What Language? - The Choice of an Introductory Programming Language

Onyeka Ezenwoye
School of Computer and Cyber Sciences
Augusta University
Augusta, Georgia 30912
Email: oezenwoye@augusta.edu

***Abstract*—Quite a few educational programming languages have been developed to make programming easier to learn. These languages have seen very little adoption in academia. This raises questions about their suitability for their intended purpose as introductory programming languages. A lot of factors are taken into consideration in choosing an introductory programming language. This work presents the results of a survey of computer science degree programs across the United States. The survey shows not only which introductory programming languages are popular but the reasons behind their popularity. It shows the features that are considered important for an introductory programming language and how well the popular introductory languages support those features. This work shows that some popular educational programming languages do not support some of those features that are considered important.**

***Index Terms*—Computer, Programming, Language, Education, CS1, Pedagogy.**

I. Introduction

IT is the desire of many educational initiatives to engrain students with computer programming skills. The choice of a programming language with which to introduce programming is an important part of this process. At some point the decision must be made as to which programming language to use in the introductory programming course. The specific language however, is often viewed as secondary to core programming concepts. The general perception is that the language does not matter when it comes to teaching programming to beginners [1]. The overreaching philosophy is that the programming language is simply a tool to implement algorithmic problem solutions. And, that as languages become outdated, it is the ability to reason logically and program correctly that matters. This thought is excusable given the multitude of programming languages that exist and their rapid evolution.

The introductory programming language however has to play a pivotal role in students' ability to learn programming. It is difficult to imagine there would much interest in computing if programming had to be taught with punch cards [2]. High-level languages are designed for instructions to be expressed in a form that is closer to human language. This high degree of abstraction is intended to make programs easier for humans to understand, thereby making programming less error-prone, and programmers more productive [3], [4]. At a pedagogical level, high-level languages are very different, and they treat important concepts with varying degrees of complexity. The language does impact the programming [5], [6]. Faced with the knowledge that the choice of programming language does collide with the ease of learning for beginners, educators take various approaches in deciding which language to use.

Quite a few educational programming languages and graphical tools [7], [8], [9], [10], [11] have been created with the goal of making it easier for students to learn how to program. Despite works touting their benefits [12], [13], [14], these languages have seen very little use in academia [15]. This fact leads to the question as to what languages educators prefer and why? What do educators look for in an introductory programming language, and might it explain why these educational programming languages are not favored? This work uses a survey of computer science bachelor's degree programs in the United States to find out what programming languages are used for the introductory programming course, and the important factors that were considered in the decision. The survey is designed to show not only which introductory programming languages are popular but the reasons behind their popularity. It looks at the practical and pedagogical factors that play important roles in the decision to choose an introductory programming language, including the features of the language which is among important factors that should be considered when choosing a programming language. The features of a language can affect how complicated the code is. These features can make it difficult for a human to write correct, and maintainable code [3], [16].

The survey tries to understand, what features these institutions consider important for an introductory programming language to have. It looks at how well the most popular introductory languages support those features. An attempt is made to understand how well the educational programming languages and tools Jeroo [10], Alice [8], Scratch [7] and Squeak [11] that are designed for ease of learning, support those features that are considered important. Previous studies have looked are the popularity of programming languages in academia but not looked at what language features educators consider as most important for the introductory programming course. This work shows that these educational programming languages do not support some of the features that educators see as important in an introductory programming language. The lessons learned from this study will influence the design of future educational programming languages, as well as provide

978-1-5386-1174-6/18/$31.00 ©2018 IEEE

a guide on the choice of introductory programming language at institutions of learning.

The rest of this paper is structured as follows; Section II presents the approach of the survey and results. Section III discusses the findings of the survey. Conclusion is presented in Section V.

II. Method and Results

In the year 2013, we conducted a survey of 200 Computer Science programs across the United States. The survey aimed to find out what language was chosen by each program, and why. Given the vast number of Computer Science programs in the country, only ABET-accredited [17] programs where surveyed to focus on a more homogenous subset. The survey asked two questions; (1) What programming language was used for the introductory programming course, (2) What was the justification for the choice of programming language. This survey had a response rate of 45% and constituted a pretest for the next survey in 2018, of which this paper is about.

