Tyler Crabtree

CS 355

Wayne Cochran

Julia: An Academic Approach to Programming

Julia is a dynamically typed language that pastiched the superlative aspects of other languages such as: MATLAB, LISP, Python, Perl, and Ruby[1]. The inception of Julia was unique compared to many other mainstream languages. The development of Julia took place at MIT, an academic institution. Generally, programming languages are created by private corporations or independent computer scientists. These groups typically leave unnecessary peccadilloes within the languages they create (such as C); Julia had a scientific and academic approach to its conception which produced a language that understands brevity and palpability.

Invented in 2012, Julia was designed with the intention to help palliate the offensive syntax of other languages. Alan Edelman is one of the designers accredited with constructing Julia. Edelman has credentials in mathematics, computer science, and most importantly education. Edelman's experience in education provides an academic approach for the structure of the language. Research was done to limit error inducing syntax. Syntax of other languages that was considered difficult to understand or often mistyped was removed or changed when added to Julia. The framework of Julia was specifically designed to be easily learned. The dynamic typing is an example of a user-friendly design choice that also enhances the terseness of the syntax[2]. Despite being dynamically typed and having a focus on palpability, Julia's performance often surpasses better known languages.

Dynamic typing has a negative connotation for being slower than statically typed languages. This generalization is often true, although not absolute. “Figure 1,” displays empirical evidence showing that Julia’s just-in-time (JIT) compiler can match the performance of C.

	Fortran gcc 5.1.1	Julia 0.4.0	Python 3.4.3	R 3.2.2	Matlab R2015b	Octave 4.0.0	Mathe-matica 10.2.0	JavaScript V8 3.28.71.19	Go go1.5	LuaJIT gsl-shell 2.3.1	Java 1.8.0_45
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36	1.86	1.71	1.21
parse_int	5.05	1.45	17.02	45.73	802.52	9581.44	15.02	6.06	1.20	5.77	3.35
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70	1.29	2.03	2.60
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66	1.11	0.67	1.35
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01	1.00	1.00	1.00
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30	2.96	3.27	3.92
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07	1.42	1.16	2.36

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

Figure 1

Julia is multi-paradigm language that combining features of imperative, functional, and object-oriented programming while maintaining solid performance[3]. The research at MIT showed that performance is considered be to most important reason programmers gravitate toward or away from a language, this is especially true with scientific languages. Julia improves on the features of previous languages while acting as a fine conglomeration of its predecessors to maximize performance. However, Julia improves on these features to make it more than a simple adopter.

Julia allows a programmer to use specific types when it benefits the programmer. Another feature is represented in “Figure 2,” which shows an isolated function that is able to return multiple values for different variables with one return statement.

```
julia> function((paperFunction(x, y)))
       x = sqrt(x)
       y = x^2 +3y + 1
       return x,y
       end
paperFunction (generic function with 3 methods)

julia> paperFunction(5, 4)
(2.23606797749979,18.0)
```

Figure 2

Python is known for containing both these capabilities; Julia simply can execute these ideas quicker (Julia's performance exceeds Python in every category in "Figure 1"). Both these functionalities represent academic design choices, to design an intuitive and powerful language. Julia is also a free and open source language (through MIT). This accessibility complements the idea of being an intuitive language; Julia's design and accessibility presents ideal circumstances for learning a foreign language[4].

The academic environment that gave birth to Julia integrated design choices that induced learnability of the syntax. For being a dynamically typed language, Julia's performance approaches the level of C. Julia performs as an improved mixture of other notable languages. For example, Python's ability to have multiple return values for one isolated function was borrowed. Julia is a modern language, and the creators at MIT used this to advance the language by borrowing from: MATLAB, LISP, Python, Perl, and Ruby. Essentially, Julia's academic creation formed an easily learned language that optimizes performance despite being dynamically typed.

Citation

1. "Why We Created Julia". *Julia website*. February 2012. Retrieved April 8, 2013. http://julialang.org/blog/2012/02/why-we-created-julia
2. "Introduction". *The Julia Manual*. February 2012. Retrieved April 8, 2013. http://docs.julialang.org/en/release-0.4/manual/introduction/
3. “Julia: A Fast Dynamic Language for Technical Computing” Research Paper Retrieved April 9, 2013. http://julialang.org/images/julia-dynamic-2012-tr.pdf
4. “Julia Lectures” MIT Postings (Youtube). Retrieved April 9 2016 https://www.youtube.com/playlist?list=PLP8iPy9hna6Sdx4soiGrSefrmOPdUWixM