On Changing the Curriculum Programming Language from Java to Python

Kaila, Erkki Tapio

2024-02-06

Kaila, E T, Luukkainen, M, Laaksonen, A & Lemström, K 2024, On Changing the Curriculum Programming Language from Java to Python. in A Mühling & I Jormanainen (eds), Proceedings of the 23th Koli Calling International Conference on Computing Education Research. ACM, pp. 1-7, Koli Calling International Conference on Computing Education Research, Koli, Finland, 13/11/2023. https://doi.org/10.1145/3631802.3631814

http://hdl.handle.net/10138/571013
10.1145/3631802.3631814

On Changing the Curriculum Programming Language from Java to Python (Discussion Paper)

Erkki Kaila
University of Turku & University of Helsinki
Finland

Matti Luukkainen
University of Helsinki
Finland

Antti Laaksonen
University of Helsinki
Finland

Kjell Lemström
University of Helsinki
Finland

ABSTRACT

The rapid evolution of programming languages in recent years has introduced new languages, tools, libraries, and technologies. Python has emerged as a prevalent programming language, leading many institutions and universities to adopt it as the default language for their programming curriculum. This paper examines the transition from Java to Python as the main programming language of the curriculum at the University of Helsinki. The shift to a new language of instruction is a significant and critical undertaking that profoundly impacts the curriculum. Thus, it requires careful consideration and meticulous execution. To assess our success in the change, we analyzed the pass rates and average student grades across multiple courses before and after the change. Moreover, we administered a questionnaire to teaching assistants experienced in Java and Python and collected weekly feedback from students in the first implementations of the renewed course. Based on our findings, the change was successfully executed, as the learning outcomes were not adversely affected by the language change. Thus, this article likely offers valuable insights for educational institutions considering similar transitions.

KEYWORDS

Programming language, Curriculum design

ACM Reference Format:
Erkki Kaila, Matti Luukkainen, Antti Laaksonen, and Kjell Lemström. 2023. On Changing the Curriculum Programming Language from Java to Python (Discussion Paper). In *23rd Koli Calling International Conference on Computing Education Research (Koli Calling '23), November 13–18, 2023, Koli, Finland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3631802.3631814

1 INTRODUCTION

The selection of the first programming language in a computer science bachelor's curriculum holds significant weight, particularly in Finland, where computer science is not typically taught in high school. This initial course serves as the freshman's inaugural exposure to the field, shaping their overall experience with computer science education. The choice of programming language can affect their performance and engagement in computer science studies. A well-considered selection sets the stage for their understanding, enthusiasm, and future success in the field. Therefore, careful consideration of the first programming language becomes pivotal in providing a solid foundation for students' computer science journey in general.

This paper discusses our experiences of changing the curriculum programming language in an established, research-first University of Helsinki, Finland. Our curriculum underwent a change in 1997 when Java replaced Pascal. This decision was based on the benefits of Java's cross-platform independence, strong object-oriented programming support, and extensive standard library. However, after almost 25 years, it was decided that Java had to be replaced by another language, which turned out to be Python. We believed that Python would take over as the preferred programming language to learn due to its simple and intuitive syntax, dynamic typing (eliminating the need to declare variable data type beforehand), and faster feedback loop, which makes writing, testing, and debugging faster. Another thing that spoke in Python's favour is its versatility in terms of applications. Whereas Java is mainly used for enterprise systems nowadays, Python is very suitable for a wide range of application areas, e.g., data analysis, AI and machine learning applications, and digital humanities.

Changing the primary programming language in a curriculum is a complex process. In this study, we 1) examine the influence of the change on students' academic outcomes by analyzing the pass rates and average grades across multiple courses, 2) present the results of a questionnaire that was administered to teaching assistants experienced in teaching both Java and Python and 3) analyze the student feedback collected from the first two instances of the renewed Python course.

2 RELATED WORK

The transition from Java to Python has happened in many universities, and several related papers describe teachers' experiences. In most studies, the transition has been regarded as a success. Necaise [22] discusses the transition at Washington and Lee University, where they observed that students who use Python could use more time for problem-solving when they have fewer difficulties with the syntax. Agarwal et al. [8] compare Java and Python when teaching CS0 and prefer Python for several reasons, including its simplicity and the fact that a non-profit organization develops it. Miller, Bradley and Ranum [21] describe another approach for teaching

Koli Calling '23, November 13–18, 2023, Koli, Finland

ACM ISBN 979-8-4007-1653-9/23/11.
https://doi.org/10.1145/3631802.3631814

programming, where they first use Python for learning the basics and then move to Java for more advanced topics.