In January of 2018, we conducted a second survey of 496 computer science bachelor's degree programs at four-year institutions across the United States. The information gained from the 2013 survey formed the basis of the questions in this survey. With the recent survey, an attempt is made to understand the reasons behind the institution's choice of introductory programming language. The survey is designed to show not only which introductory programming languages are popular but also what features are considered important for an introductory programming language to have. The survey had a response rate of 31.25%, below are the questions in the survey and the raw data of the responses. An analysis of the data is presented in Section III.

Q1: What language is used for the introductory programming course in the Computer Science bachelor's degree at your institution?

TABLE I
Responses to Q1

Answer	%	Count
Java	41.94%	65
C	4.52%	7
C++	19.35%	30
Python	26.45%	41
C#	0.65%	1
Other	7.10%	11
Total	100%	155

Those respondents who chose "Other" for this question where given the option to provide some text. The responses were Typescript (1), Ada (1), Javascript (1), Scala (1), and Racket (1). A respondent indicated C or Java, while another indicated both Java and C++.

Q2: For how long has the programming language been in use for the introductory programming course?

TABLE II
Responses to Q2

Answer	%	Count
0 - 2 years	9.03%	14
3 - 5 years	23.87%	37
6 or more	67.10%	104
Total	100%	155

Q3: Which of these do you believe are difficult for beginners to learn in the introductory programming course? (multiple selection)

TABLE III
Responses to Q3

Answer	%	Count
Pointers	17.15%	101
Object-orientation	13.24%	78
Memory management	13.41%	79
Exception handling	10.53%	62
Method overloading	9.00%	53
Operator overloading	10.36%	61
Static data typing	3.57%	21
Dynamic data typing	8.32%	49
Functional decomposition	8.49%	50
Program compilation	1.53%	9
Repetition structures	4.41%	26
Total	100%	589

Q4: Which of these do you believe are important for an introductory programming language to support? (multiple selection)

TABLE IV
Responses to Q4

Answer	%	Count
Pointers	5.67%	39
Object-orientation	12.35%	85
Garbage collection	3.78%	26
Exception handling	7.27%	50
Method overloading	8.28%	57
Operator overloading	4.65%	32
Static data typing	11.48%	78
Dynamic data typing	6.54%	45
Functional decomposition	14.53%	100
Program compilation	9.01%	62
Repetition structures	16.42%	113
Total	100%	688

Q5: Which of these were factors in the decision to use the current introductory programming language? (multiple selection)

TABLE V
RESPONSES TO Q5

Answer	%	Count
Advice of an advisory board	5.95%	25
Ease of learning of programming	18.81%	79
Institutional tradition	8.57%	36
Job opportunities for students	14.76%	62
Popularity of the language in academia	13.10%	55
Faculty availability or scheduling constraints	5.00%	21
The features of the programming language	26.19%	110
The language for AP Computer Science	7.62%	32
Total	100%	420

Q6: How many more computer programming courses are required for Computer Science majors after the introductory programming course?

TABLE VI
RESPONSES TO Q6

Answer	%	Count
1	12.95%	18
2	28.78%	40
3	58.27%	81
Total	100%	139

Q7: Do all the required computer programming courses for Computer Science majors use the same programming language as in the introductory course?

TABLE VII
RESPONSES TO Q7

Answer	%	Count
Yes	23.19%	32
No	76.81%	106
Total	100%	138

Q8: Does the introductory programming course sometimes include students from programs other than Computer Science?

TABLE VIII
RESPONSES TO Q8

Answer	%	Count
Yes	97.81%	134
No	2.19%	3
Total	100%	137

Q9: Which of these appear in the title of the college (not the department) under which the Computer Science program is housed? (multiple selection)

TABLE IX
RESPONSES TO Q9

Answer	%	Count
Computing (or Computer)	7.44%	16
Engineering	26.05%	56
Sciences (or Science)	39.07%	84
Mathematics (or Math)	8.37%	18
Business	3.26%	7
Arts	15.81%	34
Total	100%	137

Q10: Is the Computer Science program accredited by ABET (Accreditation Board for Engineering and Technology)?

TABLE X
RESPONSES TO QUESTION Q10

Answer	%	Count
Yes	52.12%	71
No	47.79%	65
Total	100%	137

Q11: Have you at any time been an instructor for the introductory programming course?

