THE RUBY PROGRAMMING LANGUAGE

A PREPRINT

Jensen Collins
UA NetID: jcollins1
jcollins1@email.arizona.edu

Makayla Worden
UA NetID: makaylaworden
makaylaworden@email.arizona.edu

April 30, 2020

ABSTRACT

This project focuses on the unique qualities that the Ruby programming language possesses. As it will show, Ruby is a high-level, object oriented language. This means it follows typical features of other common languages, such as Java and Python. Our research of this language was guided by looking at individual characteristics of the language and practicing them by writing code snippets on each topic. Such topics would include looking at a general concept, such as Control Structures, and then finding what Ruby allows its programmers to do within that topic, (i.e. loop techniques, conditional statements, etc.). The results of our research opened the door for us to program a full sized project, which utilizes multiple special features Ruby has. This project uses user input to play the game of Hangman. It will let the player chose a level of difficulty and then chose a prompt for the player to make guesses on. As the player makes guesses, the program will inform them if they guessed correctly. If their guess is incorrect, an ASCII art Hangman will begin to be drawn on their console. This process will repeat until the hangman is drawn in full or the player guesses the word correctly.

1 Introduction

Ruby is a fairly new programming language invented to encourage non-programmers to learn how to code. This means that those experienced with programming will likely find Ruby to be quick to pick up. It follows a similar syntactical structure to other high-level languages and contains almost every feature one can think of a new language to have. With that, Ruby has additional features to help programmers produce code quickly. It has many different ways to produce the same output, but each has a special aspect to it that can be helpful in different situations.

All of these characteristics will be displayed throughout this project. First, we will look into a brief history of Ruby, this will explain why Ruby was created to begin with, and how it has grown in popularity over the last few years. After establishing why Ruby came to be, we will look into what about it is so special. This will include an analysis of the different control structures and data types Ruby has. Finally, we will look into how one can write large programs in Ruby, focusing on the different sub programming techniques it allows.

2 History

Ruby was invented by Yukihiro "Matz" Matsumoto in the 1990s [15]. It was designed based on the idea that coding should be fun for programmers to do in Japan. Ruby was originally designed for beginner programmers to code games. This causes it to have characteristics unlike any other language. Although, Ruby is still object oriented, so its syntax tends to look like other languages (such as Java). Ruby was not made to be a difficult languages to program in. The concept of "being fun" is implemented by taking the difficulties of coding large programs out, and replacing with many different styles of solving a problem. This allows its users to program in a way that makes sense to them. This is how Ruby works to eliminate the idea that "programming is hard".

Currently, Ruby is growing in popularity. A Ruby framework, Ruby on Rails, created by David Hansson, has been created for Ruby programmers to be eased into computer science [15].

3 Control Structures

Ruby's selection statements, in general, are similar to what one sees in other high-level, object-oriented languages. It supports the standard if/else conditional statement, switch statements, variable assignment, return statements, and keywords such as *break* [7]. However, Ruby also supports an interesting assignment statement. The following code based off Stan Lo's in Statements vs Expressions in Ruby[11]:

```
tempBar = class Bar; end
puts(tempBar)
# This will print nil

tempBaz = tempBar = 10
puts(tempBaz)
# This will print 10

tempFoo = if true
45
end
puts(tempFoo)
# This will print 45
```

Here, we can see that one can assign values to multiple variables in the same statement, such as x = y = z. In this situation, Ruby has left associativity, so y = z would be evaluated first, followed by x = y.

This language also supports many kinds of iteration techniques. Such techniques include the standard for and while loops, in addition to iterators, until loops, do/while loops, and recursion. All of these can be cut short using a break statement. Ruby also supports a simple loop, denoted:

```
loop do
# something loopy
end
```

Using this form of loop will cause a program to run indefinitely, until the user manually stops it by hitting Ctrl + c.

While loops and until loops are particularly interesting as well. They are essentially opposites of one another. While loops will loop as long as the condition is not yet met. On the other hand, until loops run until the condition is met. An example of both of these is as follows:

```
# Until Loop
i = 0
until i == 100
    puts "i = #{i}"
        i = i + 1
        end

# While Loop
i = 0
while i < 100
    puts "i = #{i}"
        i = i + 1
end
```

Both of these loops will print the numbers 0 to 100, however their conditional statements are practically opposite.

