Programming Language Trends

Heritage college | Ronald Patterson

Systems IV

Philip Dumaresq

2018

# Executive Summary

This report is going to cover the topic of programming language trends, primarily addressing the question of why so many different programming languages exist. To answer this question, it will first take a look at a brief history of programming languages; beginning with punch cards and mathematical languages developed in the 19th and early 20th century, to the rise of high-level languages such as FORTRAN, to today with newer languages like C# and JavaScript. Following that, the report will look at some industry statistics to find out what some of the most prominent programming languages are today and try to delve into a comparison of them to determine what makes them the industry influence that they are. Some key factors that will be looked at are the speed of the language, looking at benchmarks for how quickly each one can compute certain algorithms; the paradigms these languages fall under, primarily looking at the importance of Object-Oriented Programming and Functional Programming in the industry; the influence that some of these languages have had on certain domain-applications in the industry; and where each of the languages has drawn their inspiration from and what they’ve all inspired in new programming languages that we see coming out today. Following that, Python and Ruby are going to be highlighted a bit for their huge difference in speed from the other languages here. Ruby and Python crawl in comparison to Java or C, and yet are used for some very heavy-duty work, such as machine learning and server-side scripting. This is because they’re such simplistic languages in terms of readability and syntax that they offer a greater speed of development, which mean increased productivity for the programmer – which companies seem to value more than the speed of the execution of a program. Once that’s been unraveled, the biases of certain programmers will be explored, questioning why some developers criticize some languages so heavily like JavaScript and PHP, regardless of how widespread and used they are. A lot of these biases end up coming from out-of-date knowledge of the languages and aren’t the concern today that they were 5+ years ago. With all these languages, there’s so many different opinions on how one should get into Computer Science and what languages you should learn first. That’s going to be explored next, ultimately deciding that it doesn’t really matter what language someone learns first. It’s more important to know how to think like a programmer than it is to know a specific language. Learning new languages is easy. Finally, there will be a look into the idea of having one universal high-level language. A programming language that can handle all the domains that today’s languages attempt to cover, one that can be used across all paradigms, work on all platforms. This is a language that doesn’t really exist, and probably couldn’t without becoming incredibly over-encumbered with features. To conclude, a brief look will be taken at the Computer Science program at Heritage, and how it should adapt to changes in the industry. With the growing popularity of functional programming, perhaps it should be included in the program; however, being a career program, it also needs to meet the requirements of employers in the area, and so the program should continue to adapt as needs change. Predicting the rise and fall of languages is nearly impossible, and so being reactive and keeping a close eye on employers is exactly what the college should continue doing.

Table of Contents

[Executive Summary 1](#_Toc508795717)

[Table of Figures 2](#_Toc508795718)

[1. Introduction 3](#_Toc508795719)

[2. Top Languages Today 3](#_Toc508795720)

[3. Drawing Parallels 5](#_Toc508795721)

[3.1 Speed1 5](#_Toc508795722)

[3.2 Paradigms 6](#_Toc508795723)

[3.3 Industry Influence 6](#_Toc508795724)

[3.4 Source of Inspiration 6](#_Toc508795725)

[4. Outliers 7](#_Toc508795726)

[5. Biases 7](#_Toc508795727)

[6. Starting in Computer Science and coding 8](#_Toc508795728)

[7. One universal language 9](#_Toc508795729)

[8. Conclusion 10](#_Toc508795730)

[9. Appendix 11](#_Toc508795731)

[10. Sources 13](#_Toc508795732)

# Table of Figures

[Figure 1: Tiobe Index of top 10 programming languages 3](#_Toc508550663)

[Figure 2: GitHub stats for most popular programming languages. 3](#_Toc508550664)

[Figure 3: GitHub stats for 8 similar languages with the Tiobe index. 4](#_Toc508550665)

[Figure 4: Execution time of several languages running through nested loops with basic decision making 5](#_Toc508550666)

# Introduction

Programming languages have been around in some way shape or form for a very long time. Punch cards were first used to program looms in the early 1800s to create intricate patterns in weaves. Later, Charles Babbage created a proposal for a computer called the “Analytical Engine” which was broken down by Ada Lovelace to create what’s considered to be the first programming language. Her language was purely mathematical and was used to calculate Bernoulli numbers on the Analytical Engine. The concepts of these theoretical machines and languages were eventually adapted by Alan Turing and Alonzo Church to create the “Turing Machine” and “Lambda Calculus”. These, along with other theoretical research, paved the way for some of the earliest true programming languages.

In 1957 came FORTRAN, which was released as the first high-level programming language. After FORTRAN came many others, LISP, COBOL, ALGOL, and many more. In 1973, C was first released by Bell Labs and became the predecessor for many of the modern programming languages we see today, including C++, Java, C#, and many more. Eventually came the birth of the internet, which spawned new languages like PHP and JavaScript. New languages are constantly appearing and disappearing nowadays, and we always see up and coming languages in different fields, but why? Why do we have or need so many programming languages? This is something that’s going to be investigated by looking at some of the most popular programming languages today and breaking them down. See what they’re used for, how they preform and what they bring to the table. With so many languages lying around, another question that’s often asked is what language should you learn first? And is it possible to have a single language that can do anything?