TABLE XI
RESPONSES TO Q11

Answer	%	Count
Yes	75.18%	103
No	24.82%	34
Total	100%	137

III. ANALYSIS

The survey shows that the choice of language falls mainly between three general-purpose languages, namely, Java, Python, and C++. Table I, shows the distribution of the choice of introductory programming language. Java is the most popular with 41.94% of respondents picking it as their teaching language. This is followed by Python, and C++ as the third most popular. Of note is that the popularity of the languages used in introductory courses does not quite mirror the general popularity of programming languages in use. Table XII shows current rankings, according to the Tiobe index, an indicator of the popularity of programming languages [18].

With the arrival of the internet, Java was created out of the need to develop platform-independent programs. Java was quickly adopted by many academic institutions as a teaching language [19]. Java remains popular for teaching introductory programming. Tables XIII and XIV show the duration of use of the chosen language as indicated by the respondents. Most responding institutions (104) have been using their language

TABLE XII
GENERAL POPULARITY OF PROGRAMMING LANGUAGES

Programming Language	Rank
Java	1
C	2
C++	3
Python	4
C#	5
Visual Basic .Net	6

for 6 or more years, and most of those (55.77%), use Java (Table XIII).

Part of the popularity of Java as an introductory language is due to the fact that Java is the language of choice for high school Advanced Placement Computer Science (APCS) [20]. Some of the survey respondents have indicated APCS as one of the reasons for choosing Java (Table V). Java is also seen as having pedagogical advantages over other languages. These advantages include its documentation system and vast array of libraries that allow for development in many application domains. As more applications are being created by gluing together existing components, the Java system is seen as broad enough to permit this style of programming [16], [5]. Interestingly, the data shows that most institutions (57.14%) that have been using a language for 2 years or less, use Python (Table XIV). This could indicate a change in trend towards Python.

TABLE XIII
DURATION OF USE OF CHOSEN LANGUAGE

No. of Years	Java	C	C++
0 - 2 years	7.14%	0.00%	14.29%
3 - 5 years	16.22%	5.41%	24.32%
>= 6 years	55.77%	4.81%	18.27%

TABLE XIV
DURATION OF USE OF CHOSEN LANGUAGE (CONTINUED)

No. of Years	C#	Python	Other	Total
0 - 2 years	0.00%	57.14%	21.43%	14
3 - 5 years	2.70%	40.54%	10.81%	37
>= 6 years	0.00%	17.31%	3.85%	104

The languages reported in our survey are structured and procedural. Structured programming makes use of control structures such as subroutines and repetition statements. These control structures are preferable to languages that rely on jump statements such as *goto* that make it difficult to write maintainable code [21]. Procedural programming is a form of structured programming where problems are decomposed as a set of procedures, methods or functions. The most popular languages from the survey (Java, Python, and C++) are also object-oriented. As discussed later in this section, object-orientation is one of the features the survey shows as important in an introductory language.

General-purpose programming languages are all designed to implement technical solutions to real-world problems. The difference comes in the syntax, semantics, and features of the language [3]. It is from these that a substantial part of the difficulty in learning to program arises. The basis for comparison of languages does span the theoretical to the practical [6]. This is evident from the responses when asked for the reasons for picking a specific language (Table V). Not surprising, the features of the programming language was picked by most as the reason for choosing a language (26.19%). Ease of learning of the language was in second place (18.18%). This suggests that although ease of leaning is important to educators, it is not as important as the features of the language. This also leads to the question, what language is the most popular for those respondents who indicated that ease of learning was a factor in choosing a language? Tables XV and XVI show the languages chosen by respondents against the reasons they specified as a factor in picking the language (i.e., Table V). As can be seen in Table XV, Python is the most popular language (44.30%) for those who specified ease of learning as a reason for choosing a language.

