The Need for Speed: Julia Vs. Python

M.T White

Abstract

In a world that is dominated by speed and instant gratification, it comes as no surprise that the technologies that are necessary to power applications also have to be as fast as possible. Currently, the programming language Python is the ruler of the programming world. Python is being used to program everything from web applications to advanced AI systems. However, in recent years a challenger has emerged that is attempting to dethrone Python. Julia is a new programming language that is touted as a Python killer. Sporting an expressive syntax and a JIT compilation using LLVM it has the speed of C++ with the development feel of Python. Though it is often considered faster than Python there are several areas that most studies ignore such as code styling and general program architecture. As such, this study looks at the speeds of basic functionalities such as for loops, printing basic math equations to the console, and if statements of both Python and Julia to determine just how much faster Julia is. By only studying the listed commands all bias to architecture and coding style can be mitigated or eliminated and a true understanding of how much faster Julia is can be determined.

1.1 Introduction

Julia is a fairly new programming language that has a simplistic, expressive syntax similar to Python or MatLab and it is touted to have the same execution speed as a compiled language such as C++. Currently, Julia is being explored across many different fields such as machine learning and mathematical computations. According to the official website, Julia code compiles to native code for multiple platforms because of the LLVM compiler [2]. Even with the recent rise of Python as the most popular programming language by many surveys, Julia is often touted as a Python killer.

Julia was released to developers in 2012 with Julia 1.0 being released in 2018 [1]. Due to the age of Julia little work has been done in the way of benchmark testing. Some studies are available that demonstrate the speed advantages of Julia over Python. For example, *The computer language benchmarks game* website list many complex algorithm benchmark tests between Julia and Python [3]. However, what is not looked at are benchmarks for core functionality such as loops, control statements, and basic mathematical equations that usually make up the core of most complex algorithms and programs. With that being said, at the fundamental level, how much faster is Julia than Python? The following will explore the two languages along with experimental results derived from core functionality such as loops, controls statements, and mathematical expression in an attempt to answer that question.

1.2 Python

To understand the benchmarks between Julia and Python one must first understand the differences between Python and Julia. As it stands as of now, there are two versions of Python. There is Python2 and Python3. Python2 is an older version of Python that is mainly used to

support legacy codebases while new projects are created with the later version Python3. For the study presented in this article, all Python code was written using Python3.

In general, Python is a dynamically typed, interpreted language that offers a very expressive syntax that is very easy to read. Python is by no means a new language; its debut was in 1991. Python has a very rich ecosystem and is currently a popular platform for machine learning, data analytics, web development, as well as many other applications. Due to the very simplistic syntax of Python, it is often used by scientists and engineers that need to quickly write programs [4]. According to Oliphant the clean syntax, indentions, modules, and easy-to-read loops play a significant role in drawing in scientists and engineers [4].

The only drawback to Python is that it is interpreted which means that programs written in Python will traditionally be slower. Source code for languages that are interpreted is translated line-by-line. As such, compared to compiled languages such as C++, a script written in Python will execute at a significantly slower rate.

1.3 Julia

Julia offers the same easy-to-read and easy-to-understand syntax that is offered by Python. Also, like Python, Julia is a dynamically typed language. However, as stated before Julia uses LLVM to compile source code as opposed to interpreting source code like Python. More specifically Julia is a JIT compilation. Julia is also dynamically typed which means it has a similar feel to a scripting language [2]. In short, developers developing using Julia will have a similar experience to developing with Python. However, the mere fact that Julia is compiled means that it will automatically have an advantage over Python in terms of execution speed.

Though Julia is advantageous to Python in terms of execution speed it is not without its general drawbacks. The main drawback comes from the language's immaturity. Unlike Python that has been actively developed for the past 30 years, Julia is a new language. This immaturity means that there is a lack of solid experimental data on performance, a limited and immature ecosystem, and a general lack of knowledge when it comes to the language.

2.1 Experiment Background

Testing with complex algorithms is without a doubt an excellent benchmark for a language's execution speed. However, programs with significant size and complexity will usually introduce biases such as programming style, code design, and so on which can play a significant role in tainting the results of the benchmark test. Two major factors that can affect the benchmark test of two different languages are coding style and architecture [5]. This is especially true for benchmarking large applications as each programming language dictates how a program will be set up and executed. As a way to gain an insight into how fast each language runs an experiment that disregards coding style and architecture must be employed.

2.2 Experiment

To determine how much faster Julia is compared to Python a set of rudimentary operations will be carried out. A set of test runs were timed using native functionality for both Julia and Python.

For Julia, @time was used to measure the execution speed of a code block. The @time command in Julia is a Macro that will return the time it took to run an expression as well as the total number of bytes that were allocated for the execution before the value of the expression is returned [6]. All code that was written in Julia was executed in the Julia shell.