# Top Languages Today

There’s many different places you can go to find “Top 10 programming languages of 2017” or articles akin to that, but looking at reliable data sources of data for this, what do we find? Looking at GitHub repository statistics and the Tiobe Index, which is a reliable source that pulls data from search trends across major search engines, Wikipedia, Amazon and YouTube, we can find some of the most popular languages and see how they trend.

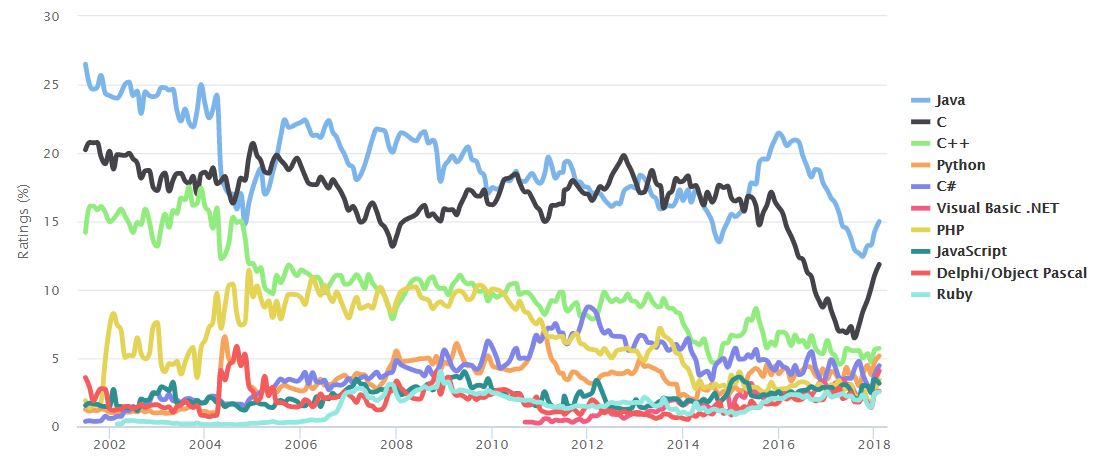


Figure : Tiobe Index of top 10 programming languages

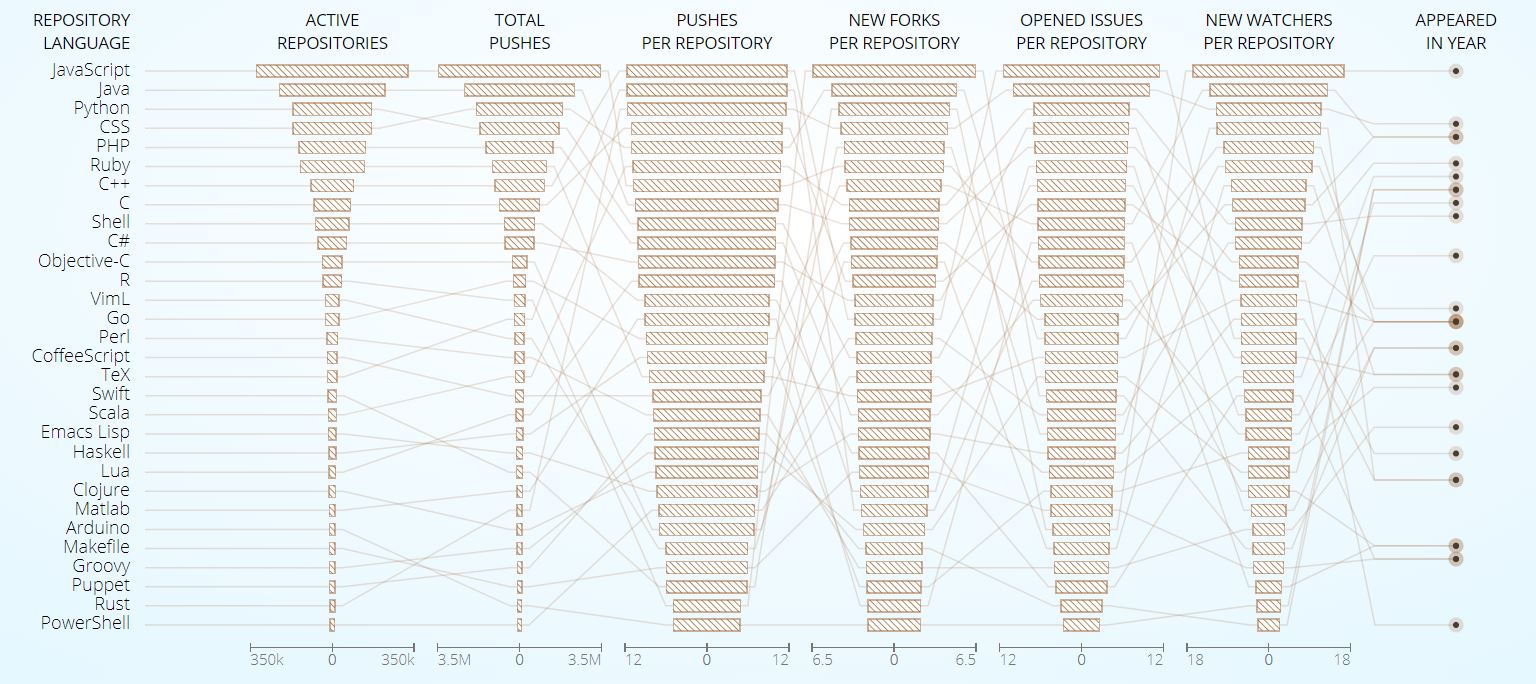


Figure : GitHub stats for most popular programming languages.