TABLE XV
REASONS FOR CHOSING A LANGUAGE, GROUPED BY CHOSEN LANGUAGE

Language	AB	Learning Ease	Tradition	Jobs
Java	60.00%	37.97%	52.78%	58.06%
C	0.00%	1.27%	5.56%	1.61%
C++	32.00%	7.59%	30.56%	29.03%
Python	8.00%	44.30%	2.78%	6.45%
C#	0.00%	1.27%	0.00%	0.00%
Other	0.00%	7.59%	8.33%	4.85%

TABLE XVI
REASONS FOR CHOSING A LANGUAGE, GROUPED BY CHOSEN LANGUAGE(CONTINUED)

Language	Popularity	Faculty	Features	APCS
Java	47.27%	61.90%	38.18%	87.5%
C	0.00%	4.76%	5.45%	0.00%
C++	20.00%	23.81%	19.09%	6.26%
Python	29.09%	0.00%	29.09%	0.00%
C#	0.00%	0.00%	0.91%	0.00%
Other	3.65%	9.52%	7.27%	6.25%

Other reasons for choice of language include job opportunities for students, faculty and scheduling constraints, institutional tradition, and the influence of a program advisory board. ABET-accredited programs typically have program advisory boards that are comprised of stakeholders, which might include local employers and alumni. Table XV shows that for most of those who specified the influence on an advisory board (AB) in selecting a language, Java is the chosen language (60%). 52.21% of respondents are ABET-accredited programs (Table X). The data does not show a significant difference in choice of languages compared to non-ABET-accredited programs. There is a slightly higher preference for Python for non-ABET-accredited programs as opposed to Java (Table XVII).

The survey asked respondents if they had served as the instructor for the introductory course. The goal is to see if there was any difference in responses from those who had to those who hadn't. 75% of respondents indicated that they had

TABLE XVII
LANGUAGE CHOICE BY ABET ACCREDITATION STATUS

Language	ABET	Count	Non ABET	Count
Java	45.07%	32	35.38%	23
C	8.45%	6	1.54%	1
C++	18.31%	13	18.46%	12
Python	23.94%	17	32.31%	21
C#	0.00%	0	1.54%	1
Other	4.23%	3	10.77%	7

(Table XI). There were no significant differences between the two groups.

Some college programs in computing do include, as an objective, student exposure to more than one programming language since languages all have a variety of relative strengths and weaknesses. This survey does attempt to find out if there are more programming courses after the introductory course (Table VI). It also asks if those courses all use the same language (Table VII). The results show that there is reasonable programming language flexibility in each program beyond the introductory course.

A. What language features are important?

Although the study of language details is seen as detrimental to learning for beginners [6], [5], language features do come into consideration when selecting a language for teaching, as the survey shows. These features cover concepts that programmers learn and can be significant as the evolution of languages has been driven by the need to address the limitations of existing languages.

The five most popular language features survey respondents noted as important are repetition structures (16.42%), functional decomposition (14.51%), object-orientation (12.35%), static data typing (11.48%), and program compilation (9.01%) (Table IV). When asked which features are difficult for beginners to learn, the five most popular features selected are pointers (17.15%), memory management (13.41%), object-orientation (13.24%), exception handling (10.53%), and operator overloading (10.36%) (Table III). An explanation of these language features is presented in section III-B below. Figure 1 offers a comparison of the most import language feature against those considered most difficult for beginners. Objection-orientation is perceived as both important and difficult for beginners.

B. Do the most popular languages support the most important features?

Tables XVIII, XIX and XXI present a comparison of the four most popular languages against the features identified above, to show their representation across the languages.

In table XVIII, the approximate year of inception of each language is shown to provide some context on programming language evolution. From the survey results, Python is the second most popular language but is the only one of the group that permits dynamic typing (non-static). Python's typing systems allow for flexible typing of data. Typing systems provide rules for data type definition and equivalence. A language

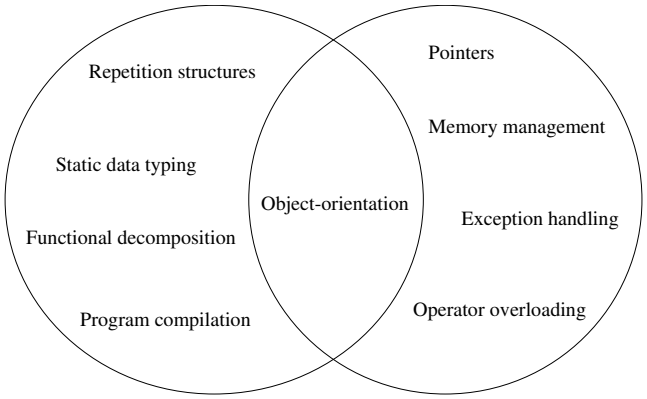

Fig. 1. Most important features versus most difficult features

TABLE XVIII
COMPARISON OF MOST POPULAR LANGUAGES

Language	Inception	Repetition	Static Typing	Functions
C	1971	yes	yes	yes
C++	1979	yes	yes	yes
Java	1991	yes	yes	yes
Python	1991	yes	no	yes