In terms of Python, a script was created in IDLE. Time was measured in the Python script by recording the difference from the time at the start of the script's execution to the end of the execution of the script. Though the import of the time library may add extra time to the overall execution time of the Python script, the extra time will be ignored due to time measurements being taken right before the operation is performed and after the operation is finished. Regardless, this gives a better one-to-one look at the raw execution speeds of the operation for both languages.

The tests measured the execution time of a for loop that counts for 1 to 4, a control statement, and a basic math computation. Though the list is not a comprehensive list of all the functionality that the languages offer, these are core attributes that programmers will see in their day-to-day activities. To test the execution speeds, each code block was run 10 times. The run time for each code block is recorded in Table 2. The slowest, fastest, and average execution times for each of the ten cycles were recorded.

2.3 Setup

As logic will dictate physical computing power will play a role in the execution speeds of a language. As such, if the experiment presented in this article is reproduced the execution speeds may vary; however, the general difference should still favor the fastest language. The experiments for this study were conducted on a standard HP laptop computer that has an i5 processor and 8GB of RAM. In short, the machine is similar to what would be found in a normal development environment.

2.4 Code

The code that was used to carry out the tests can be found in Table 1.

Table 1

Test Code for Julia and Python

Code	Julia	Python
Control	@time begin x = 3 if x == 3 print("done") end end	import time start = time.time() x = 3 if x == 3: print("done") end = time.time() print(end - start)
	@time begin for i in 1:4	import time start = time.time()

For Loop	print(i) end end	for i in range(1,4): print(i) end = time.time() print(end - start)
Print a math statement	@time begin print(3+3) end	import time start = time.time() print(3+3) end = time.time() print(end - start)

As can be seen, the code for the examples is very simple and due to the simplicity of the code biases such as architecture and style are non-existent to minimal at worst.

3.1 Results

The results were collected by running each respected section a total of ten times. As such, the Python script was run ten times as well as the Julia snippet. The data in Table 2 is the resultant of the ten execution cycles. Table 2 logs the fastest, slowest, and average execution times of the runs.

Table 2.

Time Comparison

Time	Julia	Python
	For Loops	
Fastest Time Secs	0.000502	0.0109804
Slowest Time Secs	0.000997	0.0309765
Average Time Secs	0.0005879	0.02178407
	Flow Control	
Fastest Time Secs	0.000143	0.0110004
Slowest Time Secs	0.003487	0.0269847
Average Time Secs	0.000698	0.01478628
	Print Math Statement	
Fastest Time Secs	0.00015	0.008036
Slowest Time Secs	0.001476	0.026996
Average Time Secs	0.000454	0.01529

As can be seen, when it comes to basic commands, commands that will be used to build more complex algorithms Julia does have a significant execution advantage. From the data in Table 2, it can be deduced that the average differences between each category range from 0.0141 seconds to .0212 seconds with the biggest advantage in execution stemming from loops with a .0212 second difference favoring Julia. The closest execution speeds were in the control statement group with a 0.0141-second difference in favor of Julia. In short, in terms of pure execution, Julia is superior to Python.

4.1 Future Work

Another area to explore in future research is to study how computing power and computing environments influence the speed differential between the two programming languages. Most literature focus on benchmarking the languages on a single computer and as such all the differentials will be based on the same hardware and environment. With Julia being such a new language future research should focus on its speed in different computing environments such as environments running other operating systems, running Julia in containers, and so on.

4.2 Conclusion

In terms of this study, Julia does live up to its claims of being faster than Python. As can be seen in the code in Table 1, barring the boilerplate code for timing purposes both languages share a similar syntax. In terms of execution speed, the biggest difference in runtime between the two languages was in the for loop category with a .0212 second difference. Even the closest time difference still had a 0.0141 second time difference with the fastest execution speed in favor of Julia. What can be deduced is that Julia is faster; however, there are still drawbacks to using Julia. Due to the immaturity of the language, there are still a lot of unknowns when it comes to how Julia performs in different conditions, an immature ecosystem, and a general lack of knowledge of the language. However what can be deduced is that as the language ages and the ecosystem, knowledge base, and general interest in the language matures, Julia may become a superior alternative to Python especially in fields where the expressive syntax of Python and execution speeds of C++ are desired.

References

[1] Miller, S. (2018). *MIT-created programming language Julia 1.0 debuts.* https://news.mit.edu/2018/mit-developed-julia-programming-language-debuts-juliacon-0827

[2] The Julia programming language. https://julialang.org/

[3] The computer benchmarks game https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/julia-python3.html

[4] Oliphant, T. E. (2007). Python for scientific computing. *Computing in science & engineering*, *9*(3), 10-20.

[5] Sells, R. (2020, March). Julia Programming Language Benchmark Using a Flight Simulation. In *2020 IEEE Aerospace Conference* (pp. 1-8). IEEE.

[6] JLHUB. http://www.jlhub.com/julia/manual/en/function/function/colon-at-time