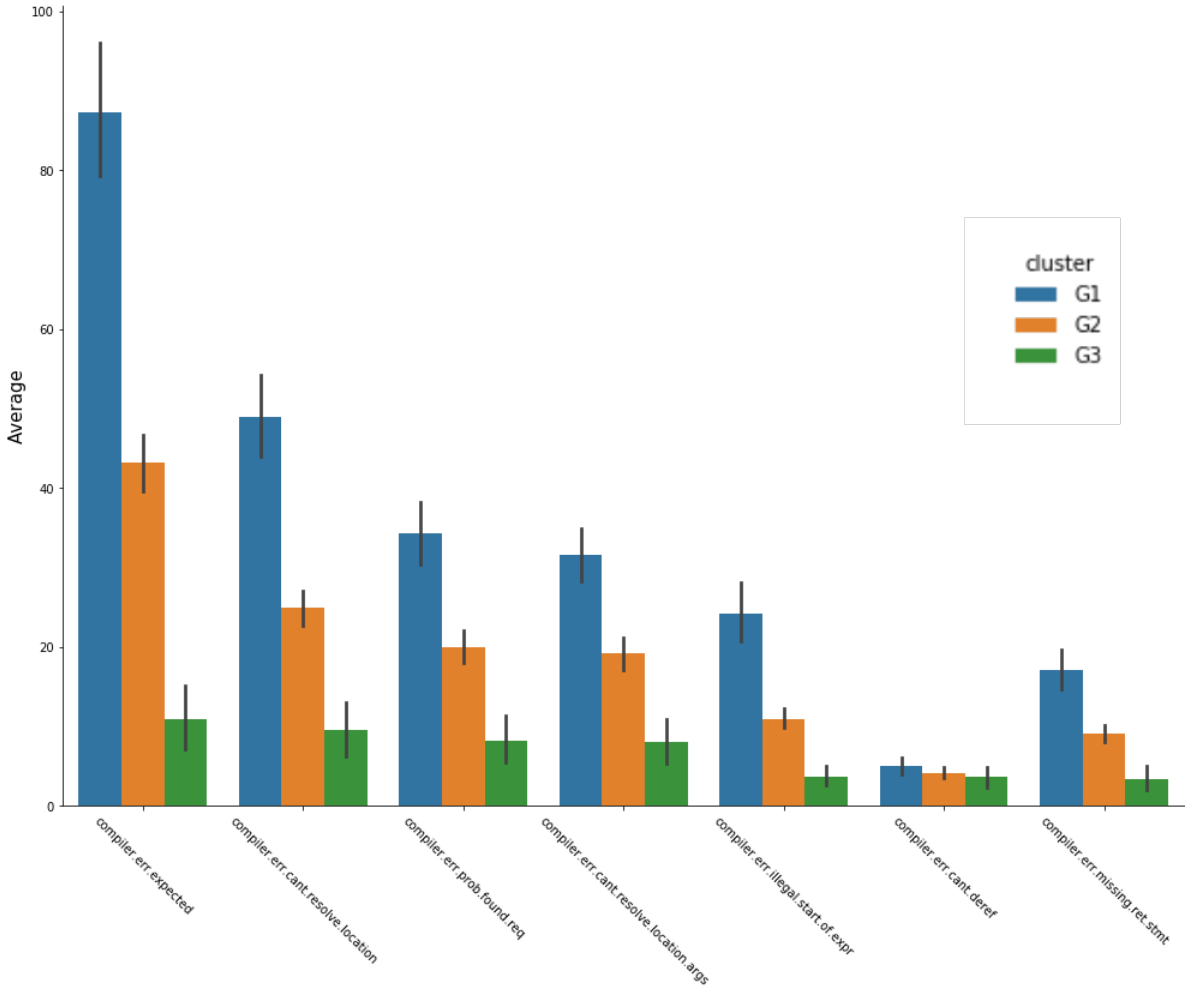


Figure 1. Top 7 Errors Ranked in Descending Order According to G3

For both programming languages, group G1 performs poorly on all these metrics, whereas group G3 outperforms all the other clusters. In order to verify this observation, we examined the distribution of the data (number of average submissions, average time on task, error rate of both languages) separately, using the Shapiro-Wilks Normality Test, and accordingly performed an Analysis of Variance (ANOVA) on the attributes. All tests yielded significant variation between the groups (p<.05). Given that the one-way ANOVA does not specify which groups are different, however, we performed a post-hoc Tukey's HSD test to verify the observed differences. We found statistically significant differences between the attributes of all groups (p<.001) except two: the "Error rate" in Python between G1 and G2 and the "Average time on task," also in Python, between G2 and G3.

Further analysis of the formed clusters revealed that the distribution of errors within clusters is remarkably similar. Figure 1 shows the seven most common compiler error types in Java and how many students made the error, ordered into descending order by G3, which we believe is the cluster containing the top-performers.

The data for Python errors revealed a similar phenomenon, hence no figure is provided. The ordering of errors is not surprising, however. The interesting observation here is that the number of errors made by the different clusters follows a seemingly similar ratio. This implies that generating these errors tends to be more a matter of lack of attentiveness and routine as opposed to a misconception about programming or lack of skill. An interesting outlier in the most common errors is the type *compiler.err.cant.deref*, which manifests in error messages such as "int cannot be dereferenced." This is caused by attempting to call a method on a primitive value, as in the following code snippet: "int i=5; i.methodCall();". This begs the question "*which errors are those that G3 did not make, but G1 and G2 did?".*

