Kali Burres

Professor Sabal

CIS 333

Unit 2

Exploring Language Growth

It can be hard to tell what causes language popularity, as it seems each language has a different popularity evolution. Some languages that held popularity during the rise of programming have slipped significantly in significance in recent years. Ada and Lisp, for example, were rated numbers 3 and 2 respectively in 1986, but received ratings of 33 and 29 for 2021(tiobe). C, on the other hand, has topped the charts consistently since 1986, (which is now 35 years ago) consistently placing 1st or second in the list of top 10 programming languages per year (tiobe). Java, too, made a rapid rise from 28 to 3 between 1996 and 2001, and has continued to average in the top 3 since (tiobe). The difference in results for these four languages shows that despite new programming trends and large companies' desire for consistency, age alone cannot come even close to explaining language popularity, either for or against it.

So, what is it then that causes language popularity? To get better insight I attempted to figure out which languages were the most popular for the longest. C, unique for its various forms, has succeeded in all of it's forms in recent years, with C# hovering between 4 and 8 since it's conception around 2000 (Homem) and C++ keeping above 5 since the early 1990s (tiobe). This makes it very much worth studying. JavaScript, extremely popular the past few years and

continuing to rise in popularity as it evolves for backend use, had a somewhat rocky rise to popularity, and tiobe has no records even recorded prior to its renaming from LiveScript in 1995 (Peyrott.). Python, only barely noticeable among the big names in programming for most of the last decade, made a recent spike in the last three years, landing at number 3 for January 2021 (tiobe). This not only makes it worth considering, but also brings questions: What is it about python that made it so popular after practical obsolescence since its inception in 1991? (Python Institute). Is this popularity built to last, or will it be a passing trend amid all-time favorites like C and Java (Java being only about as old as Python itself)?

After pinpointing those languages which have been the most significant in terms of popularity (and perhaps change in thereof) over the last thirty years I decided to compare them using rosettacode.org. While my findings were not conclusive, I was able to make some observations. First, for all the regulations and specifics Java requires in the way of syntax, it is surprisingly readable. Commands are easy to memorize and recall, or even to understand on a basic level without prior introduction if come across when reading code. ADA, on the other hand, seems to use a strange combination of long titles and short commands. Some commands have long, fully spelled out names, and the more foundational aspects of the language seem so short they are hard to understand. Declaration for different things, such as loops, also seems awkward, especially for something used so often in programming.

C almost seems the opposite of Ada in terms of its readability and structure. Where keywords were natural, easy to understand, and even long in Ada, C's keywords are often short and abbreviated, making them harder to understand but easier to type. The structural syntax, on the other hand, is the opposite. Where in Ada the syntax seemed cryptic and even

somewhat messy, in C the syntax is clear, distinctive, and even somewhat natural to read. Ada, however, was surprisingly able to accomplish a sample program in many less lines of code than C, suggesting either that length of code is not as important to programmers as one might think or that C's overall strengths far outweigh Ada's.

Python, though verbose, was able to accomplish a sample program in surprisingly few lines. Even in more complicated scenarios possibly not quite best suited for Python, its results were not significantly more than average in length. Python is also notable for its reliance on indentation for structure, a feature contrasting starkly with many languages.

After analyzing both the popularity and structure of some of the world's biggest languages, I lastly turned to outside sources for some insight on why someone might choose one language over another. One factor that I noticed cannot be ignored in the rise and fall of programming languages is power support (codeacademy). If Apple designs their operating systems in such a way that only a few languages work well with it, then whether those languages are all that great or not, they will rise in popularity. A company may also create or adopt a language and cause it to rise in popularity that way. C#, for example, was written by Microsoft (codeacademy). If Microsoft uses C# for all their applications and designed it specifically to work well with Windows, then one could argue that C# is not just popular as a result of its power, flexibility, and readability. It may be so popular largely due to Microsoft pushing it upon programmers with their large presence in the technology world.

Second, I realized that while readability seems like one of the most important factors in the popularity of a language, this may not be true for languages designed specifically for the

backend, where the focus of the language is how it communicates with machines, not people (codeacademy). Having not worked much with backend languages, I found this interesting.

Finally, I found a description that seemed to sum up what I had found. In his book, Topics in Programming Languages, Luis Homem makes an interesting statement. He says that the history of programing involves languages "becoming more and more *natural*" (Homem, emphasis added). That word seems the most appropriate because it is easy to think readability and flexibility create the most natural programming language, but that can often make security, structure, or even the physical typing of the code very *unnatural*.

It seems that for a language to be natural it must find a well-rounded and delicate balance between flexibility and structure, between readability and typeability, and most importantly, that it have an identity. Modern brands do not survive today without understanding that you cannot be everything to everyone. It seems that Java has remained popular because when people need security, they are willing to put up with the annoyance of it's structure for the benefits of it, so Java need only worry about small ways to make it's structure more convenient, and does not ever have to feel the need to be Python or JavaScript. In other words, there is no one-size-fits all in programming. Programs that are built with solid purpose and identity, remain "natural" in many (although all is likely impossible) senses of the word, and are continually maintained to best move toward these ideals seem the most likely candidates for widespread popularity, and languages that fail in any of these areas can likely expect less use.

Works Referenced

Homem, Luís Manuel Cabrita Pais. Topics in Programming Languages : A Philosophical Analysis Through the Case of Prolog. Chartridge Books Oxford, 2013.

"Latest News." *TIOBE*, www.tiobe.com/tiobe-index/.

Python Institute, pythoninstitute.org/what-is-python/#:~:text=Python is a widely-used,released on February 20, 1991.

Rosetta Code, www.rosettacode.org/wiki/Rosetta_Code.

Sebastian Peyrott. "A Brief History of JavaScript." *Auth0*, 16 Jan. 2017, auth0.com/blog/a-brief-history-of-javascript/#:~:text=In December 1995, Netscape Communications,to develop rich web components.

Team, Codecademy. "What Programming Language Should You Learn First?" *Codecademy News*, Codecademy News, 26 Oct. 2020, news.codecademy.com/what-programming-language-should-i-learn/.