supports static data typing if it enforces typing rules at compile time as opposed to runtime (dynamic typing). Static typing provides the compile-time safety that isn't available with dynamic typing [3]. Static typing helps with the understanding of data abstractions, and statically typed languages do enforce programming discipline [16]. The survey data shows that static typing is seen as a more important feature than dynamic typing (Table IV). The data also shows that dynamic typing is seen as more difficult for beginners than static typing (Table III).

TABLE XIX
COMPARISON OF MOST POPULAR LANGUAGES

Language	Compiler	Objects	Pointers	Garbage collection
C	yes	no	yes	no
C++	yes	yes	yes	no
Java	yes	yes	no	yes
Python	no	yes	no	yes

Python is also the only language from the group that does not have a compiler (Table XIX). In a compiled language, the source code is checked and converted to machine code by a compiler program. This conversion happens prior to runtime, the machine code can then be distributed for execution. The source code at times needs to be recompiled when the execution platform is different. The process of compiling helps ensure attention to data abstractions, and type-safety. It also provides pre-execution error checking. Survey respondents indicated that having a compiler is an important feature for an introductory programming language (Section III-A). In contrast to a compiled language, in an interpreted language, the source code is checked at runtime or just before runtime by the runtime environment. This is without the need for compilation. Because of the late parsing, interpreted languages, sometimes known as scripting languages, may suffer some performance drawbacks, in comparison to compiled languages.

Scripting languages such as Python, emphasize flexibility and expressiveness in language semantics. However, they might not facilitate programming discipline [16].

The idea of object-oriented programming found promise in the desire for the development of flexible, modular and reusable programs. In object-oriented programming, programs can be decomposed into a set of interacting objects where each object can encapsulate its own state. The procedural programming language C is the only language in the group of the most popular that does not support object-orientation (Table XIX). The C language was extended to include object-oriented features, this extension gave rise to C++ [3]. Although with object-orientation, C++ is much like C and thus supports a mix of programming paradigms in the same program source code. This mix of paradigms is not advisable at the introductory level [22].

Unlike C++, Java is a pure object-oriented language where source code is a collection of classes. This behavior forces both students and teachers to think of problem decomposition in terms of class abstractions [19]. There is a debate on object-oriented versus procedural programming as the preferred approach for beginners. Some views are that it is difficult for students to understand object-orientation without prior programming knowledge [5]. Others claim that students encounter problems in switching from procedural programming to object-orientation, but not vice versa [22]. Perhaps the answer lies in the application domain, and memory limitations of the execution platform. For some applications on memory-sparse platforms, the procedural approach presented by C may be preferable to the object-oriented model [19]. Meaning that C is preferable for the low-level programming that is needed for embedded system. Embedded systems applications are more common in engineering disciplines such as Electrical Engineering. Table XX shows that of the respondents who use the C language, most of them (54.55%) are in a college with Engineering in its name (Table IX). Note that for brevity, data for Math, and Business have been removed from Table XX.

TABLE XX
LANGUAGE CHOICE BY COLLEGE NAME

Language	Computing	Engineering	Sciences	Arts
C	9.09%	54.55%	27.27%	9.09%
C++	12.24%	24.49%	38.78%	14.29%
Java	7.59%	27.85%	39.24%	15.19%
Python	3.51%	21.05%	40.35%	17.54%

C and C++ allow for low-level manipulation of memory through the use of pointers (Table XIX). A pointer is a variable whose value is a reference to some memory location. They allow for memory data to be accessed directly by the programmer. Pointer arithmetic permits for direct traversal of array elements. The compiler, however, is not required to check that the element is within the bounds of the array [3]. Pointers and arrays are often interchangeable, and this confuses the beginning programmer. Another problem with the use of pointers is the unintended overwriting of memory which corrupts programs. Debugging of pointer errors is often problematic for beginners, which is consider a pedagogical weakness [6], [19]. The survey suggests that pointers are difficult for beginners (Figure 1).