Some studies focus more on the actual learning results of the students. Jayal et al. [14] present a quantitive evaluation of results when teaching programming using Java and Python. Their study suggests that learning results are better with Python. Koulouri et al. [18] compare several factors when teaching programming, including the choice of programming language. They conclude that using a simple language like Python makes it easier to learn programming concepts.

However, not all reports praise the simplicity of Python. Hunt [13] describes an unsuccessful transition from Java to Python followed by a change back to Java two years later. According to Hunt, Python is not a good language for teaching loops, arrays, and the basics of object-oriented programming. On the other hand, while Java is more complex than Python, it has also been suggested that even Java is too easy a language for teaching programming. Spolsky [26] worries that students who use Java do not learn two central programming concepts: pointers and recursion.

3 COURSE DESIGN AND IMPLEMENTATION

3.1 Background and Course Design

Our university moved from Pascal to Java as the first programming language in 1997. After that, the introductory programming course was completely redesigned in 2010 and has been developed in many ways since then following the *extreme apprenticeship model* [27]: a teaching method that emphasizes students' active learning and continuous two-way feedback between students and course staff.

While Java was still a justifiable choice in 2010 when the previous course version was designed, there were several reasons for changing the first taught programming language in 2020. Java is still used widely, but during the recent decade, other languages, especially JavaScript and Python, have gained quite a lot of ground in the industry. It can be argued that they would be ideal choices as the first programming language due to their simple syntax and less technical requirements.

After careful consideration, Python was selected as the language to this end. It is considered an excellent first language because its syntax is simple compared to other popular languages, and as an interpreted language, it is rather easy to adapt. Dynamic typing also lowers the threshold of starting to use the language. Dynamic typing may complicate things in advanced programs [7], but newer versions of Python support *type hints* [15] bringing it closer to static typing. Another significant reason was Python's versatility and broad applicability in areas other than computer science.

While Java is strongly object-oriented, Python is a multi-paradigm language. We decided to start the course with a procedural paradigm and gradually proceed to object-oriented and functional programming. Although the early introduction of objects and classes can be considered beneficial for learning (see, e.g. [10]), we argue that a procedural paradigm with no excessive syntactic features provides the students with a better start. This is in line with other studies ([9], [12], [25], [20], [24]). In addition, while object orientation is an essential skill of software professionals, procedural Python scripts can be used for data processing, and students may apply their programming skills in different fields without learning advanced programming concepts.

From the very beginning, some fundamental design principles were adopted. First, we wanted to make the course easy to adapt: starting the course should be simple, with no additional installation of tools or complicated registration required. Second, we wanted to keep the material flow simple and provide plenty of code examples and illustrations that accompany the textual content. Third, the material should be complete without the need for any additional resources. Finally, active learning was emphasized. The hands-on approach is generally seen as beneficial in learning to program (see, e.g. [16]), and we wanted to make no exception.

3.2 Course Content

Our curriculum's programming course is divided into two seven-week long parts: *Introduction to Programming* and *Advanced Programming Course*. Completing the first part of the course should provide participants necessary tools for using Python to solve simple problems and, for example, to manipulate data in CSV or text files. The second part focuses on different programming paradigms and structuring the code of more complex programs. The topics covered each week can be seen in Table 3.

We decided to focus heavily on programming exercises. Each week contains 20 to 30 exercises for students to write a Python program according to specifications. Some exercises are connected to form more complex programs. This aims to illustrate the process of designing and writing applications by dividing the work into smaller sub-tasks.

Since an entirely new course was designed, we considered collecting feedback essential. Each week ended with a feedback survey consisting of the following questions: 1) What did you learn this week?, 2) What things remain unclear after this week?, 3) How would you improve or change the material this week? The students were also asked to evaluate the difficulty level of exercises and to report how much time they used to complete the exercises. In addition, they were asked to provide a grade for the week. The feedback collection follows that of Kaila and Lokkila [17] and was included to help develop the materials further. The feedback data can also help teachers track students' skill levels and detect potential problems early.