We can see in these two images that some of the common top languages are Java, C, C++, Python, C#, PHP, JavaScript, and Ruby. All 8 of these languages appear in the top 10 of these two lists, and I’m going to try and look at some of the similarities between the languages.

Note: I’ve excluded CSS from the top 10 in GitHub since it’s not a true programming language. It is technically Turing complete, however is never used as such.

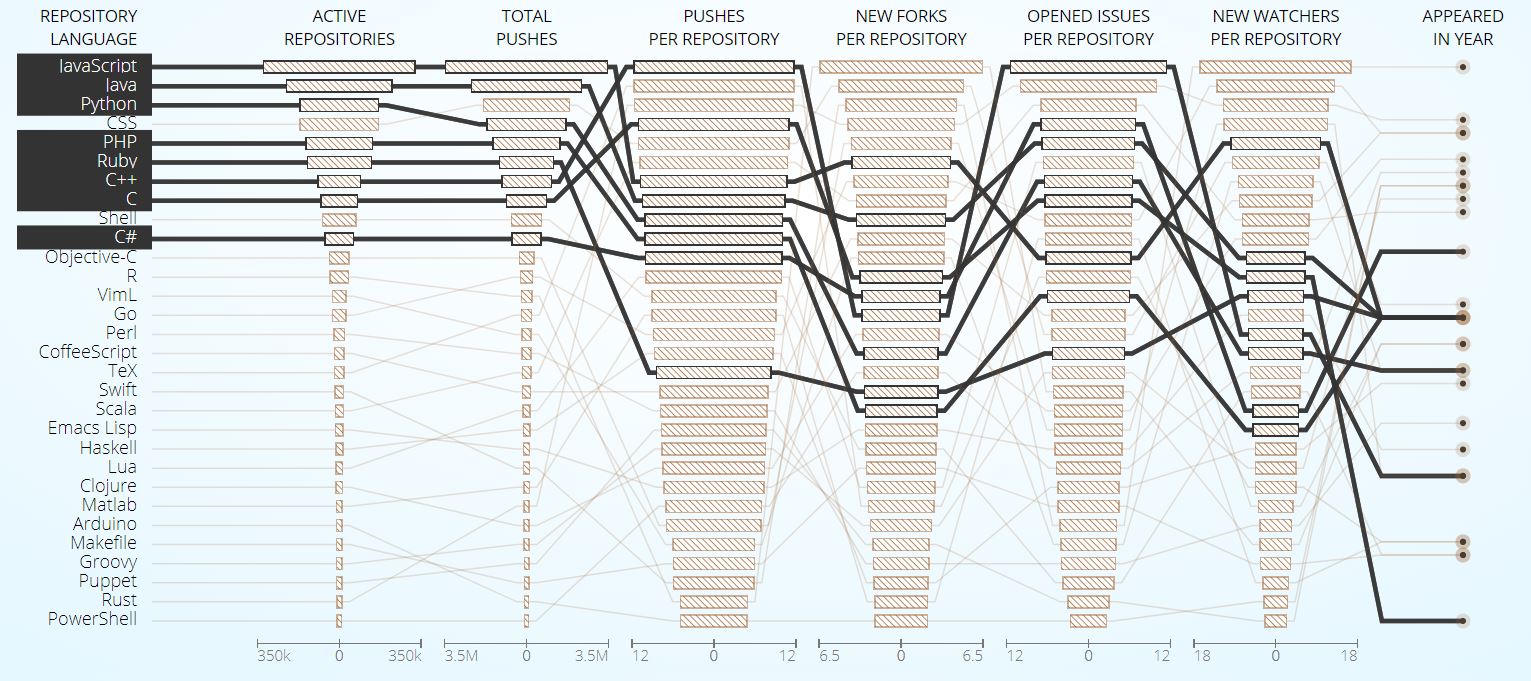


Figure : GitHub stats for 8 similar languages with the Tiobe index.

# 3. Drawing Parallels

Most of these languages were made between 1991 and 2000, except for C and C++. This is interesting, because not a single language in these lists is a newer one like Swift, Rust or Go, but rather all older ones that have been around for 20+ years. We also notice that six of these languages have the familiar C-style syntax. But what else do these languages have in common?

