International Journal of Recent Advances in Multidisciplinary Topics
Volume 2, Issue 10, October 2021
https://www.ijramt.com | ISSN (Online): 2582-7839


# Why so Many Programming Languages

Sparsh Mehra*

***Abstract*: Programming is a basic course that is instructed to each computer science during their underlying semesters. The course acquaints the understudies with nuts and bolts tasks and engineering of computers, and furthermore cleans the critical thinking abilities of understudies. Other than these benefits, programming language fills in as a basic device for examining, studying and understanding progressed ideas of computer science that the understudies are instructed in later semesters of their undergrad contemplates. Subsequently, the determination of a programming language for teaching to computer science understudies is critical. During the beyond couple of years, there have been various programming languages developed, for example, COBOL, FORTRAN, Algol, Miranda, Oberon, Ada and Java and so forth With the progression of time, some these languages have lost unmistakable quality while a few new languages have arisen. Hence, the determination of a programming language for teaching has consistently stayed a significant examination question for academicians. In this paper, a comparative investigation of contemporary programming languages is performed. After a cautious assessment of current educational plan and market requests, we have chosen C/C++, C#, Java, Pascal, GW Basic and JavaScript for examination. The target of this study is to figure out which programming language languages ought to be educated to computer science understudies at introductory level. The paper investigates the chose programming languages dependent on various boundaries and gives suggestions on the choice of programming language.**

***Keywords*: Why, so, many, programming, languages**

## 1. Introduction

Basically, the justification for why there are so many programming languages is on the grounds that every language's plan includes compromise. Engineers need a language that is not difficult to work with, exceptionally preformat, and uphold the provisions they feel are generally helpful. Nonetheless, not all are conceivable at the same time, so language originators need to pick which to focus on. In this manner, few out of every odd language can fulfill each engineer's inclinations. Luckily, the availability and development of current language creation apparatuses (lexers, parsers, compilers, transpilers, and translators) makes it simpler for designers to make new languages explicit to their own necessities. In any case, the resultant excess of new programming languages has not been met with a fast cease to exist of more seasoned programming languages. Pillars like C/C++, Java and PHP are above and beyond 20 years of age and are giving no indications of dialing back, because of their unwavering quality and solid local area support. Considerably more established languages like Fortran, a language previously delivered in 1957, still see successive use in government and monetary frameworks, where the expense of reworking such gigantic, crucial programming in another language offsets the advantages. Obviously, these languages have upsides and downsides. In this article, we'll talk about a portion of the reasons there are so many programming languages to look over. There are more computer languages in presence than anyone knows, and even more continue to get made each year. There isn't actually any incredible motivation to continue making more languages, on the grounds that current languages are satisfactory to accomplish any assignment we can imagine Mostly individuals simply make new languages since they can, and from time to time someone fosters a language that is adequate that a many individuals begin utilizing it.

That doesn't imply that disliked languages are terrible. Some generally excellent languages are not exceptionally well known by any means. A few languages are as yet being used however commonly viewed as old. At the point when a language becomes outdated, that implies it is basically dead, in light of the fact that no new applications will be made with it. So the main motivation behind why there's a ton of languages is on the grounds that individuals continue to make them, in any event, when they don't actually have to. Another explanation is on the grounds that a few languages are greatly improved fit to specific errands than others. Some programming languages are additionally a lot simpler to learn than others.

## 2. Compiled versus Interpreted Programming Languages

Maybe the most powerful choice in a language's plan is whether it will be compiled or interpreted. Compiled languages, like C++, Go, and Rust, are converted into machine code early, yielding a streamlined executable document. Interpreted languages like Python, JavaScript, and Ruby are meant machine code at runtime.