3.3 Technical Implementation

The course was implemented on an automated assessment system *Test My Code* (TMC) [28]. The material was written in Markdown, and the text was accompanied by illustrations and figures. Numerous code examples were used throughout the material to demonstrate the topics. Very early on, we chose not to include any videos in the material. The decision was made based on the success of our earlier courses which also did not have videos. On the other hand, it takes a lot of time to produce high-quality videos, and we felt that the instructors' limited time was better spent improving the written material and guiding the students. The registration process of the course was streamlined only to include email and password, and even the opening survey was conducted quite late in the first week's material.

To test the correctness of students' solutions, unit tests were used. The tests use Python's built-in unit test module (see, e.g. [23]), and the student gets the point(s) from the exercise when all tests are passed. As an example, consider an exercise from week 1 of the course shown in Figure 1a. The unit test of each exercise tests the answer with multiple inputs. Figure 1b shows a buggy solution and error message for a test that provides the program with the string "5" as input and tries to ensure that the program prints out the expected result.

Compiler error messages or run-time exceptions often seem too vague or complex for beginners to interpret. For example, a missing parenthesis at the end of a code line can lead to the compiler reporting the error in the subsequent line. For more experienced programmers, these errors can be easy to detect, but for beginners, they can be complicated.

To further emphasize active learning, instant, and continuous, high-quality feedback is essential. So in addition to testing students' solutions with different kinds of input or parameter values, the tests are meant to catch typical errors and provide instructions on correcting these errors. As the example in Figure 1b shows, we tried to be as descriptive as possible in the error messages shown to students. In case of error, an error message is given that describes 1) the given input, 2) how the program responded, and 3) what was the expected outcome.

We wanted to make the threshold for starting the course as low as possible. Hence, for the first three weeks, the programming tasks are completed in a web browser, and the students do not need to install any additional programs or plugins to start working. After the first three weeks, the students start using an external editor. We selected Visual Studio Code (VSC) [11] as the editor for its flexibility and support for different languages and because it is used widely in the industry. To use the editor, a plugin was written for VSC. This plugin can download any week's exercises, run the tests locally, and submit solutions to the TMC system to get points.

Since detecting and fixing errors is a natural part of software development, it should also be emphasised in teaching. Because of this, the students are encouraged to test their solutions as often as needed without scoring penalties. The in-browser exercises enable the execution of code at any time. When the students are using VSC to write their code, they can additionally execute all tests locally whenever they want before submitting their solution. Since the tests are also provided locally, students can look at the test code and modify them locally if they want to. Hence, the whole testing procedure is made transparent to students.

3.4 Changes made during and after the initial pilot iteration

Creating an entirely new course involves several risks. Therefore we decided to pilot the first part of the course (first seven weeks) before launching the entire course to degree students. The pilot course was conducted during the pandemic, and all the instruction was given online. To eliminate the most apparent rough edges, the pilot course also had a handful of students for whom the material and exercises were released roughly one week before the release. The number of participants in the pilot course was 657.

The main issue in the pilot course was the quality of tests. The relatively liberal syntax and dynamic typing of Python allowed students to write some wrong solutions that caused extremely confusing error messages in tests. Students reported the confusing errors actively in the course chat. Our browser-based programming environment even has a button (see Figure 1b) that students can use to quickly share a failing solution with the course staff and fellow students. Most of the causes of confusion in tests were fixed immediately after they were reported.

The other major pain point noticed during the pilot was that students had problems debugging their code. Several sections about typical issues, debugging, and related problem-solving approaches were added to the material to remedy that. Some new material was written during the pilot course, and some more material was added later and was available when the first entire course iteration started. We also encouraged students heavily to use Python Tutor [6], an online service where one can visualize how their code is executed.