While all clusters made errors, G1 made the most number of distinct error types (67), G3 the least (38) and G2 in between these (61). This further confirms our assumption that the formed clusters represent groups of students with different programming skill levels. We found a total of 31 different error types that were made only by either G1 or G2, but not G3. A majority of these errors occurred only a few times. Table 7 shows, for Java, the errors which were done most often. We chose the cutoff point to be the mean value of the occurrences (8). The same analysis for Python data indicated that all errors made by G1 and G2 were also created by G3.

Error	Occurrences (students)
Unclosed string literal	47
Array dimension missing	39
Empty character literal	22
Illegal start of statement	18
Missing method body	17
Break outside loop or switch-case	16
Integer literal too large	11
Cannot assign value to final variable	10

Table 7: Java Errors Which Were Made by G1 and G2, but Not G3

To facilitate the reading of the above mentioned observations, we plotted the students according to their total score and their number of submissions in view of cohort-clusters. As Figure 2 illustrates, strong students clumped together with high scores and low submissions, whereas the total score of the others grows with the number of submissions.

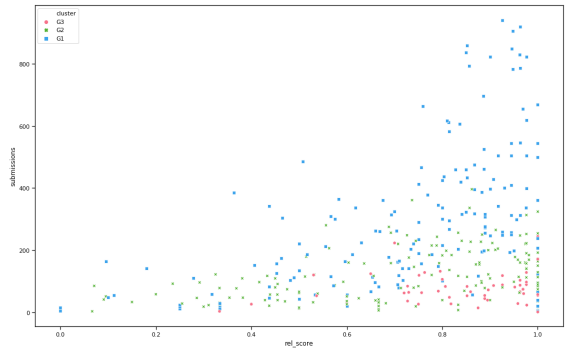


Figure 2. Clustering of Python Students in C3 and C4 Using a Model Trained with Python Data

When interpreting Figure 2, it should be noted that students in the course had been continuously encouraged by the lecturer to work collaboratively. It is, therefore, likely that some students from G1 were helped by their peers which, in turn, enabled them to achieve the maximum score on the course. In either case, based on the available data, we can conclude that clustering the transition matrix using the *k*-means algorithm can, indeed, separate students into distinct groups.

Next, we determined whether the difficulty of Java and Python is comparable. Our null hypothesis is that there is no difference in the difficulty of the two languages. If this hypothesis holds true (i.e., the two languages are indeed of equal difficulty), using a trained *k*-means model from one course to cluster the other should not produce a significant change in the clustering result. To test our hypothesis, we computed a transition matrix for all students in C1 and C2 and trained the model (k=3). We then computed transition matrices for all students from C3 and C4 and used the trained model to cluster them. The results are described in Table 8.