Compiled languages will in general be quicker than interpreted ones. Since the most common way of changing source code over to machine code doesn't have to happen as fast, compilers can present advancements that mediators can't. In any case, this implies that designers utilizing compiled languages should stand by longer than interpreted language clients between rolling out an improvement in code and testing the program. Another distinction is that compiled languages will in general be statically composed (every factor holds a predefined type like an Integer or String, known early), while interpreted languages are ordinarily progressively composed (a

*Corresponding author: mehrasparsh671@gmail.com

variable can store a worth of any sort). Static composing can be viewed as excessively prohibitive by certain engineers, however it diminishes the quantity of blunders that can happen at runtime, since most befuddling type mistakes will be gotten at incorporate time. For instance, endeavoring to gather a number and a memory address into a single unit will cause a runtime blunder in a progressively composed language, however a statically composed language can distinguish and caution about this mistake before program execution. Taking care of these sorts of choices is essential for the explanation such countless new languages have been made over the long run. Designers requiring the quickest program execution might partake in the simplicity of programming in Python, yet that language's interpreted nature restricts its presentation contrasted with compiled languages like C++. This Separation has prompted the production of Nim, a statically-composed, compiled language with Python-like syntax and components, among different languages.

## 3. Syntax

One more vital part of making a programming language is the syntax. At the most minimal level, computer preparing chips comprehend machine code - twofold guidelines comprised of examples of ones and zeroes. Hypothetically people could compose programs thusly, however it would be extraordinarily illogical. Low level computing construct is a stage higher up in the chain, permitting a negligible arrangement of intelligible watchwords that can be utilized to compose code. Albeit some extremely specialized specialists can code thusly, by far most of people need something better. More elevated level programming languages like C, C++, Java, Python, and Javascript give a hearty arrangement of receptive catchphrases that make programming considerably more available to an expansive crowd. This decreases the expectation to absorb information and permits languages to acquire foothold develop their networks. In any case, even a hearty arrangement of catchphrases can be furnished with shifting syntax to attempt to further develop designer encounters. For instance, a few languages use semicolons to isolate proclamations, and enclosure to embody blocks/scopes. Here is one more table with regards to how simple it is for fledglings to gain proficiency with every language:

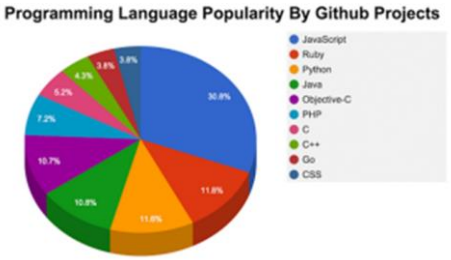
Relaxed to absorb somewhat interesting Very hard
BASIC C (and C++) Fortran
Xojo PHP Ruby
Python JavaScript Ada
HTML Pascal Java

| | | |
|---|---|---|
| CSS | SQL | Perl |

And, lastly, this table displays how numerous abilities decode into career occasions:

Ability Employ Base Wage Base
Ada very low high
ASP very low low to mid
BASIC very low low
C / C++ very high very high
Fortran very low high
HTML & CSS very high low
Java high high to very high
JavaScript high high
Pascal (and Delphi) very low very low
PHP & SQL high mid to high
Python mid mid to low

| | | |
|---|---|---|
| Ruby | low | very high |




Utilizing these tables will make it somewhat simpler to pick the language that you feel is ideal for you. That is vital, that you ought to pick it yourself dependent on your own measures, and not what another person believes is best for you. Learning ought to be fun, and attempting to learn something you don't actually like is normally not going to give the best outcomes.

*1) Models for Comparison*

The models for correlation depend on the reaction of different computer science instructors to embrace a specific language for teaching. We chiefly embrace the measures gave in [6] and [5]. Following models have been considered for assessment of language: Effortlessness: In request to be educated as introductory level course, a language ought to be exceptionally basic and nearer to regular language. A language nearer to human language is intelligible and simpler to be perceived by a beginner client.

*2) Writability*

A language ought to give an extensive arrangement of develops and APIs to be helpful for universally useful just as explicit programming errands.

*3) Dependability*

A language ought to have great unwavering quality. Specifically support for pointers, association and associating and so on ought to be debilitate. Moreover, support for affirmations, mistake checking and special case taking care of ought to be given by the language with the end goal that unusual states of the program are appropriately dealt with by the developers. Suitable Data Structures: A language ought to have support for assortment of crude information types and furthermore have the arrangement to develop client characterized information types on a case by case basis.