Operations on data in programs tend to cause the allocation of memory for the storage of data objects. As the program runs, deallocation needs to happen for unused memory objects otherwise the program may fail as it runs out of memory. This is phenomenon is known as a memory leak and can be observed especially in long-running programs. In the C and C++ programming languages, deallocation of memory objects that are no longer needed in the program must be specified explicitly by the programmer. Thus, the programmer needs to pay attention to deallocation operations. In Python, and Java, deallocation is performed implicitly. That is, the run-time environment will reclaim memory objects when they are no longer reachable within the program. This behavior is known as garbage collection [3]. Garbage collection frees the programmer up from specifying deallocation tasks, thereby simplifying the programming and reducing the risk of memory leaks and accidental deallocation. Java and Python, the two most popular languages in our survey support garbage collection as feature (Table XIX). This feature helps with memory management which is considered difficult for beginners (Figure 1).

TABLE XXI
COMPARISON OF MOST POPULAR LANGUAGES

Language	Operator overloading	Exception handling
C	yes	no
C++	yes	yes
Java	no	yes
Python	yes	yes

In some languages, ad hoc polymorphism for methods and operators is provided through a technique called overloading. In overloading, an identifier or symbol may have more than one meaning or refer to more than one object in a specific context. In C and C++, most operators including arithmetic operators can to extended so that the behavior is dependent on the arguments used. Some programming languages allow for methods to be overloaded so that more than one subroutine may share the same identifier. Overloading provides a lot of flexibility for the programmer but tends to make languages more complex for beginners, especially when operators are overloaded [19]. This fact is also captured by the survey (Figure 1). Java, the most popular language, does not support operator overloading (Table XXI).

Programming languages are notorious for issuing errors messages that are ambiguous and confusing for beginners. Exception-handling provides mechanisms for detecting and managing runtime errors. Languages such as C that just generate error codes, require some level of expertise to understand. Some languages are better than others at managing errors in the sense that they permit the extension of existing error codes in the form of user-defined exceptions and exception handling. This feature does lead to the display of more useful error messages that make debugging easier for beginners, but sometimes at the expense of less complex code [6]. The three most popular languages in the survey do provide some

support for exception handling (Table XXI). The survey shows exception handling as one of the most difficult for beginners (Figure 1)

C. Do educational programming languages support the most important features?

Covered in this section are languages that were designed for ease of learning of programming [10], [13], [14], [11]. These languages which emphasize simplicity, often come with a visual programming environment that has animated sprites. The environments show varying degrees of code and attempt to overcome the syntax hurdle for beginners. These languages however, see very little use outside of their originating institutions [15], a fact that is supported by this survey as none of the responding institutions indicated use of these languages. Compared here is how well these languages support those features that the survey data captures as the most important for introductory programming.

TABLE XXII
COMPARISON OF EDUCATIONAL PROGRAMMING LANGUAGES

Feature	Jeroo	Alice	Scratch	Squeak
Repetition	yes	yes	yes	yes
Static Typing	no	no	no	no
Functions	yes	yes	no	yes
Compiler	no	no	no	no
Objects	yes	yes	yes	yes

All the languages incorporate some control statements that allow for the repetition of instructions. They also all have some facilities for the instantiation of objects. Jeroo [10], Alice [14] and Scratch [13] allow the user to create objects from an existing collection of objects. These objects have predefined properties through which the object's state can be modified. This form of object use is somewhat restrictive as the user is not able to define new classes.

None of the languages allow for the static typing of data, nor do they have compilers, two of the most important features that our respondents indicated in the survey (Section III-A). Based on their support for these features, it should be noted that these languages are most similar to Python, the second most popular language in the survey (Table I). As discussed previously in Section III, Python is the language most chosen by respondents who indicated that ease of learning was a factor in picking a language (Table XV). Table XXIII shows those features respondents have indicated as important, grouped by the language used. Surprisingly, Python users chose the same features at a similar rate as Java users, including static typing and the need to have a compiler. There is however a slight preference for dynamic typing amongst Python users. Python is a popular language in industry (Table XII), unlike the educational languages discussed here.