## 3.1 Speed1

Looking at one of the first factors you’d think of when choosing a language, speed is crucial when developing large scale applications. C and C++ are some of the lowest level languages commonly used today and are hailed by programmers for being so quick. Java and JavaScript are both languages that are often criticized for their speed, Java running through the JVM and the common thought that JavaScript is interpreted. However, the thought that these languages are slow simply isn’t true, they’re both JIT compiled (Chrome’s V8 and Firefox’s JaegerMonkey JS engines) making them almost as fast as their predecessors. The graph bellow shows them being even faster for this one benchmark test, although overall this probably isn’t true. We also see the latest version of PHP running among the top tier of languages in this list. However, this is where we start to see Python and Ruby differ from the rest. Python and Ruby, in their official supported implementations of CPython and CRuby are both interpreted languages, which slow them down quite a bit. Official versions of the languages are where the focus will be, however, PyPy is a bootstrapped version of Python written as a JIT compiler, and you can see the massive increase in speed that it creates.

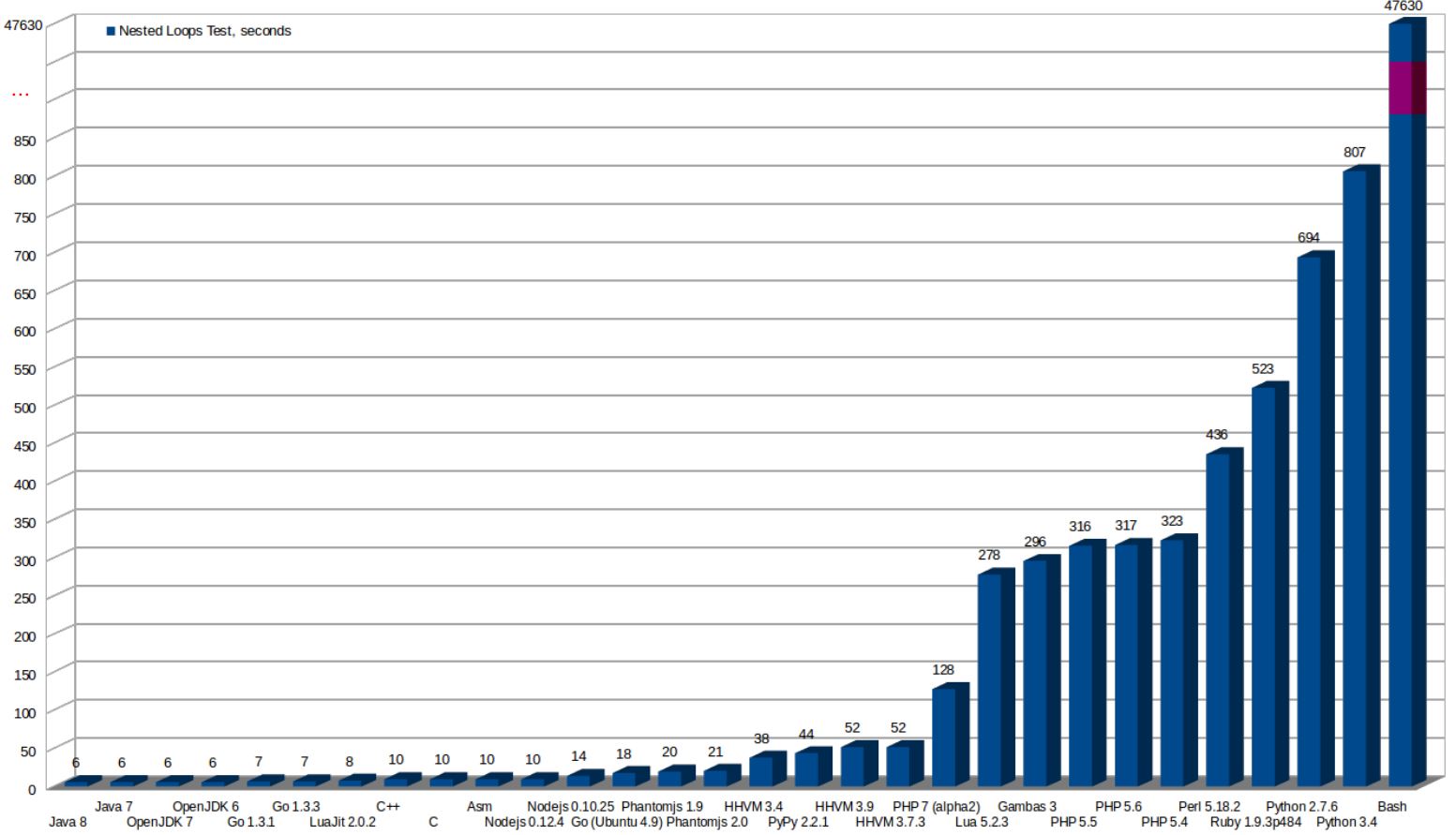


Figure : Execution time of several languages running through nested loops with basic decision making2

This huge jump in speed from PHP to Ruby to Python begs some questions that will be addressed later about their use in the industry.