*4) Accessibility/Cost to understudies*

The expense of advancement stage ought to be low. In a perfect world, improvement apparatuses, compilers, mediators and Integrated Development Environment (IDEs) ought to be unreservedly accessible as open source instruments. Market interest: It is prescribed that the language being instructed to

computer science under study ought to have appeal in market. There ought to be suitable positions accessible for designers.

*5) Local area Support*

The language ought to have documentation, instructional exercises, and local area's help and engineer gatherings broadly accessible.

*6) Operating system/Machine Limitations*

The language ought to have least stage prerequisites. Specifically, it ought to be effectively ready to run on normal working framework. Expansions/Libraries accessible: The language ought to have augmentations accessible in wealth for explicit errands, for example, drivers for equipment interfacing, information base availability.

*7) APIs, parsers, GUI libraries and so forth*

Inclusion: The language ought to have adequate inclusion to be valuable to show computer science ideas. These points include: object arranged programming, multithreading, I/O, versatile registering, data sets, framework level programming and so forth.

*8) Clarity and viability*

A few languages are simpler for an individual to peruse, which makes it simpler for one software engineer to team up on another developer's code. Python, for instance, has gained notoriety for being not difficult to peruse. It authorizes severe space of lines to characterize its code blocks, which makes it simple to look at a program and sort out its design. Different languages permit space also, however as a complex decision, not as a necessity. Conversely, Perl is a language that permits the developer to compose similar program in various ways. In any case, the program's motivation probably won't be promptly obvious to another peruser. Such a program might be helpful to compose, however hard for another person to comprehend and alter.

*9) Execution*

A few languages are interpreted, and some are compiled. A compiled program should be handled by a preprocessor, compiler, and linker before it very well may be executed by the computer. This particular middle of the road programming performs lexical examination, making an interpretation of the program into machine language. It might likewise enhance the subsequent guidelines, searching for smart approaches to make the program run quicker. Compiled programs ordinarily perform better compared to interpreted projects. For example, C, C++, and Objective-C are languages that gather to extremely quick machine code. Computer games and framework programming are regularly written in these languages, to extract all of execution from the CPU. Then again, interpreted-language programs are controlled by programming called a mediator, which executes the program's directions without first ordering them to machine code. Albeit the mediator some of the time parses the program to a moderate language, bringing about some improvement, the presentation is never pretty much as quick as compiled machine code.

One significant advantage of interpreted languages is their potential for intuitive turn of events. Since the whole program shouldn't be compiled before it tends to be executed, the code can run intuitively. You know about this in the event that you've at any point utilized your working framework's order line: you enter an order and see the outcomes. Such an interface is known as a REPL, or "read-eval-print-circle." A REPL grants you to execute orders (or squares of orders) separately, and see the outcomes. Stutter, Perl, Python, NodeJS, Ruby, and JavaScript are instances of interpreted languages that can run in a REPL. Intuitive order interfaces, for example, the Windows Command Prompt and slam, additionally qualify as interpreted languages. Projects in these "languages" are called cluster documents, or shell scripts.

*10) Explicit use cases*

Frequently, languages are particularly acceptable at composing explicit sorts of projects. For example, NodeJS is intended to compose single-strung applications for the web. Its non-obstructing document I/O licenses projects to keep working ("not be hindered") while they trust that necessary information will send. Another model is the R programming language, which has practical experience in measurable investigation. Projects written in R advantage from worked in scientific tests and models, and apparatuses to productively control monstrous amounts of information.

*11) Simplicity of prototyping*

A few languages take into account quick prototyping: the software engineer can "begin composing," and assemble part upon part until the program is full-fledged.

For instance, the site Reddit was initially written in Lisp. After Reddit dispatched, the whole site was revamped in Python for some reasons, both specialized, and strategic. Despite the fact that refactoring all the code was a significant endeavor, the site's proprietors communicated no Second thoughts. In a 2005 blog entry, they commented that Lisp permitted them to make something without knowing precisely what it would turn into.