Cluster	Size	Avg. submissions	Avg. time on task	Avg. error rate %
G1	9	407.78	2498.0	58.72
G2	181	266.29	1563.42	49.74
G3	175	106.27	841.76	42.97

Table 8. Python Data Clustered with a Model Trained on Java Data (compare with Table 6)

The resulting clustering was strikingly different (Figure 3). Not only did the cluster sizes change significantly, but remarkably most students "moved up" from their cluster, when compared to Figure 2.

Precisely, most students in G1 in the Python-trained model were in G2 in the Java-trained model and nearly all students from G3 in the Python-trained model remained in the best-performing cluster. Consider our null hypothesis of no difference in difficulty. If the students on the Python courses had been on the Java courses and performed as they did on the Python course, almost none of them would have been in the struggling group (G1). Thus, we reject the null hypothesis and accept that Python and Java are different in their difficulty, with Python being the easier language to create working code.

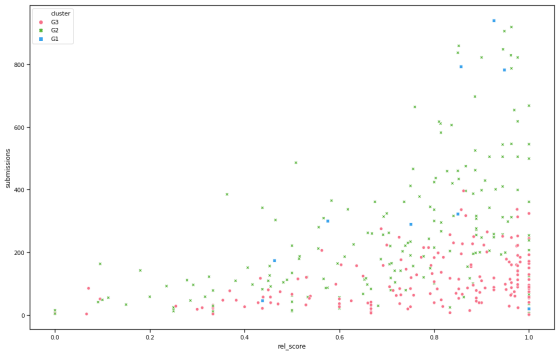


Figure 3. Clustering of Python Students in C3 and C4 Using a Model Trained with Java Data

6. DISCUSSION

In this work, we sought to find which language—Java or Python—is syntactically easier for novice programmers to learn. In our mission to attain this objective, we introduced a complete, data-driven approach, which can be used to examine and compare the difficulty of different programming languages. This is especially true when introduced in the context of introductory programming courses and further highlighted the differences that novices face when learning one of the two most commonly used first languages. In the remaining discussion, while considering the two-fold contribution of this work, we provide direct answers to the *Review Questions* put forward and further complement them with guidelines that can assist educators and instructors to reform and reconstruct their courses (be it introductory or advanced).

The choice of first programming language is important, but not critical (Pellet et al., 2019), as introductory programming courses tend to have high dropout rates (McCracken et al., 2001; Medeiros et al., 2018; Quille & Bergin, 2019). Choosing

a language where students make, on average, fewer mistakes is beneficial to improving self-efficacy as it is highly linked with motivation (Öqvist & Malmström, 2018). The data that emerged from this exploratory study shows that Python is easier for students to program in, at least when compared to Java. They make fewer errors and, therefore, need fewer attempts to compile working code. In view of this, we can expect a positive motivational boost on their attitude toward programming, especially at this fragile stage where most of the concepts taught are new. This outcome does not ultimately mean that Python is "the best" choice. Other factors need to be considered as well, such as which related courses that the institution offers (Sobral, 2021).

The choice of language does not seem to affect the types of errors students make. Receiving errors like "syntax error" and attempting to use "undefined variables" top the charts for both languages. Having awareness of the most common errors that novices make enables more efficient teaching. Developers, therefore, are advised to create tools, error messages, and other resources—specifically addressed for novices—on how to solve these errors. Teachers and instructors can focus on the causes and how to correct them in-class, enabling students to quickly build better programming routines. Researchers can utilise this knowledge to build better models and possibly design even better programming languages for novice programmers.

While analysing the errors of both languages, it became apparent that roughly 30% of them were related to syntax. When considering all the errors, the difference between languages grew to roughly 60% of attempts for Java and only slightly over 50% for Python. The three most commonly made errors in Java were "Missing semicolon," "Missing bracket," and "Unknown variable." These three errors account for roughly 30% of all errors made. The three most commonly made errors in Python were syntax errors (such as missing commas or semicolons), using "undeclared variables," and "providing faulty arguments" to the inbuilt int-method, which transforms strings of digits into an integer. These three error types cover over 55% of all errors encountered by students in our dataset. Educators and instructors, therefore, are advised to pay particular emphasis when delving into these topics.