The initial programming course should introduce students to reasoning about formulating solutions to problems. These solutions must then be expressed as algorithms, and finally these algorithms implemented in a programming language. All the parts of this process are important. The problem solving and algorithm construction are indeed language-independent.

TABLE XXIII
IMPORTANCE OF FEATURE, GROUPED BY CHOSEN LANGUAGE

Feature	Java	Python	C++
Pointers	4.20%	5.11%	7.53%
Object-Orientation	15.03%	11.36%	10.96%
Garbage collection	4.20%	3.98%	2.74%
Exception handling	8.04%	7.95%	6.16%
Method overloading	9.09%	7.95%	8.90%
Operator overloading	5.24%	1.70%	6.16%
Static data typing	10.84%	10.80%	10.96%
Dynamic data typing	5.94%	8.52%	5.48%
Functional decomposition	12.94%	16.48%	13.70%
A compiler	8.74%	7.39%	12.33%
Repetition structure	15.73%	18.75%	15.07%
Total	100%	100%	100%

Expressing the algorithm, is not language-independent. Although the problem-solving is the most important objectives in a first programming course, its achievement is leveraged upon the ability to express the solution in a programming language [30]. In an introductory programming course, they should be taught in an integrated and wholistic manner. The programming language serves as the tool through which the problem solving is realized. Educational programming languages however, in an attempt to get begginers over the syntax hurdle, minimize the language-independent aspects of programming.

IV. RELATED WORK

This work attempts to shed some light on the various reasons behind the choice of an introductory programming language, it does not delve into the ease of assimilation of each language by beginners. Different pedagogical approaches are also not assessed. Some works have addressed these issues [1], [5], [22], [19], [23], [24], [25], [26].

Rabai et. al. [15] present the results of their 2010 and 2013 surveys on the adoption of programming languages in academia and industry. Their survey focuses on general adoption and not use in the introductory course. They also do not consider the reasons for the adoption.

Koorsse et. al. [27] conduct a review of the programming tools including Scratch [13], to assess their suitability for introducing programming to IT students at the secondary school level in South Africa. Kereki [28] looks at the use of Scratch to teach CS1 at Uruguay University. Also discussed is the effectiveness of Alice [14], and Jeroo [10]. These works, amongst others [29] focus on specific programming tools. Results on their effectiveness vary greatly and are at times contradictory.

V. CONCLUSION

This work showed the most popular languages for introductory programming as well as which language features are considered important for an introductory programming language to support. The most important features include a static typing system, object-orientation, and the use of a compiler. Some educational programming languages and tools that are designed for ease of learning of programming, are compared

against these features. The results show that the educational programming languages do not support the static data typing and compile-time type-checking that is seen as important for introductory programming courses. These languages focus more on problem solving and algorithm construction and less on the language-independent part of expressing the algorithm. The design of educational programming languages should leverage and provide support for the important features of existing popular languages that educators most desire. It would also be worthwhile to provide some flexibility in their configuration to allow for the educator to select what features to incorporate in the introductory course.

REFERENCES

[1] E. Giangrande, "CS1 programming language options," *Journal of Computing Sciences in Colleges*, vol. 22, no. 3, pp. 153–160, 2007.

[2] D. Fisk, "Programming with punched cards," 2005, http://www.columbia.edu/cu/computinghistory/fisk.pdf, Retrieved June, 2018.

[3] M. L. Scott, *Programming Language Pragmatics*, 3rd ed. Morgan Kaufmann, 2009.

[4] P. Deitel and H. Deitel, *Visual C# 2012 How to Program*, 5th ed. Prentice Hall, 2013.

[5] K. Dai, Y. Zhao, and R. Chen, "Research and practice on constructing the course of programming language," in *IEEE International Conference on Computer and Information Technology*, 2010.

[6] L. McIver and D. Conway, "Seven deadly sins of introductory programming language design," in *International Conference on Software Engineering: Education and Practice*, 1996.

[7] D. J. Malan and H. H. Leitner, "Scratch for budding computer scientists," in *38th SIGCSE Technical Symposium on Computer Science Education*, 2007.

[8] S. Cooper, W. Dann, and R. Pausch, "Alice: A 3-D tool for introductory programming concepts," *Journal of Computing Sciences in Colleges*, vol. 15, no. 5, pp. 107–116, 2000.