## 3.2 Paradigms

Looking at programming paradigms, we can see that all of these languages, with the exception of C, support Object-Oriented Programming. Object-Oriented Programming is one of the biggest paradigms today, allowing you to build very large, scalable, modular applications. Object-Oriented also allows the use of modules to be more easily reused across multiple applications within the same domain to a much greater degree than most other paradigms. This large proportion of object-oriented languages is telling of how welcoming the industry is to it, and how supporting object-oriented is so important to developers today. Looking beyond these 8 languages we’re looking at, so many other growing languages are OOP including Swift, Rust, Scala, Erlang, Hack, GoLang, VB, and many more. Of our top languages, many of them also support concepts born in functional programming, including JavaScript, Python, PHP and Ruby. Newer versions of Java and C# also supporting basic functional concepts like closures (lambda functions), and being entirely non-functional languages, this highlights how functional programming is growing if they’re attempting to support some of their features. Beyond Object-Oriented and functional programming, these languages cover a variety of other paradigms as well, including procedural, event-driven and imperative programming, however Object-Oriented and functional programming appear to be two of the key paradigms we see.

## 3.3 Industry Influence

Java is still huge in enterprise scale applications, with an estimated 75+% of fortune 500 companies using it for many different purposes. It’s also the primary language being used for Android development, and the language that a lot of colleges and universities are still opting to teach first. JavaScript is the language that the web runs on. Interactive and dynamic web applications wouldn’t exist without JavaScript, and it’s being used more and more on the server side in Node.js and Electron development. Python is growing in machine learning, big data and artificial intelligence, etc. The industry influence that each of these languages has had keeps going, and they have the kind of support that they need to stick around for many more years to come.

## 3.4 Source of Inspiration

We see languages everywhere that draw inspiration from these 8 languages. C++ was made as an improvement to C, and Java made to fix a lot of the issues in C++. PHP was originally made as a web library for C that turned into its own language. A lot of these languages draw inspiration from each other, and dozens of new languages that spawn every year from each of these. Going back to the mention of Swift, Rust and Go, if we look at the syntax and features of these languages, they all share a lot of similarities with the top 8. They’re all syntactically similar, touch on a lot of the big paradigms and all have large companies backing them giving them the library and documentation support needed to grow.

# Outliers

A lot of these languages follow many of the same concepts. As mentioned, they mostly follow the same familiar C-style syntax, are very fast, compiled, procedural languages, have lots of community support, and have been around for quite some time. Python is interesting in this regard because it doesn’t fit into a lot of this mold. While the community support is there, and it’s been around longer than most of the languages on this list, it deviates from the rest in terms of syntax, runtime environment, speed and paradigms. Python, first of all, is slow; very slow if we compare it to the other languages up there. Python’s biggest applications right now are in machine learning, big data and artificial intelligence. These are 3 fields where you need to be parsing through massive datasets, running complex algorithms and doing a lot of heavy lifting. Python being over 100x slower than Java (for this benchmark specifically) makes it seem like an odd choice for doing such heavy lifting. While this is something that seems like an odd choice for a company to make at first, after looking into it, it makes a lot of sense. Python was a language that was built to be easy to develop, easy to read. The syntax is very simplistic, often looking much like pseudocode. It also requires none of the boilerplate code that a lot of other languages use to get up an running; you can just make a new file and start coding right away. This simplicity allows software developers to be much more productive when writing code in Python. The increased productivity is what companies are looking for in Python. It costs more to pay good developers than it does to buy more servers or more processors to speed up the code. This trade-off of speed for productivity is important for companies, and one of the big reasons that Python is used – among the number of libraries that are available for just about everything.

Ruby falls into a lot of the same categorizations as Python does. It follows a very similar syntax to Python, is interpreted much slower than the other languages, but stands out because it also doesn’t really exist outside the context of Ruby on Rails. Rails is a web development framework for Ruby and is the only real usage for Ruby in the industry. Rails is a very straightforward and easy framework to use though, making it a top choice for a lot of younger companies.

# Biases

Another interesting trend in languages is the biases that some people hold against certain languages. Two of the first languages that come to mind that people object against are JavaScript and PHP. So many developers criticize PHP for historically having a lack of consistency and some bizarre syntax choices, ignoring the improvements made to it over the course of the past several years since PHP 7 came out. PHP is a simplistic language with a familiar syntax, without any of the boilerplate code. This makes PHP a language that’s quick and easy to develop in. PHP 7 also turned PHP into a very quick language, running several times faster than any of it’s predecessors. PHP also has built-in functions for handling just about anything you’d want to do in a web-server language. You can query databases, send emails, and manipulate arrays or strings in so many ways natively. A large percentage of the web today runs on PHP, and it’s a language that anyone doing web development should know. Yet people still avoid it because of this impression that’s been left with it.

JavaScript has a similar issue, where people can’t seem to let go of the fact that it’s not the security concern that it used to be. JavaScript will be avoided at all costs for some developers, even though over 90% of all websites use it. JavaScript, like PHP, follows the familiar C-style syntax, is an incredibly fast language, and is just about the only option for any client-side code in web applications. Somehow though, JavaScript is still heavily critiqued by many programmers. These biases, especially those based on out of date knowledge of certain languages, can steer people away from perfectly good programming languages, which can really hurt when it’s the best tool for the job.