Another way to facilitate students learning can be via the provision of more exercises targeted to these topics. Since the provided data-driven approach was seemingly able to divide students into similarly performing clusters, teachers and educators can focus their efforts on the worst-performing cluster and offer these exercises to the portion of students who truly need them. How to most effectively help the students identified as needing help is outside the scope of this paper and remains subject to further study.

The merits and the challenges that Python and Java present, when introduced in introductory programming courses have been examined via the proposed transition matrix in which we trained a *k*-means clustering model using independent datasets (Java or Python groups). To facilitate the clustering process, we chose the number of groups to be three (k=3), so as to divide students into a high-performing group (G3), average group (G2), and low-performing group (G1). We proved that these groups were distinct, as determined by the average number of submissions, the average time spent on assignments, and the error rates.

We created one clustering of the transition matrices for the Python group (C3 and C4 from Table 1), using the Java-trained model, and another using the Python-trained model, then compared them. With the Java trained model, G3 comprised nearly half of the students and G2 the other half. Only nine students were left in G1, the low-performing group. Whereas with the Python-trained model, both G3 and G2 comprised of over 40% of the sample, whereas G1 enclosed 15% of all students. This difference in cluster sizes can be explained by considering the difference in language difficulty.

Since these are all introductory programming courses, we assume that, in terms of distributions, no major knowledge-related differences exist between the students who took the Java course and those who took the Python course. That is, we assume that the percentage of students who have previous experience in programming is similar between all courses, as well as the percentage of students who do not have any previous experience. This allows us to interpret the difference in the cluster sizes to the mean values under the assumption that, all else being equal, if the students who took the Python course had instead taken the Java course, only 9 students out of 356 would have struggled with programming. This result is very unlikely, given the extensive literature on the difficulty of programming (e.g., McCracken et al., 2001; Jadud, 2005; Kohn 2019; Quille & Bergin, 2019). Thus, we can safely conclude that Python is easier for novice programmers to grasp.

7. LIMITATIONS AND FUTURE DIRECTIONS

For the replicability of the study, the following limitations and delimitations should be taken into account. First, we did not check what percentage of exercises was shared between courses with the same language. For instance, if one course had exercises which had significantly harder questions with longer answers, it is possible that differences in error rates and counts are due to the exercise difficulty. This could partially explain why C4 had significantly more compile errors and fewer run-time errors than C3. Nevertheless, we mitigated this issue by including two courses in our dataset as a means to average out any differences. We also did not account for different pedagogic approaches used in the different courses. As the courses using the same programming language had almost exactly the same error rate (overall), however, we believe the courses applied the same pedagogic approach. We currently have no explanation for why C3 and C4 error types (compile versus run time) differed to such a degree. Finally, we did not introduce any test to identify cohorts' prior experience with programming or programming skills. Therefore, it is possible, albeit very unlikely, that the students in the Python group had significantly more programming experience than the students in the Java group. We have no reason to believe any drastic difference exists in the starting skill level between novice Java learners and novice Python learners, given that both cohorts are first-year undergraduate students with no previous contact with programming or individuals (from the general public) who are simply interested in learning programming.

This study only scratched the surface regarding the different error types that students of different skill levels generate. We observed that the students identified as top-performers did not make certain types of errors at all. Further studies should find fruitful ground in determining whether

specific errors or patterns of generated errors correspond directly to a misconception in programming.

The data collection model and the clustering method described in this work can be adopted by educators and practitioners worldwide to discover their learners' programming patterns or trends as well as their skill levels so that they can efficiently cluster them into distinct groups. These groups can be offered targeted exercises to aid in learning programming. For instance, the worst performing group may be offered simpler syntax-based exercises (since that is what they struggle with), whereas the other groups may instead be given more challenging algorithmic or problem-solving focused exercises.