[9] K. Dai, Y. Zhao, and R. Chen, "Greenfoot: A highly graphical ide for learning object-oriented programming," in *13th Annual Conference on Innovation and Technology in Computer Science Education*, 2008.

[10] D. Sanders and B. Dorn, "Jeroo: A tool for introducing object-oriented programming," in *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '03. New York, NY, USA: ACM, 2003, pp. 201–204.

[11] D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, and A. Kay, "Back to the future: The story of squeak, a practical smalltalk written in itself," in *Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. New York, NY, USA: ACM, 1997, pp. 318–326.

[12] P. Mullins, D. Whitfield, and M. Conlon, "Using alice 2.0 as a first language," *Journal of Computing Sciences in Colleges*, vol. 24, no. 3, pp. 136–143, 2009.

[13] U. Wolz, H. H. Leitner, D. J. Malan, and J. Maloney, "Starting with scratch in cs 1," in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2009, pp. 2–3.

[14] K. Powers, S. Ecott, and L. M. Hirshfield, "Through the looking glass: Teaching cs0 with alice," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2007, pp. 213–217.

[15] L. Ben Arfa Rabai, B. Cohen, and A. Mili, "Programming language use in us academia and industry," *Informatics in Education*, vol. 14, pp. 143–160, 10 2015.

[16] D. Spinellis, "Choosing a programming language," *IEEE Software*, vol. 23, no. 4, pp. 62–63, 2006.

[17] "ABET - Accreditation Board for Engineering and Technology," http://www.ABET.org, Retrieved November, 2016.

[18] "The TIOBE programming community index," https://www.tiobe.com/tiobe-index/, Retrieved May, 2018.

[19] T. Minor and L. Gewali, "Pedagogical issues in programming languages," in *International Conference on Information Technology: Coding and Computing*, 2004.

[20] "AP Computer Science," http://apcentral.collegeboard.com, Retrieved July, 2018.

[21] W. A. Wulf, "A case against the goto," in *Proceedings of the ACM annual conference*, vol. 2. ACM, 1972, pp. 791–797.

[22] M. Saeli, J. Perrenet, W. M. G. Jochems, and B. Zwaneveld, "Teaching programming in secondary school: A pedagogical content knowledge perspective," *Informatics in Education*, vol. 10, no. 1, pp. 73–88, 2011.

[23] T. Koulouri, S. Lauria, and R. D. Macredie, "Teaching introductory programming: A quantitative evaluation of different approaches," *ACM Transactions on Computing Education*, vol. 14, no. 4, pp. 26:1–26:28, Dec. 2014.

[24] A. Vihavainen, J. Airaksinen, and C. Watson, "A systematic review of approaches for teaching introductory programming and their influence on success," in *Proceedings of the Tenth Annual Conference on International Computing Education Research*. New York, NY, USA: ACM, 2014, pp. 19–26.

[25] G. Silva-Maceda, P. David Arjona-Villicana, and F. Edgar Castillo-Barrera, "More time or better tools? a large-scale retrospective comparison of pedagogical approaches to teach programming," *IEEE Transactions on Education*, vol. 59, no. 4, pp. 274–281, Nov. 2016.

[26] Y. Qian and J. Lehman, "Students' misconceptions and other difficulties in introductory programming: A literature review," *ACM Transactions on Computing Education*, vol. 18, no. 1, pp. 1:1–1:24, Oct. 2017.

[27] M. Koorsse, C. Cilliers, and A. Calitz, "Programming assistance tools to support the learning of it programming in south african secondary schools," *Computer and Education*, vol. 82, no. C, pp. 162–178, Mar. 2015.

[28] I. Friss de Kereki, "Scratch: Applications in computer science 1," in *Proceedings of the Frontiers in Education Conference*, 11 2008.

[29] A. V. d. A. Leal and D. J. Ferreira, "Learning programming patterns using games," *International Journal of Information and Communication Technology*, vol. 12, no. 2, pp. 23–34, Apr. 2016.

[30] I. Utting, S. Cooper, M. Kolling, J. Maloney, and M. Resnick, "Alice, greenfoot, and scratch – a discussion," *ACM Transactions on Computing Education*, vol. 10, no. 4, 2010.

-