In summary, Ruby functions as a common high-level language, and its capabilities in control flow are no exception. It supports all standard conditional and iteration techniques, in addition to some of its own unique qualities.

4 Data Types

Due to it being object oriented, Ruby creates its data types through classes. It supports the following data types: arrays, hashes, booleans, numbers (integers, floats, complex, rational, bigDecimal), strings, symbols, and a nil value.[6]

Arrays and hashes are the main two ways to store collections of data. Both are the standard implementation used in many other programming languages. Arrays store values in an ordered fashion, while hashes store based off of key-value pairs. You can declare a new, empty array by doing either of the following:

```
tempo2 = Array.new
tempo1 = []
```

Otherwise, declaring an array with elements can be done by setting it to a list of values in square brackets. These elements do not need to be of the same type. Accessing elements can be done for an array, a, by saying a[index].

To declare a hash, one can either call hash.new, or list out its contents like so:

```
tempo = {
  'a' => 1,
  'b' => 2,
  'c' => 3,
  'd' => 4,
  'e' => 5
}
```

Similar to most, if not all, programming languages booleans in Ruby represent true and false values. These true and false values are singleton objects deriving from TrueClass and FalseClass, respectively. Everything in a boolean context (such as conditional statements) is considered "truthy", except for the two values false and nil. These are considered "falsy". [7]

Ruby's Numeric class is a parent to many subclasses. These subclasses include Integer, Float, Complex, Rational, and BigDecimal. The Integer class is a parent of Fixnum and Bignum. Fixnum is integers that are either 32 or 62 bits while Bignum is used for big numbers. Floats are typical of most other programming languages, consisting of decimal numbers. However, Floats are imprecise, which can cause problems when comparing them. This is why the BigDecimal type was created. BigDecimal numbers are used for perfect precision decimal numbers. Complex numbers in Ruby can be used when working with imaginary numbers. Rational numbers are exactly that, rational numbers. They can be used to represent fractions (i.e. 1/2). [7]

The way Floating point numbers are stored causes them to be imprecise. This means that the numbers a programmer is working with can get altered while it is being put into memory. For example, 0.2 + 0.1 == 0.3 will return false. If we look at what 0.2 + 0.1 returns in Ruby, we'll surprisingly get 0.30000000000000004. This means in order to get precise decimals, the BigDecimal type is required. [7]

The Integer subclasses Fixnum and Bignum also have some remarkable qualities. Bignum is used for storing big number, this is to speed up the time it takes to work with numbers that are not big. Bignum's implementation takes more time that Fixnum's. Fixnum is interesting because it isn't used by creating an object. Fixnum uses an Object ID, which for a number, n, is calculated by (n*2) + 1. This means that the numbers value is never stored in the object, it can just be calculated using that formula off the Object ID.[7]

A String type is represented in Ruby by using either double or single quotes. Unlike most other high-level programming languages, Strings are mutable in Ruby. This means that a programmer can manipulate them directly, rather than creating an entirely new String every time they want to change something about the original. To declare a String, the user can either print it out directly or save it to a variable. This would look something like this:

```
# single quote
```

```
puts 'Hello, world!'

# double quote
puts "Hello, world!"

# use variables to save strings
tempo1 = "hello"
tempo2 = 'world'

puts tempo1
puts tempo2
```

This would result in an output of:

```
Hello, world!
Hello, world!
hello
world
```

To index different characters in a String, one will use a similar syntax to indexing an array. For example, str = "Hello, world!", str[0] will return "H". Using a negative index will start at the end of the string and index backwards.

Symbols are other objects that can be used in several situations. They are declared with a colon in front of the symbol name. These can be used similarly to variables, where Ruby handles its value assignment. Symbols can be used similarly to Strings, since all they store is their internal identification and their name, which is helpful because they tend to be more efficient than Strings. The value of a Symbol will stay the same throughout the duration of a program running, so they are better for checking equality than Strings.[5]

The absence of a value will be referred to as nil in Ruby. This means when a data type is empty, it will have a value of nil. This is the only value other than false that is considered falsy when working with booleans. If a programmer mistakenly attempts to access data that is not inside an array or hash, nil is returned in its place.