# Starting in Computer Science and coding

With so many new languages popping up all the time, biases against so many languages, pros and cons to each, specific domains covered, etc., how does one get into Computer Science or coding? When you look up anything along the lines of “which language to learn first”, there’s thousands of hits with articles written by programmers on their blogs and its entirely opinion based. Some developers say you should learn Python because it’s easy, JavaScript because it’s everywhere, Java because there’s so many resources, etc. An argument can be made for just about any language, so it can be confusing for someone new to slog through all these articles about what to learn first. There’s so little written that’s based on research, statistics or facts. There’s also the issue of new languages not gaining traction and disappearing after a couple year.

If you’re doing some research into figuring out which language you should learn first, it’s very easy to stumble across languages like Rust or Go, which are quite new in the industry. Languages come and go, and some languages just don’t stick around the way that they were expected to. But is the first language you learn really that important?

Learning how to code is just learning how to think in new and different ways. The language that you learn to code in doesn’t really matter all that much in the end because no matter what it is, you’ll learn the same basic concepts. You’ll learn conditional statements, looping, functions, etc., and these are the things that are required for programming. While it can be difficult to start learning all this new syntax for the language you learn, that’s not what programming is about. Once you learn the concepts of programming, be it in just about any language, you can take those skills an apply them in a new language. Sure, you might need to spend the first couple days of programming looking up all the syntax, but nothing will come to you as much of a shock. Where things can start to become more surprising is when you begin shifting programming paradigms, because again it will change your way of thinking. To be a good programmer isn’t to know all the syntax of a language or make a language bend to your will, but to know how to think in ways that will help you solve a problem in the best way possible.

# One universal language

Java is a language that’s used for just about everything today. It’s used to make Android apps, web servers, desktop applications, and plenty more. However, Java is also slowly disappearing. It’s a language that’s become so cumbersome to use that it’s beginning to kill it. This begs the question of how manageable such a massive language really is. If you had a language that could handle everything from low level work like interacting with your hardware and doing work with input/output to control physical components, to building web applications or querying a database, working in event-driven, object-oriented or functional paradigms, etc., this language would be impossibly huge.

Because Python has had several decades to embed itself in the industry, it’s developed a huge community backing. This means that Python has libraries for it that can help you do so many different things. It’s become kind of a one-stop-shop, where if you know what you need to use, Python can do it decently well. It’s got libraries and frameworks for web development, scientific use in engineering, statistics and mathematics, artificial intelligence and machine learning, etc. This makes Python an easy language to default to in order to get something up and running quickly.

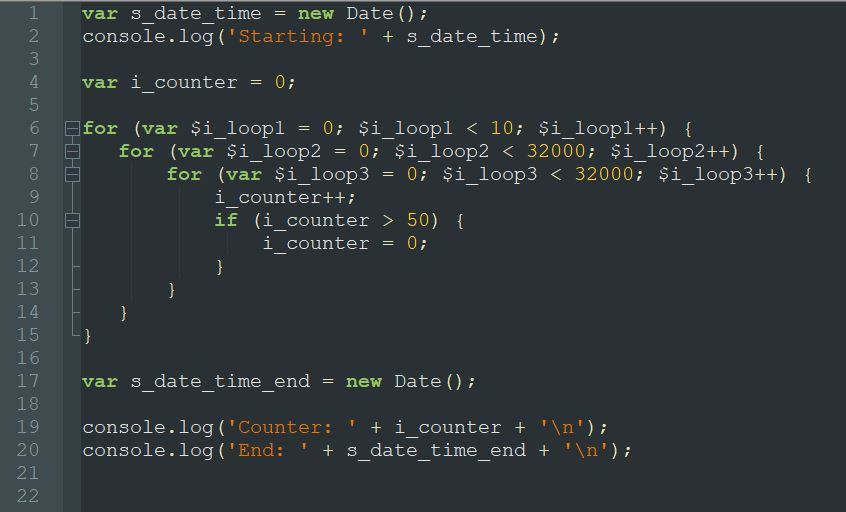
Old languages like these are as close as we come to something universal, but we certainly wouldn’t use them for everything, which is what something universal would be. You would never build an OS using Java or Python because they’re both too far from the metal, you would use C and ASM for that. But you’d never use C or ASM for your web-server’s back end because they’re not scalable and too complex. There’s no language that’s a good tool for any job, and there likely never will be one. While it’s confusing to some people who are getting into programming, having so many languages laying around is necessary because each language does its own thing, and you can’t have one language doing all the work.