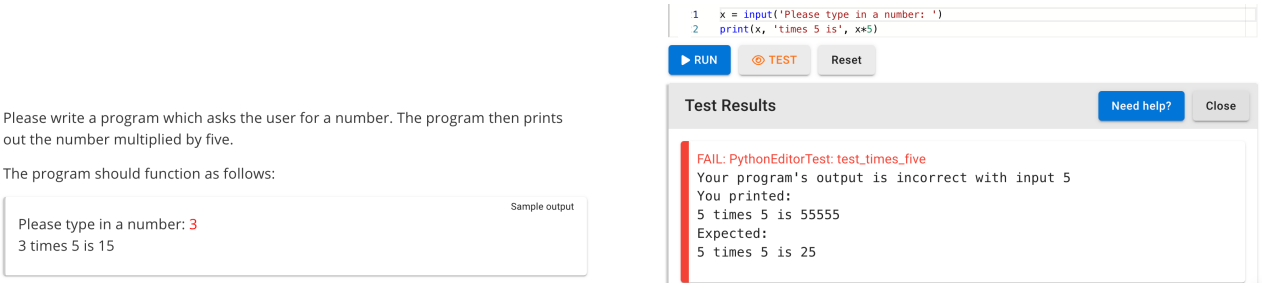
The first complete course iteration provided to degree students had the renewed first part and the entirely new second part. For the second part, we also had a handful of test students for whom the material and tests were released one week beforehand; the necessary fixes were done on the go. The process of producing high-quality tests had already improved thanks to the lessons learned during the pilot course, so the number of problems encountered was lower. Some new material sections were again needed and written during the course, e.g., material describing problematic issues concerning object orientation.

4 METHODOLOGY

Data Collection. In addition to the anecdotal evidence and observations by the course creators, we collected the following data to evaluate the success of our new course and its effect on students and the whole curriculum:

Performance data was collected as obtained scores, pass rates, and grades in the first two course implementations. We also looked at grades and pass rates of other related curriculum courses before and after the change. An *online survey* for course instructors was used to collect perceptions about differences between the courses. We were especially interested in what kind of questions the students asked in the Python course compared to the previous Java course. *Feedback* was collected at the end of each week and the end of both parts. We were particularly interested in students' estimates of difficulty level and grades for each course week.

Participants. We collected quantitative data on the open MOOC versions of the Java and Python programming courses (including university students). For the related courses, we analyzed the progress of five intakes of computer science (CS) majors, explicitly focusing on students who undertook their first programming course after starting their studies. To gather survey responses, we used an online form sent to the teaching assistant chat forum and distributed via email to teaching assistants who had instructed courses in Java and Python. Furthermore, we obtained feedback data from the initial instances of the Python MOOC. These sources of student demographics and feedback provided valuable insights for our analysis.



(a) An exercise from the first week of the course. **(b) A clear error message.**

Figure 1: Exercise from the first week of the course, completed in browser, and an example of the error message provided.

5 RESULTS

5.1 Grades and Pass Rates

Table 1 contains grade and pass rate data from four course instances. Two scheduled instances and two unscheduled instances of both Java and Python courses were selected. It should be noted that the instances were implemented as MOOC courses and anyone interested was allowed to access them. This explains the rather significant N of each instance. In scheduled instances, each week was open for a limited time (7 to 14 days) only, while in unscheduled instances, the students could complete the weeks throughout the year with no other deadlines.

Language	Year	Type	N	Pass rate	Mean grade
Java	2020	Scheduled	3972	31.62 %	4.44
Java	2020	Unscheduled	7148	9.72 %	4.51
Python	2020	Scheduled	945	33.18 %	4.46
Python	2021	Unscheduled	9998	11.41 %	4.67

Table 1: Comparison of Java and Python instances.

The pass rate and mean grade were marginally higher in the Python versions. Still, the differences are minor and there were many changes in the course content and exercises, so we cannot argue that the programming language was the reason behind the slightly better performance.

5.2 Course Data

The grade averages and pass rates of five courses from our core curriculum are displayed in Table 2 for CS major students who started in the years between 2017 and 2021. Three Java instances and two Python instances from each course are selected for comparison (the 2022 to 2023 results were not yet completed when this article was written). A student is regarded to have passed a course if they have a passing grade in the course within three first semesters of studies. Percentages of passing for some of the courses are relatively low. This does not necessarily mean the students would not have passed a course. Even though the courses are scheduled to be completed within the first three semesters, some students may not follow the model syllabus. The reasons for that vary; some may not be able to study full-time because of work, and some may need more practice in programming before moving on to the following courses.

5.3 Online Survey Data

A total of sixteen instructors answered our online survey. Thirteen of the respondents had instructed the *Introduction of Programming* course in Python and Java, and eight of those had also instructed another course that involved programming. Three respondents had not been instructed in the programming course (with two language versions) but had done that in some more advanced courses where programming plays a significant role. Let us next consider the feedback collected from the survey point by point.