This study also paves the way for interesting research directions. By clustering programming students, educators can provide learners with adaptive exercises, adjusted to their particular needs and tailored to their capabilities. Combining the capability to cluster students and existing models for processes that cause errors in student programming (c.f., Ko & Myers 2003) can further increase our understanding of programming errors, their causes, and better ways to teach programming. Other future works can look into the verification of the clusters, their validity, and practical application. In addition, researchers may consider the collection of qualitative data so that the key findings can be further related to and cross-referenced with the clusters. For instance, exploring learners' feelings over programming may reveal whether the worst-performing students also have negative feelings toward programming. Finally, in terms of adaptive and personalised learning, another direction may be the introduction of procedures capable of providing struggling students with additional learning material (e.g., recommendations for further reading) or targeted exercises. The impact of such intervention can then be evaluated on a weekly basis as new data will emerge.

8. REFERENCES

Dierbach, C. (2014). Python as a First Programming Language. *Journal of Computing Sciences in Colleges*, 29(3), 73-73.

Donaldson, T. (2003). Python as a First Programming Language for Everyone. *Proceedings of the 2003 Western Canadian Conference on Computing Education* (pp. 1-2).

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Identifying Important and Difficult Concepts in Introductory Computing Courses Using a Delphi Process. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (pp. 256-260). Portland, USA: ACM.

Jadud, M. C. (2005). A First Look at Novice Compilation Behaviour Using BlueJ. *Computer Science Education*, 15(1), 25-40.

Jadud, M. C. (2006). Methods and Tools for Exploring Novice Compilation Behaviour. *Proceedings of the 2nd International Workshop on Computing Education Research* (pp. 73-84). Canterbury, United Kingdom: ACM.

Jain, S. B., Sonar, S. G., Jain, S. S., Daga, P., & Jain, R. S. (2020). Review on Comparison of Different Programming Language by Observing Its Advantages and Disadvantages. *Research Journal of Engineering and Technology*,11(3), 133-137.

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). Comparative Analysis of Python and Java for Beginners. *International Research Journal of Engineering and Technology*, 7(8), 4384-4407.

Ko, A. J., & Myers, B. A. (2003). Development and Evaluation of a Model of Programming Errors. *Proceedings of the IEEE Symposium on Human Centric Computing Languages and Environments* (pp. 7-14). Auckland, New Zealand: IEEE.

Kohn, T. (2019). The Error Behind the Message: Finding the Cause of Error Messages in Python. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 524-530). Minneapolis, USA: ACM.

Kruglyk, V., & Lvov, M. (2012). Choosing the First Educational Programming Language. *Proceedings of the 8th International Conference on ICT in Education, Research, and Industrial Applications* (pp. 188-189). Kherson, Ukraine: CEUR Workshop Proceedings.

Laakso, M. J., Kaila, E., & Rajala, T. (2018). ViLLE–Collaborative Education Tool: Designing and Utilizing an Exercise-Based Learning *Environment. Education and Information Technologies*, 23(4), 1655-1676.

Lukkarinen, A., Malmi, L., & Haaranen, L. (2021). Event-Driven Programming in Programming Education: A Mapping Review. *ACM Transactions on Computing Education*, 21(1), 1-31.

Mahatanankoon, P., & Wolf, J. (2021). Cognitive Learning Strategies in an Introductory Computer Programming Course. *Information Systems Education Journal*, 19(3), 11-20.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., ... & Wilusz, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (pp. 125-180). Canterbury, UK: ACM.

McMaster, K., Sambasivam, S., Rague, B., & Wolthuis, S. (2017). Java vs. Python Coverage of Introductory Programming Concepts: A Textbook Analysis. *Information Systems Education Journal*, 15(3), 4-13.

Medeiros, R. P., Ramalho, G. L., & Falcão, T. P. (2018). A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. *IEEE Transactions on Education*, 62(2), 77-90.