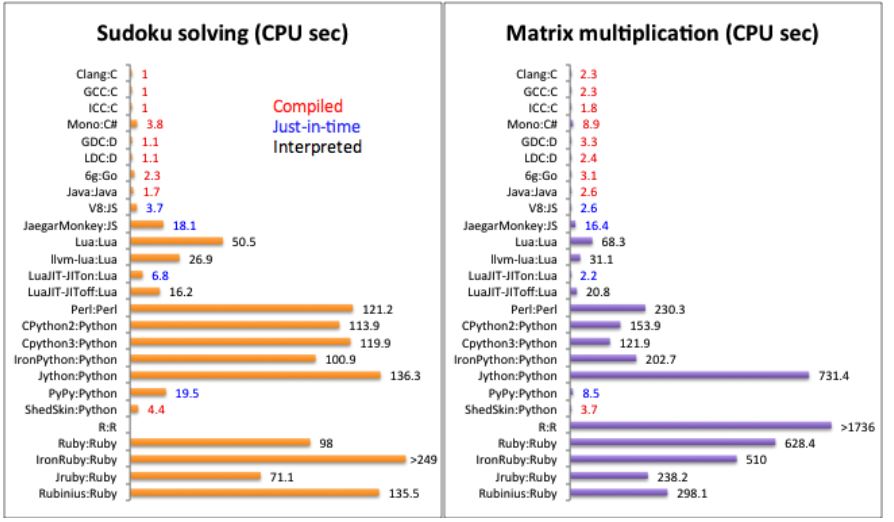
# Conclusion

In conclusion, we really do need all the programming languages that are out there. Each one has its own strengths and weaknesses and makes improvements upon those its based off. This has lead to a very diverse field of languages to chose from, giving programmer’s the freedom to hold biases against ones they do or don’t like, and chose one that’s the best for the job, and not just because it’s all there is. Having so many languages out there leads to innovation and evolution of the field of Computer Science. But it also means we barely know what the industry will look like in 2 or 3 years, never mind 10 or 15. So how do Computer Science programs know what to teach to new students?

Ultimately, it probably doesn’t matter all that much. Teaching Java has given Heritage students strong foundations in programming, but very few students actually get to use it during their coops. By the time students reach the end of their time at Heritage, they know what they’re doing. Graduates shouldn’t be struggling with any of the foundational knowledge that Java teaches, and so the only different moving from one language to another is syntax. Syntax is quick and easy to learn, and most programmers are going to be able to pick up new programming languages relatively quickly – so long as they don’t drastically differ somehow. The biggest shift from one language to another though is going to be shifting programming paradigms, which is going to change the way you think about things anyways. So, it doesn’t really matter what language Computer Science programs teach first. Heritage, being a technical program however, aims to get students jobs as soon as they can by putting important languages on their resumes; and for this reason, nothing really needs to get changed. In the end, it would likely benefit students to learn some languages that make them think in new and different ways, like Haskell or some derivative of Lisp. However, that doesn’t get them where they need to be in quite the same way learning Java and C# does, and so it shouldn’t really be changed. Heritage should continue to do what they have been doing; keeping a close eye on the changing needs of the industry and reacting to those changes as the faculty sees fit, instead of trying to predict them or teach by ignoring them all together.

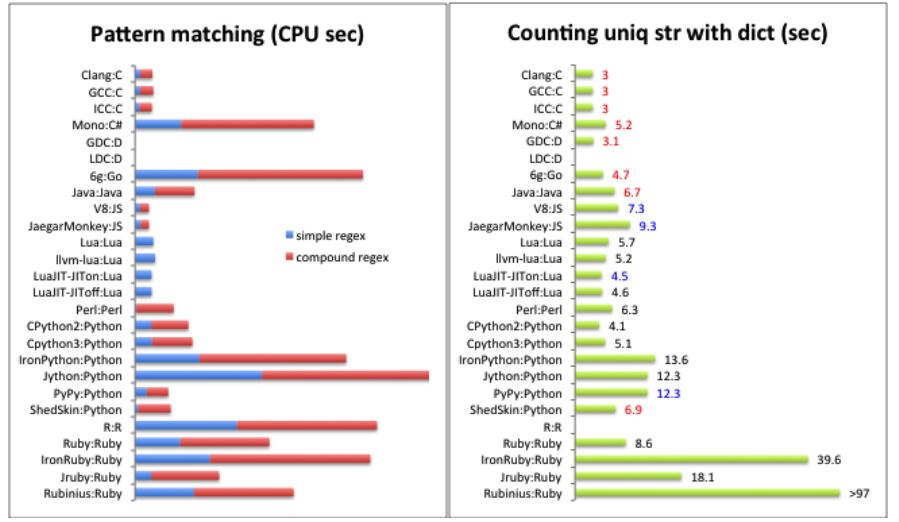
# Appendix

1. Each language has its strengths and weaknesses. A single algorithm used as benchmark isn’t enough to fully evaluate the speed of each of these languages; for that you would need to do much more.3 This is simply drops each language into a rough field of where they lie. Is Java really faster than C or assembly? Probably not, but in this one benchmark it happened to come out executing the code faster. Take anything to do with execution time of a language as a ball park estimate, and not the end all be all of benchmarking speed.
2. The algorithm used to generate the benchmarks – JavaScript implementation 
3. More benchmarks for comparing languages.



Sudoku: Time taken to solve 20 extremely difficult sudoku puzzles repeated 50 times.

Matrix: Time taken to multiply two 1000x1000 matrices using the standard O(n3) algorithm.