5 Subprograms

Because Ruby is perfectly object-oriented, it has all the key features of its language type. These include Data Encapsulation, Data Abstraction, Polymorphism, and Inheritance [4]. Because of this, its programmers can create Class objects throughout the development of their program. This can be done by using the keyword *Class* followed by the object name, and indentation inside the class indicating where the class definition begins and ends. At the end of the class and outside of the indentation, "end" needs to be written to tell the compiler to stop reading everything after as a part of the object. For example, here is an object Animal that contains a method animalSpeak:

```
class Animal
  def animalSpeak
    puts "Animal is talking"
  end
end
```

To initialize an object, the *new* keyword is required.

With being able to define classes, it is important to note to different access levels of variables in Ruby. Local variables are variables only defined inside of a method. These are declared by using all lowercase letters, separating words with underscores. Instance variables are defined in an object declaration, so they can be accessed within that object or from any of instances of it. There are declared with a preceding "@" symbol. Class variables are available to other objects, however it belongs to the class it was defined in, these are declared with preceding "@@" symbols. Finally, there are global variables, there are available to all classes in the program and are defined with a preceding dollar sign symbol. [4]

Ruby methods syntactically follow similarly to most other high-level programming languages. To declare a method, the programmer must write "def method_name". More often than not, a programmer will need to pass parameters into their methods, this can be done as so "def method_name (parameters)". When calling a method with parameters, the programmer can just list the parameter values after the call, for example "method_name val1 val2". [4] Here is an example of a method in Ruby, which prints the contents of an array that was passed in as a parameter:

```
def printArray(*array)
  array.each do |i|
    puts i
  end
end
```

In Ruby, every method returns something. This means that without a specific return statement, Ruby will return the last statement's value [4]. Return statements can return one or more items out of a method. If this is the case, the values will be returned in order, so if a method returned "abc", "a", "b", "c", these values would be received as such "abc", "a", "b", and "c". This is the same for receiving parameters in a method. Each parameter will be taken in the order they are passed in.

Ruby also has installable libraries to assist programmers in what they are wanting to accomplish. In Popular Ruby Libraries, an explanation of a few is given. For debugging, this author recommends installing Byebug, and Bettor_errors, which assist one in fixing errors one has in their code [10]. However, libraries in Ruby do not strictly assist with debugging, for examples Minimagick is for uploading files [10].

6 Summary

In conclusion, Ruby is famous for its usability. Ruby was invented so those who have little knowledge of how to program can strengthen their skillset. Because of this, the entire language is centered around making programming relatively easy. This means that Ruby allows its programmers to "get away with" many programming techniques that may not be allowed in most other languages. Because it is so new, Ruby piggybacks off many good characteristics of other languages. This can be seen in its control structures, data types, and subprograms, each of which was discussed in detail during this paper. These all have certain traits about them that most programmers see in other languages, in addition to items unique to Ruby. Each element this paper discusses can be displayed in our Hangman game. Hangman utilizes in particular, the object oriented build of Ruby. It does this by utilizing a class for the puzzle prompt and the hangman.

[9] [15] [8] [12] [11] [7] [14] [13] [2] [1] [3] [6]

References

[1] Atom. Latest access 2020.

[2] Downloads. Latest access 2020.

[3] Online ruby compiler. Latest access 2020.

[4] Tutorials point. Latest access 2020.

[5] Erwin Aligam. Ruby symbols. 2020.

[6] Jan Bodnar. *ZetCode Ruby data types*. 2020.

[7] Jesus Castello. Ruby guides. 2020.

[8] Code Conquest. Ruby 101: Programming projects for beginners. 2020.

[9] Matthew Ford. Ruby on rails: What it is and why you should use it for your web application. 2014.

[10] HARIKRISHNA KUNDARIYA. Popular ruby libraries. 2020.

[11] Stan Lo. Statements vs expressions in ruby. 2017.

[12] The Odin Project. Ruby programming. 2020.

[13] ruby forum.com. Ruby forum. Latest access 2020.

[14] ruby lang.org. Ruby: A programmers best friend. 2020.

[15] Launch School. *Introduction to Programming with Ruby*. 2020.