Öqvist, A., & Malmström, M. (2018). What Motivates Students? A Study on the Effects of Teacher Leadership and Students' Self-Efficacy. *International Journal of Leadership in Education*, 21(2), 155-175.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., ... & Paterson, J. (2007). A Survey of Literature on the Teaching of Introductory Programming. *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education* (pp. 204-223). Dundee, Scotland: ACM.

Pellet, J. P., Dame, A., & Parriaux, G. (2019). How Beginner-Friendly Is a Programming Language? A Short Analysis Based on Java and Python Examples. *Proceedings of the 12th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives* (pp. 1-13). Larnaca, Cyprus: Springer.

Pereira, F. D., Fonseca, S. C., Oliveira, E. H., Cristea, A. I., Bellhäuser, H., Rodrigues, L., ... & Carvalho, L. S. (2021). Explaining Individual and Collective Programming Students' Behavior by Interpreting a Black-Box Predictive Model. *IEEE Access*, 9, 117097-117119.

Perera, P., Tennakoon, G., Ahangama, S., Panditharathna, R., & Chathuranga, B. (2021). A Systematic Review of Introductory Programming Languages for Novice Learners. *IEEE Access*, 9, 88121-88136.

Quille, K., & Bergin, S. (2019). CS1: How Will They Do? How Can We Help? A Decade of Research and Practice. *Computer Science Education*, 29(2-3), 254-282.

Ren, Y., & Ji, S. (2021). A Study of Teaching International Students Java Programming at Shanghai Dianji University. *International Journal of Science*, 8(3), 71-74.

Robins, A. V. (2019) Novice Programmers and Introductory Programming. In S. A. Fincher & A. V. Robins (Eds.), *The Cambridge Handbook of Computing Education Research* (pp. 327-376), Cambridge, UK: Cambridge University Press.

Rubiano, S. M. M., López-Cruz, O., & Soto, E. G. (2015). Teaching Computer Programming: Practices, Difficulties and Opportunities. *Proceedings of the 2015 IEEE Frontiers in Education Conference* (pp. 1-9). Texas, USA: IEEE.

Sabharwal, C. L. (1998). Java, Java, Java. *IEEE Potentials*, 17(3), 33-37.

Sharma, M., Biros, D., Ayyalasomayajula, S., & Dalal, N. (2020). Teaching Programming to the Post-Millennial Generation: Pedagogic Considerations for an IS course. *Journal of Information Systems Education*, 31(2), 96-105.

Siegfried, R. M., Greco, D., Miceli, N., & Siegfried, J. (2012). Whatever Happened to Richard Reid's List of First Programming Languages?. *Information Systems Education Journal*, 10(4), 24-30.

Siegfried, R. M., Siegfried, J., & Alexandro, G. (2016). A Longitudinal Analysis of the Reid List of First Programming Languages. *Information Systems Education Journal*, 14(6), 47-54.

Smith, T. C., & Jones, L. (2021). First Course Programming Languages within US Business College MIS Curricula. *Journal of Information Systems Education*, 32(4), 283-293.

Sobral S. R. (2021). The Old Question: Which Programming Language Should We Choose to Teach to Program?. *Proceedings of the 2021 International Conference on Advances in Digital Science* (pp. 351-364). Salvador, Brazil: Springer.

Uysal, M. P. (2012). The Effects of Objects-First and Objects-Late Methods on Achievements of OOP Learners. *Journal of Software Engineering and Applications*, 5(10), 816-822.

Wainer, J., & Xavier, E. C. (2018). A Controlled Experiment on Python vs C for an Introductory Programming Course: Students' Outcomes. *ACM Transactions on Computing Education*, 18(3), 1-16.

Xinogalos, S., Sartatzemi, M., & Dagdilelis, V. (2006). Studying Students' Difficulties in an OOP Course Based on Bluej. *Proceedings of the International Conference on Computers and Advanced technology in Education* (pp. 82-87). Lima, Peru: ACTA Press.

