**Red Programming Language**

Sean O’Brien, Luke Smith

**1 Brief History**

Red was first introduced in 2011 by developer Nenad Rakocevic [1]. Rakocevic began programming as a teenager but was unhappy with the what he felt were overly-complicated languages and tools. In 1999 Rakocevic discovered a Lisp-derivative language called Rebol, which he felt was a much simpler alterative to other popular languages of the time. However, Rebol was a closed-source language, which meant that the community could not contribute to improvements. Additionally, any improvements to be found were slow.

Rakocevic felt that competitor languages such as Perl, Python, and Ruby were only becoming popular because they were open-source. This allowed faster updates, crowdsourced improvements, and greater community engagement. Frustrated with this paradigm, Rakocevic began work on an open-source variant of Rebol that he called Red [2].

Though work began in 2011, Red is still very much a new language. It is currently in version 1.1, and still has a great deal of work to be done. The developers have completed the system compiler, lexical scanner, and the interpreter. However, they are still working on garbage collection, datatypes, I/O, GUI support, the backends, and more [3]. That being said, the language is currently Turing-complete and usable, and is already being used by a small group of early adopters and fans [2].

(Way more stuff at <http://www.red-lang.org/2018/01/overview-of-red-development-history.html>)

**2 General Language Overview**

A general language overview: paradigm, syntax, purpose, common uses, etc.

(This section was written for “history” but ended up being more appropriate for general overview.)

Red is designed to be a “full stack” programming language. That is, while other languages are designed for specific uses, Red is meant to be a general-purpose programming solution [3]. It is meant to be used for any programming tasks, everything from low-level system work to high-level scripting. It is a language that is human-friendly and understandable. It is homoiconic, which means it is its own meta-language and data-format. Red is a functional, imperative language, and supports reactive and symbolic programming. It also uses prototype-based object support and has a low memory footprint [1].

**3 Language Evaluation**

Sebesta’s Concepts of Programming Languages gives us a set of criteria for evaluating languages, divided generally among readability, writability, and reliability. Though the criteria themselves are somewhat controversial, they are useful to begin a discussion on the merits and drawbacks of a language [5]. We have chosen three such criteria here as a starting point on the characteristics of Red.

**3.1 Simplicity**

Lots of language constructs? A few? Feature multiplicity? Operator overloading?

**3.2 Support for Abstraction**

Ability to define and use complicated structures in ways that allow many details to be ignored.

**3.3 Data Types**

Lots of data types in this language (50)? Could be really bad for orthogonality.

**4 Strengths and Weaknesses of the Language**

Put an introduction here, or just scrap the subtopics and make it one section.

**4.1 Strengths**

Blah

**4.2 Weaknesses**

Blah

**5 Individual Opinions**

Your individual, overall opinions of the language, supported by explanations. Each person separately.

**5.1 Sean’s Opinion**

Blah

**5.2 Luke’s Opinion**

Blah

**6 Errata**

Anything else worth mentioning about the language. Can remove.

**7 Conclusion**

Brief wrap-up. Should probably not remove.

**8 Works Cited**

Use the built-in tools.