HACETTEPE UNIVERSITY

DEPARTMENT OF

COMPUTER ENGINEERING

BBM301 Programming Languages Assignment 1 Evaluating a New Programming Language

Author
Kayla AKYÜZ
21726914
b21726914@cs.hacettepe.edu.tr

Advisors
Prof. Pinar DUYGULU SAHIN
pinar@cs.hacettepe.edu.tr
A. Prof. Nazli IKIZLER-CINBIS
nazli@cs.hacettepe.edu.tr

November 12,2020

Ruby Programming Language

Ruby is a general purpose programming language created by a Japanese computer scientist named Yukihiro Matsumoto. Matsumoto states he got the idea back in 1993 when he was talking with a colleague. He later describes that moment as:

> *"I was talking with my colleague about the possibility of an object-oriented scripting language. I knew Perl (Perl4, not Perl5), but I didn't like it really, because it had the smell of a toy language (it still has). The object-oriented language seemed very promising. I knew Python then. But I didn't like it, because I didn't think it was a true object-oriented language – OO features appeared to be add-on to the language. As a language maniac and OO fan for 15 years, I really wanted a genuine object-oriented, easy-to-use scripting language. I looked for but couldn't find one. So I decided to make it."*

2 years after first thoughts of creating a language on December 21, 1995 he published first version of Ruby 0.95. Even at this stage language presented many features like object-oriented design, classes with inheritance, mixins, iterators, closures, exception handling and garbage collection. Currently language is at version 2.7.2. In 2020 version 3.0 release is expected. Uniqueness of Ruby is driven from the inspiration idea of its creation "user-friendliness". Even at performance costs Ruby always provides easiest usage and simplest syntax.

In these days Ruby may be mainly known for web applications like Ruby on Rails, Metasploit however it is a general purpose language so it can be used to create variety of programs.

Why I chose Ruby ?

I chose Ruby to examine in this assignment because it was one of the most popular language that is also easy to learn and is general-purpose coding. I have heard about Ruby way too many times for me to not chose it. First look at syntax also gave me an impression that this language was capable of coding I didn't even imagined of. I have always interested in using words as operators even with Python and I think in the future we will be able to code as we speak. Ruby seems to be a new and promising language which will get us closer to future of programming.

Now that I have partially completed my assignment and research, the more I learn about Ruby language the more it seems to be capable of. However its unpopularity and freshness is evident when research result are harder to find because of getting overwhelmed by ruby gemstones web results.

Evaluation of Ruby

Overall Simplicity ✓

First of all Ruby provides many words as operators like 'plus', 'sum'. This may seem to increase complexity at first but Ruby retains simplicity with simple humanoid syntax.

```
class Numeric
  def minus(x)
    self. -(x)
  end
end
sum = 10.minus 3
```

Orthogonality ✓

Ruby once more takes programming to next level with extreme orthogonal structure of itself. With Ruby you can assign statements to variables. Default orthogonality is already present.

```
a = 5
x = if a == 5
"Five"
else
"Not Five"
end
```

Data Types and Structures ✓

Ruby provides variety of data types because it is a pure object oriented language. Primarily it has 8 data types and 3 additional data types created from Numeric superclass. Ruby's primitive data types are "Numbers, Boolean, Strings, Hashes, Arrays, Symbols". In Ruby everything is an object even nil.

Syntax Considerations ✓

Being the most user friendly programming language Ruby has reserved special words that are easy to understand even for unfamiliar programmers. Ruby does not present heavy syntax rules. For ambiguous statements placement of white space can deter the meaning.

```
# Printing 1 2 3 4
def foo(x) x + 1 end

a = 0
b = 2

puts(a + b)
10/5 # This does no thing.
puts(foo +b) # Notice white space.

BEGIN{puts(1)}
END{puts(4)}
```

Readability ✓

Passing all of the criteria Ruby is a readable language. Programmers can increase their codes readability with usage of indentations, ternary expressions, keyword arguments, string interpolation and many more.

Support for Abstraction ✓

Ruby support abstraction and compositions of abstractions.

```
class Numeric
  def calculate(x)
    puts("Beginning calculation")
    return self. -(x)*x^3
  end
end

def checkValid(x)
  x == -53 ? true : false
end
puts("Calculation is: ", checkValid(1.calculate 8))
```

Data Abstraction ✓

Being an object oriented language Ruby supports data abstraction.

Process Abstraction ✓

Ruby also support process abstraction. Array class can give example to both data abstraction and process abstraction.

```
array = [1,2,3,4]
p arr.class
p arr.sort
```

Expressivity ✓

Ruby does not provide increment operator however ranking # 34 most expressive language it surely can do more with less compared to C [# 50], C++ [# 45], Java [# 44]. It lacks against Python [# 27] however when it comes to creating webapps amusing micro frameworks can be created with Ruby.

Writability ✓

Expressing complexity with ease, having simple syntax rules and ability to coding with humanoid typing Ruby seems to pass writability criteria as well.

Type Checking ✗

Main language itself having a weak type checking and usual throwings of "NoMethodError" Ruby might seem to fail this criteria however Sorbet, a powerful type checking solution, satisfies programmers needs.

Exception Handling ✓

Ruby provides exception handling and checks for standard errors.

```
def divide(a)
  begin
    if a == 0
      raise 'Number 0 chosen!'
    rescue
      puts 'Chose a number other than 0!'
    end
end
```

Aliasing ✓

Ruby supports aliasing

Reliability ✗

Overall reliability of Ruby seems to be lacking at standard level. May be in the future as languages matures more it would get more reliable.

