Perl 6: Project: Part 2

Elliot Chandler (Wallace)

19a20 Sept. 2017

Abstract

This article introduces and summarizes the Perl 6 programming language and some of its various features and capabilities, as well as discussing its applications, effective uses, popularity, prospects, and limitations.

Perl 6 is a programming language first released at the end of 2015. It was originally conceived as a revision of the Perl programming language, but because it evolved during the course of its development (begun in 2000). It uses a specification and test-suite driven language development process, rather than an implementation-driven process. Consequently, there is not a single official implementation, although the Rakudo implementation is currently the most complete implementation available, and is the only implementation that is currently available. Perl 6 is a general purpose language that by default for modules is just-in-time compiled to bytecode, and for scripts is interpreted, although this can be changed. The language is implemented using a subset language called NQP, which is run by a virtual machine. There are a few virtual machine implementations available, including a native one, one running on the Java Virtual Machine, and one for JavaScript. The native one, "MoarVM", is the most complete of these. [project, 2017] Perl 6 has many complex and capable programming constructs included

incomplete sentence

v Needs paragraph

Why is it the "most complete"?

with it. Consequently, it seems quite capable to me. The main detriment compared to other programming languages in general that it seems very module-centric, and that it is fairly a new language, and consequently that it does not have a substantial amount of documentation available for it. Another issue that bothers me specifically, is that it lacks strong facilities out of the box for use as a shell REPL environment and shell-style use for executing external shell commands and manipulating their output, although this is a problem shared by many languages that are not purpose-built as shell languages. The problem with being a module-centric language, exacerbated by it being a new language, is that packages for it are not available easily as native packages. This is also a common issue with languages that come with their own package managers, such as .NET's NuGet, Node.js's npm, and Python's pip. The consequence of this is that there is no significant pressure on package maintainers to keep their packages up to date and interoperable, because any given app only needs to work with one version of each package on which it depends. That system, however, breaks down when an app depends on two packages, each of which depend on a conflicting version of the same package, and one wants to avoid having outdated versions of the packages installed on one's system: the standard model of an elegant one-minor-version-per-package used by system package managers is broken, and rendered unable to ship the app properly. This should be in its own paragraph. Maybe try a Pros and Cons breakdown.

While being a new language could conceivably be a problem because of a lack of available libraries for a language, this problem is avoided by Perl 6 by including a system for interacting with "C, Python, Perl 5, and Ruby" [Lenz, 2017, p. 2]. Perl 6 contains a wide range of capabilities and features. The feature that interested me the most is grammars. That is what made me choose to pursue learning about the language. Perl 6's grammars allow writing code that parses a language and returns an abstract syntax tree without having to manually write the parser: rather, one can write a set of rules that combine to express the syntax of the language. This provides a powerful facility for being able to handle

problems of implementing languages and document formats, and consequently is something that interests me, as I have a document format that I would like to implement. The rules in Perl 6 are created using *patterns*, which are somewhat similar to regular expressions in other languages, and generally take their place in Perl 6, although Perl 5–style regular expressions are still available in Perl 6 in case they are needed. Perl 6 has many other features, as well, that contribute to it being effective and capable at solving a wide range of other problems. Its mascot is a butterfly named Camelia [Documentation, 2017a]. It integrates several techniques different from the way Perl works, such as using lazy evaluation. This creates some changes in the way the language operates compared to Perl, and consequently it is somewhat different, with some language constructs not being available in Perl 6, thus necessitating working around their absence. My lack of familiarity with Perl prevents my assessing thoroughly the extent of this as being a problem, but apparently the wantsarray construct from Perl is not available in Perl 6, according to the language FAQ [Documentation, 2017c]. Perl 6 uses (limited-precision [News, 2016]) rational numbers by default, rather than floating point, so that decimals will follow the traditional rules of arithmetic rather than the slightly strange behavior of floating-point numbers (floating-point numbers will produce unexpected results in some cases when doing arithmetic on them [Overflow, 2010–2013]), but at the cost of performance. Floating-point numbers can be used if they are desired, though.

v Support for this claim?

Could be replaced with a "For example"

Perl 6 *grammars* are an extremely expressive tool for writing parsers for languages and document formats. [Documentation, 2017a] They are a way to specify the syntax of a language in such a way that given a grammar and a string to parse using it, the Perl 6 language can build an abstract syntax tree that can be used by code generators for compiling a language, or by other tools that can work with abstract syntax trees. A grammar is a set of patterns (which can be named, and reference each other) using which the input string is parsed. A Perl 6 grammar is a class, so it can be used as other classes. To allow better expressivity from

grammars, *actions* are available (a type of class that allows the parse process to call its methods). [Documentation, 2017b] This allows hooking into the parse process with other code, to allow a wide range of uses outside the traditional scope of grammars.

Perl 6 seems to have good prospects ahead of it, being a new language. Now that it has been released, it will presumably see extensive testing, and consequent development, due to its being in more widespread use.

References

Perl 6 Documentation. Perl 6 documentation. 19 Sept. 2017a. The official Perl 6 documentation is very helpful for getting an understanding of the language.

Perl 6 Documentation. Grammars. 19 Sept. 2017b. This documentation page was one of my primary sources for getting an understanding of the scope and capabilities of grammars in Perl 6.

Perl 6 Documentation. FAQ: Frequently asked questions about Perl 6. 19 Sept. 2017c. I used this article to learn some basic information about Perl 6.

Ruth Holloway. The must-have features for Perl 6. 23 Sept. 2016. This article provided a resource for learning about some of the things that distinguish and differentiate Perl 6 from other languages.

jnthnwrthngtn. Grammar::Tracer and Grammar::Debugger. *Perl 6 Advent Calendar*, 2 Dec. 2011. Debugging grammars is hard out of the box.

The Perl 6 Programming Language. The perl 6 programming language. n.d. This home page for the Perl 6 language introduces it and its culture. Retrieved 20 Sept. 2017.

Moritz Lenz. How to debug a perl 6 grammar. 1 Sept. 2010. This is more information on debugging Perl 6 grammars.

Moritz Lenz. *Perl 6 Fundamentals: A Primer with Examples, Projects, and Case Studies*. doi:10.1007/978-1-4842-2899-9, 2017. ISBN 978-1-4842-2899-9. This book provides a general introduction to Perl 6. It seems quite accessible to me.

Evan Miller. A review of Perl 6. 13 Aug. 2017. This article discusses Perl 6's merits and weaknesses.

Hacker News. Why hasn't perl 6 taken off yet? Nov. 2016. This is where I learned that Perl 6 Rationals aren't arbitrary-precision.

Stack Overflow. Floating point inaccuracy examples. 2010–2013. A citation for the claim regarding floating points being a little odd.

Perl 6 project. Nqp - not quite perl (6): Readme.pod. 31 May 2017. This is where I got information about how Perl 6 is implemented, and its portability.

Laurent Rosenfeld and Allen B. Downey. Think perl 6. 2017. I used this book as a supplementary resource for getting an overview of Perl 6.

Wikipedia. Perl 6. I used Wikipedia's article on Perl 6 as a way to find other articles and books to look at (as well as Fogler Library and Web search).