Which version of the courses do you think students found more difficult? Why?

Ten out of sixteen respondents said that Java was more difficult for students, and the other six did not have an opinion. The following causes of the difficulty for Java were mentioned: heavier syntax (five mentions), objects (two mentions) and configuration (four mentions).

What were the typical problems encountered? How did they differ between courses?

Configuration issues were the most common problem for both languages. Java got nine and Python five mentions. The syntax in Java got only three explicit comments. Static typing in Java was pointed out as a specific cause of problems in four answers. For Python, the lack of static typing got one mention. Debugging problems got two mentions in Python and no mentions in Java.

Despite configuration issues getting quite many mentions, it was also raised that in Java, the dependency management (that is done with Maven [19] in our case) tends to be a bit more robust and causes, in general, less amount of problems compared to Python where students use Pip [1] and Poetry [2], often in quite a random manner.

The answers to this question partially overlapped with the answers to the previous question. Some respondents repeated the same responses in both questions (such as problems with syntax), but some did not do that.

What other differences were there in instructing the courses?

Only one instructor responded to this question and gave two quite enlightening points of view.

With Python, there are far more "basic questions" and problems with lists and dictionaries, for example. Students do not understand the difference between a reference and a value. The wrestling with these was less with Java, almost non-existent.

Starting year (N)	2017	(N=65)	2018	(N=59)	2019	(N=38)	2020	(N=67)	2021	(N=70)
Programming language	Java		Java		Java		Python		Python	
Introduction to Databases	80.0 %	4.20	85.1 %	4.40	81.6 %	4.26	77.6 %	3.50	94.3 %	3.36
Programming Techniques	56.9 %	3.57	55.9 %	3.45	68.4 %	4.31	50.8 %	3.35	61.4 %	3.33
Software Engineering	41.5 %	4.15	33.9 %	4.40	42.11 %	4.50	37.21 %	3.08	50.0 %	3.38
Database Lab	40.0 %	4.04	35.6 %	3.67	44.8 %	4.53	46.3 %	3.65	48.6 %	4.62
Introduction to University Math	86.2 %	3.80	91.5 %	4.60	84.2 %	4.80	95.5 %	4.00	90.0 %	4.00

Table 2: Pass rates and grade averages of core courses in the curriculum for CS major students who started in years between 2017 and 2021.

Another observation is that Python's more free syntax causes some confusion – error messages are more complex to understand.

Which course did you find more work to teach? Why?

Six respondents mentioned the Java version being more laborious to teach, while three said the Python version was more complicated. Two of the Python mentions were due to the instructor having less expertise with the language. The only language-related mention was the difficulty in debugging Python programs. The only other explicit statement was the configuration issues in Java, which three instructors voiced.

Based on your own teaching experience, which do you think is the better teaching language? Why?

Eight instructors said that Python is the better language. The most common argument was the simplicity of the syntax. It was also pointed out that Python is a universal language: it is much more applicable to projects outside the programming courses.

Four instructors preferred Java for teaching. Five instructors mentioned that Java gives a better foundation for learning other languages since it is, e.g., much more explicit with typing.

Two of the instructors mentioned that Java supports students to structure the code a bit better: the object-oriented paradigm is understood better, and the coding style conventions (rules about naming, preferred method length, etc.) are followed more closely.

5.4 Feedback Data

The feedback was collected in the pilot course and the first entire course iteration. The results are displayed in Table 3. For each week, the number of students replying is displayed. In addition, we show the mean values of perceived difficulty levels (on a scale of 1 to 7, where 7 is the most difficult) and the mean grades (again, on a scale of 1 to 7, where 7 is the best) awarded by students.

6 DISCUSSION

6.1 Interpretation of Results

The pass rates and grade averages from the first Python MOOC courses (see Table 1) seem to follow the trend set earlier in the Java MOOCs. Despite the Python versions having a slight edge in both pass rate and mean grade, there were no statistically significant differences in pass rates or grade averages when we compared the scheduled and unscheduled versions of Java and Python MOOCs. Hence, it seems that changing the programming language did not lead to a dramatic change in the measurable learning outcome of the introduction of the programming course. A more detailed analysis description can be found in (reference removed for reviewing).