Cost ✓

There is no cost for learning Ruby as free lessons are available. There is nmo usage cost either since Ruby is free language.

Portability ✓

Ruby language and programs can be ported to multiple platforms without need to modification of main code.

Generality ✓

Ruby is mainly used for developing web applications however it is a general purpose language so it can be used in different types of applications.

Well-definedness ✗

Official documentations are complete and language retains completeness even with frequent updates by improving itself. However it seems Ruby has a long way to go before ever coming updates are completed.

Sample Code

```
a = 0
s = String.new
# It feels better to being able to write this instead of new()

def hello
# Again when there is no input for a function we don't need to put ()
  puts("Hello")
  # This and many abilities even with some cost is amazing.
end

hello
# Variables, functions everything in this language is an object so I can
# type this. This much of freedom might be easy to code and be creative
# but for debugging or reading it might cause problems. However I can
# also use it this way:
hello() # Freedom of syntax.
# This language seems very strong but it's might be weakness to
# programmers used to obey heavy syntax regulations.
a = 'b'
def calculator(a)
  a > 0 ? a - 1 : a + 1
  # No need to type return statement result of last line will automatically
  # returned.
end
# Working with languages where even return types of functions need to
# be declared this is so freeing.
calculator(a.to_i) # Even shorter than to_int

x = if true
5.1 end
ary = [] # Arrays
while x > 0
  ary.insert 0,x
  x -= 1
end
ary.each do |i|
 puts i
end
hsh = names = { "luke" => 0xf00, "anakin" => 0x0f0, "rey" => 0x00f }
# Ruby it self has hashes as primitive data type
hsh.each do |key, value|
 print key, " is ", value, "\n"
end
animals = {:s1 => "cat", :a2 => "Dog", :c3 => "GOOSE"}
# Symbols in Ruby basically dictionaries of Python
puts animals[:s1]
puts animals[:a2]
puts animals[:c3]
```

Identifiers in Ruby

Definition of identifiers in Ruby is as following:

Ruby identifiers are consist of alphabets, decimal digits, and the underscore character, and begin with a alphabets(including underscore). There are no restrictions on the lengths of Ruby identifiers.

According regular expression for this freeing definition would be:

```
IDENTIFIER : sequence in /[a–zA–Z_]{a–zA–Z0–9_}/.
```

Functions in Ruby

With Ruby there are many ways to type and define a function with different BNF rules. At first this might look annoying that I have chosen Ruby however a close up look at how functions work would result on a BNF mainly dependent on primaries and operations:

```
FUNCTION : OPERATION ["(" [CALL_ARGS] ")"]
         | PRIMARY.OPERATION "(" [CALL_ARGS] ")"
         | PRIMARY :: OPERATION "(" [CALL_ARGS] ")"
         | PRIMARY.OPERATION
         | PRIMARY :: OPERATION
         | super "(" [CALL_ARGS] ")"
         | super
```

Where it gets messy is definition of primary:

```
PRIMARY: "(" COMPSTMT ")"
       | LITERAL
       | VARIABLE
       | PRIMARY :: IDENTIFIER
       | :: IDENTIFIER
       | PRIMARY "[" [ARGS] "]"
       | "[" [ARGS [,]] "]"
       | "{" [ARGS | ASSOCS [,]] "}"
       | return ["(" [CALL_ARGS] ")"]
       | yield ["(" [CALL_ARGS] ")"]
       | defined? "(" ARG ")"
       | FUNCTION
       | FUNCTION "{" ["|" [BLOCK_VAR] "|"] COMPSTMT "}"
       | if EXPR THEN
      COMPSTMT
      {elsif EXPR THEN
      COMPSTMT}
      [else
      COMPSTMT]
      end
       | unless EXPR THEN
      COMPSTMT
      [else
      COMPSTMT]
      end
```

```
| while EXPR DO COMPSTMT end
| until EXPR DO COMPSTMT end
| case COMPSTMT
when WHEN_ARGS THEN COMPSTMT
{when WHEN_ARGS THEN COMPSTMT}
[else
COMPSTMT]
end
 for BLOCK_VAR in EXPR DO
COMPSTMT
end
| begin
COMPSTMT
{rescue [ARGS] DO
COMPSTMT}
[else
COMPSTMT]
[ensure
COMPSTMT]
end
| class IDENTIFIER [< IDENTIFIER]
COMPSTMT
end
| module IDENTIFIER
COMPSTMT
end
| def FNAME ARGDECL
COMPSTMT
end
| def SINGLETON (. | ::) FNAME ARGDECL
COMPSTMT
end
```

As a humanoid language in depth examination of Ruby's BNF rules would be complicated and long. Function calls on the other hand are very simple, by typing the function or command:

```
CALL : FUNCTION
     | COMMAND

COMMAND : OPERATION CALL_ARGS
        | PRIMARY.OPERATION CALL_ARGS
        | PRIMARY :: OPERATION CALL_ARGS
        | super CALL_ARGS
```

REFERENCES

Tiobe Index
LaTex Tutorials
Ruby Language Website
BrandonByars.com
Artima.com
GeeksForGeeks.org
StackOverFlow.com
PLs Ranked by Expressiveness
On Ruby's Expressiveness
Type Checking in Ruby
What is Ruby Used For
Aliasing in Ruby
BNF Sytnax of Ruby
Online Ruby
Ruby Wikipedia Page
Yukihiro Matsumoto Wikipedia Page