*12) Accessible libraries*

Ordinarily, when you start another programming project, you would prefer not to re-imagine the wheel. That is, you would prefer not to compose capacities for normal errands like computing a square root, or tracking down the primary event of a person in a string. Thus, practically every programming language gives a bunch of standard libraries of normal capacities. Software engineers might favor a language on account of the libraries it gives. For example, the C standard libraries give exceptionally performing capacities to some low even out framework activities. Perl gives numerous powerful libraries, and furthermore the CPAN store of modules to be downloaded and utilized in your program. Python gives a wide exhibit of implicit capacities and modules for nearly absolutely everything. Clojure, a Lisp-like language that sudden spikes in demand for the JVM, benefits from its capacity to run code from the broad existing libraries of Java items and techniques.

*13) Security*

Not all languages loan themselves to composing safe code. The C programming language, for instance, is infamous for having provisions (or scarcity in that department) that lead to obliterating weaknesses, for example, invalid pointer dereferencing. Different languages attempt to address these worries with stricter principles. For example, a few languages place limitations on what activities can be performed on

different sorts of information. The strictest of these languages are in some cases called "specifically," and they can offer significant serenity to software engineers who focus on security and solidness in programming improvement. Instances of specifically languages incorporate Rust, Nim, Ocaml, and Haskell. Languages may likewise put restricts on "variability," the capacity of an information object to change state. Rather than objects whose qualities are overwritten, these languages favor "permanent" objects: values in memory that can't be changed without unequivocal special case. Unchanging articles have drawn in revenue as multi-center CPUs have become broad, due to their propensity to advance "string security." In a string safe program, more than one processor might work on one bunch of information with an extraordinarily decreased danger of blunder. Languages that focus on changeless items incorporate Rust and Conjure.

*14) Local area support*

When programming in another language, it assists with getting to a functioning, enthusiastic local area of designers who effectively utilize and add to one another's work. Prior to picking a programming language, discover more with regards to that language's local area. A few languages have an invigorating, dynamic, energetic client base you should be important for and different languages might have almost no local area.

*15) Expressiveness*

When composing a program, the software engineer's musings and critical thinking capacities are “communicating in" through that language. Thus, software engineers will in general favor languages where they are happy with articulating their thoughts. What makes a language and software engineer function admirably together is difficult to characterize, notwithstanding. Eventually, the best way to know which language you are alright with is to utilize various languages for various activities and look at them for yourself.

## 4. Results and Discussion

In view of the above conversation, it tends to be presumed that Java is the best broadly useful programming languages to be utilized for teaching computer science ideas. It has great writability, unwavering quality, market interest and can be utilized to show any computer science idea like working framework, portable processing and so forth Other than Java, C# can likewise be utilized to show computer programming. Languages, for example, Pascal and GW Basic had been utilized broadly to show introductory level course, however are presently not sought after in market, nor would they be able to be utilized to execute current ideas of computer science like agents, plan examples and article situated programming and so forth

## 5. Conclusion

In this paper, an investigation of significant programming languages of computer science is finished. The paper looks at the chose languages dependent on various factors like their clarity, writability, support, market interest and inclusion. It has been inferred that Java is the most fitting language to be utilized for teaching computer science ideas.

## References

[1] https://www.juniorcoders.ca/blog/why-are-there-so-many-programming-languages/
[2] https://initialcommit.com/blog/why-are-there-so-many-programming-languages
[3] https://www.computerhope.com/issues/ch000569.htm
[4] Robins, A., J. Rountree, and N. Rooftree Learning and Teaching Programming: A Review and Discussion. Computer Science Education, vol. 13, no. 2, pp. 137-172, 2003.
[5] Milne, I. and G. ROWE Difficulties in Learning and Teaching Programming— Views of Students and Tutors. Education and Information Technologies, vol. 7, no. 1, p. 55-60, 2002.
[6] Davies, S., J.A. PolackWahl, and K. Anewalt A Snapshot of Current Practices in Teaching the Introductory Programming Sequence.beligin, pp.23, 2011.
[7] Schulte, C. and J. Bennedsen. (2009), What do teachers teach in introductory programming?, IEEE conference proceeing,pp.58-65.