In the courses following the introduction of programming (see Table 2), the CS major students who started their studies in 2020, the first Python generation, performed worse (lower pass rate and lower average grade) in several other courses. The only exception is *Introduction of University Mathematics*, which has hardly anything to do with selecting the first programming language. For the students who started in 2021, there is not a similar phenomenon visible, so it is possible that the COVID-19 pandemic caused the most problems for the 2020 batch. All the courses were online, and the usual in-campus instructions and peer support were missing. However, let's look at the average grades. There is a drop in three of the observed courses (*Introduction to Databases*, *Programming Techniques*, and *Software Engineering*) for the students who started in 2021. A possible explanation for the dropped grades in *Programming Techniques* and *Software Engineering* might be the less solid foundation in object orientation caused by the multi-paradigm Python course compared to the Java version with a strong focus on the object-oriented paradigm.

There is a notable drop in average grades in the *Introduction to Databases* course for the students who started in 2020 and 2021. When analyzing this further, the reason seems to be that there have been some changes in how the course is graded. The number of students who, instead of failing the course altogether, pass the course with a low grade has increased. The percentage of students in the 2021 cohort who passed the course is remarkably high compared to all the observed years.

The feedback analyzed (see Table 3) reveals a small but not significant correlation ($R = -0.2$) between the perceived difficulty level and the overall grade awarded for each week. Overall, it seems the students found the first part of the course more difficult than the second. There was minimal variation in grades for individual weeks: week 5 (where the topic was references and additional data structures) was the only one receiving an average grade less than 5 (4.9, to be specific). The grade for each week showed a slight improvement from the pilot iteration to the first entire course. This outcome was unsurprising, given that there were fewer rough edges after receiving feedback from the initial set of more than 600 students.

6.2 Experiences

The results of the instructor survey reveal that, as expected, most instructors preferred Python: it was easier to instruct and better for students as the first language. Quite a remarkable portion of problems attributed to Java was caused by the difficulties in configuring the programming environment, most notably the Java FX

		Pilot			First full version		
Week #	Topic	*N*	Difficulty	Grade	*N*	Difficulty	Grade
1	Input and output, variables, types	657	2.739	5.64	724	3.05	5.79
2	Conditional execution, simple loops	574	3.707	5.44	656	3.98	5.50
3	More about loops, simple functions	477	5.12	4.79	530	4.89	5.22
4	More about functions, strings, lists	404	5.09	4.86	454	4.90	5.32
5	References, dictionaries, tuples	321	5.48	4.78	405	5.60	4.90
6	Reading and writing files, errors, and exceptions	292	5.22	4.96	385	5.27	5.23
7	External modules	265	4.93	5.35	363	5.08	5.46
8	Objects and classes				448	3.41	5.96
9	Encapsulation, static members, object references				423	3.83	5.83
10	Inheritance, member visibility, larger applications				405	4.89	5.53
11	List comprehension, recursion				401	4.78	5.61
12	Functional programming, generators, regular expressions				383	4.78	5.65
13	Introduction to Pygame				370	4.84	5.42
14	Writing own game project				390	5.09	5.53
	Intro mean	427.10	4.61	5.12	502.43	4.68	5.35
	Advanced mean				402.90	4.52	5.65
	All mean				452.60	4.60	5.50

Table 3: Python course's weekly content topics and the difficulty and grade of each week in the pilot and the first full version of the course. The first seven weeks belong to the intro part of the course, and the last seven to the advanced.

UI library [3] to work in all the computer setups. So most of the burden Java caused for the instructor was *not* due to the language but the ecosystem's fragility.

There were also some conflicting reports about configuration problems. Some of the instructors stated that the package management systems of Python (mainly used in later courses) are a potential cause of problems in contrast to relatively robust solutions (maven in our case) used with Java. One of the authors of this paper, a lecturer in many courses where programming is applied, has witnessed the same to quite a large extent.

Instructors also saw many merits in Java: it gave a more versatile background for learning other languages, gave a more sound understanding of object-oriented programming, and forced students to achieve a more precise structure to the program. The reason for the latter also has to do with the course design. In the Python version, we emphasized object-oriented programming less than the Java version, which followed an object-first approach.

