RUBY

A PREPRINT

Jensen Collins
jcollins1@email.arizona.edu

Makayla Worden
makaylaworden@email.arizona.edu

April 14, 2020

ABSTRACT

1 Introduction

2 History

Why was Ruby designed? 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. The popularity spread as the code is easy to read as it reads like English and it is a very high-level language which allows for beginners to quickly create something from scratch in less lines of code.

Who designed it? Yukihiro "Matz" Matsumoto in the 1990s

What is Ruby's current status? 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.

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* [6]. However, Ruby also supports an interesting assignment statement. The following code based off Stan Lo's in Statements vs Expressions in Ruby[9]:

```
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.

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.[5]

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”. [6]

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). [6]

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. [6]

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.[6]

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.[4]

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

6 Summary

[8] [13] [7] [10] [9] [6] [12] [11] [2] [1] [3] [5]

References

[1] Atom. Latest access 2020.

[2] Downloads. Latest access 2020.

[3] Online ruby compiler. Latest access 2020.

[4] Erwin Aligam. Ruby symbols. 2020.

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

[6] Jesus Castello. Ruby guides. 2020.

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

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

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

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

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

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

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