Ruby Programming Language

John San Diego

October 9, 2016

Contents

1 BASIC SYNTAX STRUCTURE

1.1 Identifiers

The Lexical structure of Ruby is based on ASCII. Variables in Ruby has no restrictions to the length of the variable name. Ruby variables can compose of letters, and numbers, however, it cannot begin with a number. Variables cannot include whitespaces and non printing numbers. Using the dollar character designates a variable as global meaning it can be used anywhere in the program. Ruby is a case sensitive language meaning that certain keywords which is spelled the same have different meaning when lowercased or uppercased. Using the @ character designates an instance variable. In addition, method with a ? mark at the end designates a boolean expression. Ruby also uses ! character to designate that a method should be used cautiously. The exclamation point are usually used on mutator methods that can alter objects which returns a copy of the original object.[3]

1.2 Basic Syntax

In Ruby, comments are designated with the pound character. The pound character only works until the end of the line. To extend a comment beyond the end of the line, you either have to put another pound character or use unique commenting structure used in Ruby. To get multiple lines of comment you must use the structure =begin (comment) =end. This type of commenting resembles c or java's multiple line comment using /* (comment) */.However, this type of commenting only works if the equal character is in the beginning the begin line. In addition, if a pound character is in a string that pound sign does not designate a comment, it would just be part of the string. [2]

In Ruby, conditionals are made simpler by eliminating expression that are unnecessary such as parenthesis and brackets. Ruby uses a different method. For example, to create an if statement it would look like this.

```
if expression
code
end
```

Another conditional statement in Ruby is the unless statement. The unless statement works the opposite of if statement meaning unless will only work if the condition is false or null.

In addition to unless, Ruby also has a case statement which is a multiway conditional. The case statement is similar to C's or Java's switch statement. Instead of using Switch, case, and default. Ruby uses case, when, and else for its clauses. For example,

```
case x
when x == 1 then "one"
when x == 2 then "two"
else "many"
end
```

Ruby uses many different syntax for its structure, however, it is also similar to various other programming language such as, C, JAVA and PERL just to name a few. Ruby incorporates the same paradigms that had been proven to work in the past for everyday usages.

1.3 Syntax Structure

Ruby interpreter uses expressions to produce values. Expression is a group of Ruby code that the Interpreter can evaluate to produce a value. Numbers and strings are called primary expressions because they can be combined to create larger expressions.[3] Compound expression on the other hand are values that are made from arrays, hash and range expressions. For example, [12,13,14] is a compound expression which is an array literal. Also, 1..3 is a range literal used in loops to designate the range of the loop. In addition, expressions in Ruby can be used to create statements such as an if statement and the while loop. For example,

```
if value == 3 then
value = value + 1
end
```

In addition, Ruby doesn't require the use of parenthesis to designate precendence or associativity, but it is optional.

1.4 Block Structure

A new type of syntax in Ruby are called Block. Ruby codes have a block structure meaning statement and methods are nested together in the program. Blocks are implemented using keywords such as .times, do and end.[3] For example, to create a do loop you would have to write:

```
1.upto(10) do |x|
print x
end
```

The .upto syntax designates x as a new variable and it will print x up to 10 and end it. Using block makes it simple to loop through array and linked list where you need a value right away. The syntax do are usually used when a block has more than one line. As stated earlier, blocks can be nested inside a multiple different body type such as a class body, a method body, and an if statement body.[3] For example,

```
class dogs
def age(number)
file.each(number) do |line|
if line[0,1] == "@"
next
end end end end
```

Each end is needed to close each block of code.

2 UNITS AND LEVEL OF SCOPE

2.1 Object Scope

Variables in Ruby has different level of scope. Variables that are instatiated inside a method can only be used inside that method. In addition, variables that are instantiated outside a method cannot be used inside a method unless it is used as a parameter. Lastly, variable that are instantiated after a method cannot be used inside that method. For example,

```
def dog(num)
puts age
end
age = 50
```

```
Will print an error
```

It will print an error because the variable age is outside the scope of the method dog rendering age as an undefined variable. In addition, block variables are always local to their own block meaning a variable can only be used inside the block its on. Unless, the variable becomes a global variable by using the dollar character, a variable scope is dependent on where it is instantiated.

2.2 Primitive Data Types

Everything in Ruby is an object. Every object has a method class that returns that objects class. Ruby works with references meaning it is not the object that we change but a reference to that object that is changed.[4] References in Ruby works similar to C or C++ pointers, however, Ruby does not use pointers. Precedence are needed in operators to eliminate or decrease ambiguity in expression. For example,

```
s = "Ruby" // Create a String object.  Store a reference to it in
s.
t = s // Copy the reference to t.  s and t both refer to the same
object.
t[-1] = "" // Modify the object through the reference in t.
print s // Access the modified object through s.  Prints "Rub".
t = "Java" // t now refers to a different object.
print s,t // Prints "RubJava".
```
[3]

3 OPERATORS

3.1 Precedence

The precedence of an operator affects the order of operation of an expression. In Ruby, order of precedence works exactly the same as PEMDAS. For example, 1+1*6/3 would equal 3 because the operation specifies the which operator evaluates first. However, order of operation can be change by using parenthesis, so you can specify your own order of operation. For example,

```
(1+1)*6/3 would equal 4
```

The parenthesis has a higher precedence than the multiplication, so it evaluates by doing addition first.

3.2 Associativity

Associativity in Ruby are needed when the same operator appears in the same precedence level. Ruby uses left associativity when evaluating most numerical operations, so evaluating 9/3*3 would equal 9. [1] Ruby uses right associativity when assigning values to a variable because if left to right is used evaluating a=0 will not work. Boolean, relational, and bitwise operators are all left associative in Ruby.

References

[1] Jan Bodnar. Expressions. 2014.

[2] Yukihiro Matsumoto. Ruby syntax. 2002.

[3] Yukihiro Matsumoto. The ruby programming language book. 2008.

[4] Wikipedia. Ruby(programming language/data types. 2016.