A weaker understanding of object-oriented programming is reflected in the course *Programming Techniques*, which students take in the semester following the programming course. During this course, students are developing a more extensive program on a topic of their choice. The course assessment focuses on several structural aspects of the code (e.g. separation of concerns). Both the grades (see Table 2) and anecdotal evidence by the course lecturer suggest that students who have the Python course as a background struggle a bit with the structural aspects of the code. It should be noted that this does not necessarily have anything to do with the programming language; it merely reflects the differences in the course design. In the Python course, we emphasized the multi-paradigm nature of Python, whereas in the Java course, a strict objects-first style was followed.

Before implementing the automated tests in the course, we assumed that testing Python exercises with the unit test library would be easier than testing Java exercises for which we used the JUnit [4] testing framework. For example, it is easy to check the existence of a Python class or method, while in Java, this should be done with the laborious Reflection API [5]. However, several things caused problems in testing Python. A significant difficulty is that a student can write arbitrary code to a file besides the function or class that is asked to be written by the exercise. This can cause confusing error messages even if the students' solution seems to work fine when they run it locally (for example, when a global variable is used instead of the function parameter).

6.3 Effect on other courses

The introductory programming courses do not take place in a vacuum; the chosen programming language significantly impacts other curriculum courses where programming is used as a tool. Our initial plan was to split the course into two parts. The first (11 out of 14 weeks) would be an introduction to programming with Python. During the last three weeks, the language was switched to Java. The motivation for ending the course with Java was twofold: firstly, it would act as a bridge to other courses where Java was still the primary language, and secondly, it would deepen the student's learning experience by giving an example of a language that uses a static type system.

The idea of having the bridging Java part was quickly abandoned, and it was decided that other courses would start supporting Python besides Java right after the first complete Python iteration of the introduction to programming. This was a significant investment in time since programming is vital to many other courses. Also, the support for two languages (Python and Java) that continued for some courses for four semesters caused an extra burden.

The effect of changing the primary programming language has been largely positive for most of the other courses. Thanks to the light syntax, Python is much more easily applicable than Java to topics such as web and database programming, algorithms, and data analysis. There have also been some drawbacks, e.g., in the

algorithm courses where the extensive standard library of Python sometimes hides what happens under the hood, and that can make it harder for students to analyze the time and space complexities of algorithms.

7 CONCLUSIONS

This paper discussed our experiences changing the curriculum programming language from Java to Python at the University of Helsinki, Finland.

The previous iteration of the introductory programming course, established in 2010, focused on preparing students for professional development careers, emphasizing object-first programming, industry-standard tools, and best practices.

In the past decade, programming has gained popularity beyond the scope of CS majors, attracting a broader audience through minor programs and MOOCs. Consequently, we aimed to design a course that caters to diverse needs. The course aimed to establish a strong programming foundation for CS majors across multiple paradigms. In contrast, for a broader audience, it aimed to provide practical programming skills for everyday applications like data analysis scripting.

The first part of the course focuses on procedural and imperative programming, emphasizing the creation of small scripts for practical scenarios. To reduce barriers to entry, we implemented a browser-based environment for the first three weeks of the course. The second part targets aspiring professionals and covers object-oriented and functional programming. To evaluate the success of the new courses, we relied on our observations as course creators and gathered feedback and data from various sources available. We found no significant differences in pass rates or grades between the Java and Python versions of the course, contrary to our initial expectations based on existing research. However, due to significant differences in content and exercises, definitive conclusions cannot be drawn from the data. Most teaching assistants preferred Python from both teaching and learning perspectives while recognizing the value of Java in building a foundation for other programming languages.

When analyzing the performance of CS majors in subsequent courses, we observed consistent pass rates regardless of the initial programming language. However, there were indications that the Python course may not adequately establish a robust foundation for courses emphasizing code's structural aspects. This discrepancy likely results from the altered content emphasis rather than the choice of programming language.