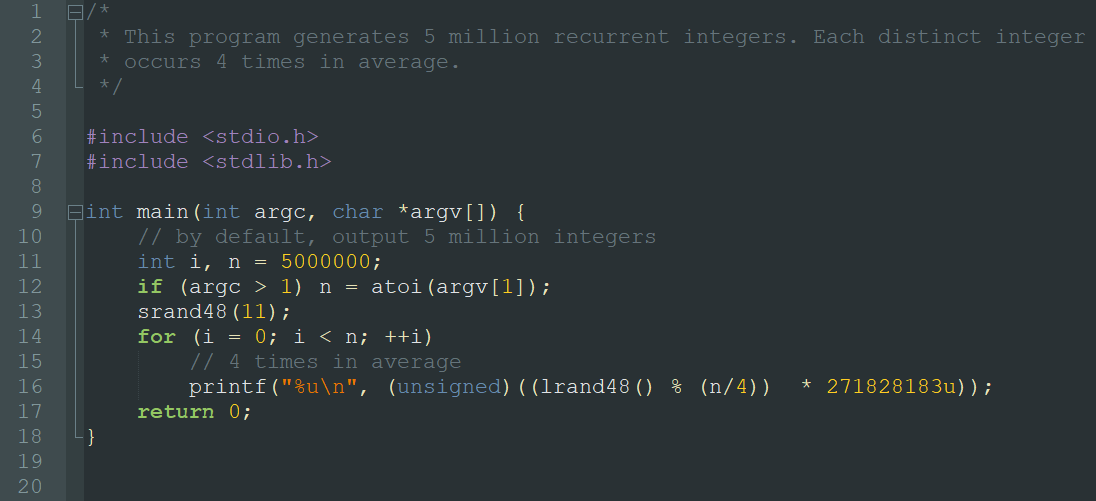
**

Patterns – simple: time taken to find all matches of “([a-zA-Z][a-zA-Z0-9]\*)://([^ /]+)(/?[^ ]\*)” in a 37mb file.

Patterns – compound: Same file, matching ([a-zA-Z][a-zA-Z0-9]\*)://([^ /]+)(/?[^ ]\*)|([^ @]+)@([^ @]+) instead.

Counting uniq str with dict: Time taken to count the occurrence of each unique string among 5 million strings.4

1. The 5 million strings are all integers generated by the following C code:



# Sources

1. Alpheus, “*Universal Programming Language*”, October 16th 2013, <http://wiki.c2.com/?UniversalProgrammingLanguage>
2. AttractiveChaos, “*Programming Language Benchmarks*”, June 21st 2011, <http://attractivechaos.github.io/plb/>, Accessed March 12th 2018
3. Carles Mateo, “*Performance of several languages*”, March 26th 2017, <http://blog.carlesmateo.com/2014/10/13/performance-of-several-languages/>, Accessed March 12th 2018
4. Carlo Zapponi, “*A small place to discover languages in GitHub*”, December 2014 <http://githut.info/>, Accessed March 12th 2018
5. “*History of Programming Languages*”, August 21st 2009, <http://www.unhas.ac.id/rhiza/arsip/kuliah/Komputasi-dan-Pemrograman/history-prog-lang.pdf>, Accessed Mach 12th 2018
6. [Itamar Turner-Trauring](mailto:itamar@codewithoutrules.com), “*Your technical skills are obsolete: now what?*”, October 23rd 2017, [https://codewithoutrules.com/2017/10/23/obsolete-skills/](https://l.messenger.com/l.php?u=https%3A%2F%2Fcodewithoutrules.com%2F2017%2F10%2F23%2Fobsolete-skills%2F&h=ATM9ICXcx6MEpo_-GwYH-84ev03CZ5uYd8JSFklWGMNrx3x61UT097a0vbTcH5j_JZyGWVYFu_a3eonMzzYLnBMybplUUtrdBfxvhMj5mHbW0NgAl-6ZRsrQB8c5N3dA5UJ1TZCR2Ow148s), Accessed March 12th 2018
7. Jason Brownlee, “*Best Programming language for Machine Learning*”, May 10th 2014, <https://machinelearningmastery.com/best-programming-language-for-machine-learning/>, Accessed March 12th 2018
8. Mike Vanier, “*Scalable Computer Programming Languages*”, April 11th 2017, <https://machinelearningmastery.com/best-programming-language-for-machine-learning/>, Accessed March 12th 2018
9. Tiobe, Tiobe Index, March 1st 2018, <https://www.tiobe.com/tiobe-index/>, Accessed March 12th 2018
10. Yaofei Chen; Rose Dios; Ali Mili; Lan Wu; Kefei Wang, IEEE, “*An Empirical Study of Programming Language Trends*”, June 6th 2005, <http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/trend/s307225.pdf>, Accessed March 12th 2018
11. Yi-Jirr Chen, “*What Programming Language should a beginner learn in 2018?*”, January 23 2017-March 7th 2018, <https://www.codementor.io/codementorteam/beginner-programming-language-job-salary-community-7s26wmbm6>, Accessed March 12th 2018
12. <https://w3techs.com/>