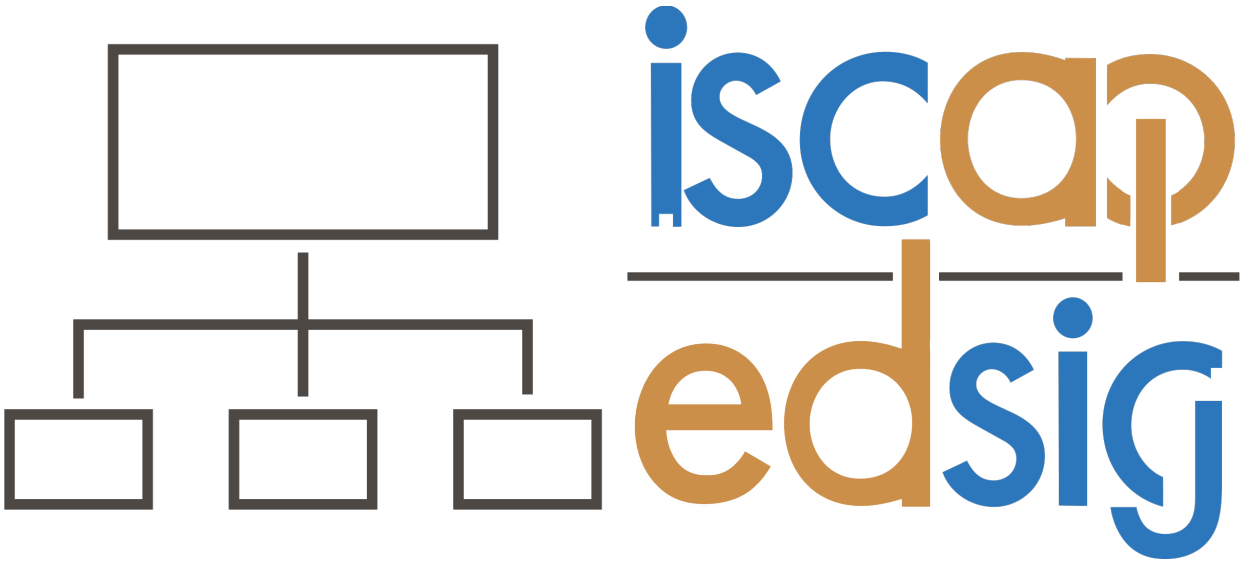
Zhang, X., Crabtree, J. D., Terwilliger, M. G., & Jenkins, J. T. (2020). Teaching Introductory Programming from A to Z: Twenty-Six Tips from the Trenches. *Journal of Information Systems Education*, 31(2), 106-118.

AUTHOR BIOGRAPHIES

Erno Lokkila is a doctoral candidate in the Department of Computing at the University of Turku, Finland. He is currently working as a university lecturer and teaches several first year computer science courses. His PhD thesis involves improving programming education by identifying the students in need and providing them with targeted assistance. His other research interest are gamification, learning analytics, and programming language theory.

Athanasios Christopoulos is a Research Fellow in the Faculty of Science at the University of Turku, Finland. Dr. Christopoulos is currently working for the Centre for Learning Analytics where he investigates matters related to digital inclusion, educational technology advancement, immersive technologies, and learning analytics. The Centre is also developing "ViLLE," a digital learning platform, that includes content and exercises for studying mathematics, programming, and languages.

Mikko-Jussi Laakso is the director of the Centre for Learning Analytics at the University of Turku, Finland. His main research interests are Learning Analytics, Computer Assisted Learning, Math & Programming Education, Gamification, Learning Design, Machine Learning & AI in Education. He has 20 years of experience from university and research-based development of the education through educational technology solutions. He has published more than 100 international peer-reviewed articles, and collected more than 4M € in R&D projects. The centre is developing an UNESCO-awarded tool named "ViLLE" – The collaborative education platform. The system is utilized by 60% of schools and learners are doing more than 200,000,000 tasks annually.



Information Systems & Computing Academic Professionals

Education Special Interest Group

STATEMENT OF PEER REVIEW INTEGRITY

All papers published in the ***Journal of Information Systems Education*** have undergone rigorous peer review. This includes an initial editor screening and double-blind refereeing by three or more expert referees.

ISSN: 2574-3872 (Online) 1055-3096 (Print)