REFERENCES

[1] 2023. https://pypi.org/project/pip/
[2] 2023. https://python-poetry.org/
[3] 2023. https://openjfx.io/
[4] 2023. https://junit.org/
[5] 2023. https://docs.oracle.com/javase/tutorial/reflect/
[6] 2023. *Python tutor: Learn Python, JavaScript, C, C++, and Java.* pythontutor.com
[7] Martin Abadi, Luca Cardelli, Benjamin Pierce, and Didier Rémy. 1995. Dynamic typing in polymorphic languages. *Journal of functional programming* 5, 1 (1995), 111–130.
[8] Krishna K. Agarwal, Achla Agarwal, and M. Emre Celebi. 2008. Python Puts a Squeeze on Java for CS0 and Beyond. *Journal of Computing Sciences in Colleges* 23, 6 (jun 2008), 49–57.
[9] Andrey Bogdanchikov, Meirambek Zhaparov, and Rassim Suliyev. 2013. Python to learn programming. In *Journal of Physics: Conference Series*, Vol. 423. IOP Publishing, 012027.
[10] Stephen Cooper, Wanda Dann, and Randy Pausch. 2003. Teaching objects-first in introductory computer science. *Acm Sigcse Bulletin* 35, 1 (2003), 191–195.
[11] Alessandro Del Sole. 2021. Introducing Visual Studio Code. In *Visual Studio Code Distilled: Evolved Code Editing for Windows, macOS, and Linux*. Springer, 1–15.
[12] Charles Dierbach. 2014. Python as a first programming language. *Journal of Computing Sciences in Colleges* 29, 3 (2014), 73–73.
[13] John M. Hunt. 2015. Python in CS1 - Not. *Journal of Computing Sciences in Colleges* 31, 2 (dec 2015), 172–179.
[14] Ambikesh Jayal, Stasha Lauria, Allan Tucker, and Stephen Swift. 2011. Python for Teaching Introductory Programming: A Quantitative Evaluation. *Innovation in Teaching and Learning in Information and Computer Sciences* 10 (02 2011). https://doi.org/10.11120/ital.2011.10010086
[15] Wuxia Jin, Dinghong Zhong, Zifan Ding, Ming Fan, and Ting Liu. 2021. Where to start: Studying type annotation practices in python. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 529–541.
[16] Erkki Kaila. 2018. Utilizing educational technology in computer science and programming courses. *TUCS Dissertations* 230 (2018).
[17] Erkki Kaila and Erno Lokkila. [n.d.]. EVALUATION OF CONTINUOUS STUDENT FEEDBACK ON A LARGE COMPUTER SCIENCE COURSE. ([n. d.]).
[18] Theodora Koulouri, Stanislao Lauria, and Robert D. Macredie. 2015. Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM Transactions on Computing Education* 14, 4, Article 26 (dec 2015), 28 pages. https://doi.org/10.1145/2662412
[19] Mike Loukides. 2008. *Maven: the definitive guide : [everything you need to know from ideation to deployment].* O'Reilly, Beijing [u.a.].
[20] Linda Mannila, Mia Peltomäki, and Tapio Salakoski. 2006. What about a simple language? Analyzing the difficulties in learning to program. *Computer science education* 16, 3 (2006), 211–227.
[21] Bradley Miller and David Ranum. 2006. Freedom to Succeed: A Three Course Introductory Sequence Using Python and Java. *Journal of Computing Sciences in Colleges* 22, 1 (oct 2006), 106–116.
[22] Rance D. Necaise. 2008. Transitioning from Java to Python in CS2. *Journal of Computing Sciences in Colleges* 24, 2 (dec 2008), 92–97.
[23] Ashwin Pajankar. 2017. *Python Unit Test Automation: Practical Techniques for Python Developers and Testers.* Apress.
[24] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. 2007. A survey of literature on the teaching of introductory programming. *Working group reports on ITiCSE on Innovation and technology in computer science education* (2007), 204–223.
[25] Atanas Radenski. 2006. "Python first" a lab-based digital introduction to computer science. *ACM SIGCSE Bulletin* 38, 3 (2006), 197–201.
[26] Joel Spolsky. [n.d.]. The Perils of Java Schools. https://www.joelonsoftware.com/2005/12/29/the-perils-of-javaschools-2/
[27] Arto Vihavainen, Matti Paksula, and Matti Luukkainen. 2011. Extreme Apprenticeship Method in Teaching Programming for Beginners. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 93–98. https://doi.org/10.1145/1953163.1953196
[28] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding Students' Learning Using Test My Code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education* (Canterbury, England, UK) *(ITiCSE '13)*. Association for Computing Machinery, New York, NY, USA, 117–122. https://doi.org/10.1145/2462476.2462501