6.170 Tutorial 3 - Ruby Basics

Prerequisites

1. Have Ruby installed on your computer
   a. If you use Mac/Linux, Ruby should already be preinstalled on your machine.
   b. If you have a Windows Machine, you can install Ruby using the Ruby Installer here http://rubyinstaller.org

Note: Having completed Tutorial 0, Ruby should already be installed on your computer.

Goals of this Tutorial

Become familiar with the Ruby programming language.

Useful Resources for Ruby

1. Chapter 4 of Ruby on Rails Tutorial http://ruby.railstutorial.org/ruby-on-rails-tutorial-book
2. Interactive Code School Tutorial http://www.codeschool.com/paths/ruby
3. Learning Ruby

http://proquest.safaribooksonline.com/book/web-development/ruby/9780596529864

[Note: Need MIT Certificate for this]

4. Programming Ruby http://www.ruby-doc.org/docs/ProgrammingRuby/

Topic 1: Basic Information about Ruby

1. What is it?
   a. It is a programming language invented in the mid-1990s by Yukihiro Matsumoto
   b. It is a general purpose object-oriented programming language that has a syntax influenced by Perl/Python/Lisp
   c. Similar to Python, it is an interpreted language, meaning that its code is executed line by line during runtime (Rather than being compiled)

2. Pros of using Ruby
   a. Allows for simple and fast creation of web applications
   b. Access to a library of over 50,000 RubyGems
   c. “Concise, Elegant, and Powerful”

3. Cons of using Ruby
   a. Ruby runs quite slow compared to other languages. In fact, it can be up to 20 times slower than Java when processing
   b. Poor Standard library documentation

Topic 2: Ruby vs Python

1. Philosophy
    a. Python was designed to emphasize programming productivity and code readability
    b. Ruby was designed to “make programming fun for its creator”

2. Object Oriented Programming
    a. Both programming languages support object oriented programming
    b. In Ruby, an instance of literally any type is an object. For instance, in Ruby, C Class and Module instances are objects, with Classes and Modules themselves being objects.
    c. However, where in Ruby all functions and most operators are in fact methods of an object, in python functions are first-class object themselves. A function in Python is typically declared as a member of another object (typically a Class, Instance, or Module), but can be freely passed around and attached to other objects at will.

3. Syntax
    a. Python has nice functional features such as list comprehension and generators. Ruby doesn’t have these.
    b. However, Ruby gives you more freedom in how you write your code: parenthesis are optional and blocks aren’t whitespace delimited like they are in Python

4. Speed
    a. Python generally executes code faster than Ruby

5. Applications
    a. Python sees use in automating system administration and software development, web application development, analyzing scientific data (with help of numpy, scipy, and matplotlib modules), biostatisics, and teaching introductory computer science and programming.
    b. Ruby sees use in web application development, and general programming.

Topic 3: Lexical Conventions of Ruby

1. White Space
    a. Whitespace characters such as spaces and tabs are generally ignored in Ruby code, except when they appear in strings.

2. Line Endings
    a. Ruby interprets semicolons and newline characters as the ending of a statement.
    However, if Ruby encounters operators, such as +, - , or backslash at the

end of a line, they indicate the continuation of a statement.

b. Be careful when dealing with expressions spanning multiple lines. Remember to always append a '+' at the end of a line if the expression is continued on the line below it.

c. Ex: a = (4 + 5) //prints out 9

d. Ex: a = (4
+ 5) //prints out 5 **[NOTE THE DIFFERENCE]**

The reason this happens is because the expression is treated as a = (4; +5). That is, it evaluates a "4" and then evaluates a "+5" (which then is the resultant value, as it was the last evaluated).

e. Ex: a = (4 +
5) //prints out 9

3. Comments
   a. Comments are lines of annotation within Ruby code that are ignored at runtime. Comments extend from # to the end of the line.
   b. Ex: # This is a comment
   c. Ruby code can also contain embedded documents too. Embedded documents extend from a line beginning with =begin to the next line beginning with -end. =begin and =end must come at the beginning of a line.
   d. Ex: =begin
      This is an embedded document.
      =end

4. Identifiers
   a. Identifiers are names of variables, constants, and methods
   b. Ruby distinguishes between identifiers consisting of uppercase characters and those of lowercase characters.
   c. Identifier names may consist of alphanumeric characters and the underscore character (_).
   d. You can distinguish a variable's type by the initial character of its identifier

5. Reserved words
   a. Ruby has a set of reserved words that may not be used as constant or local variable names.
   b. List of reserved words http://www.tutorialspoint.com/ruby/ruby_quick_guide.htm

Topic 4: Ruby is Object-Oriented

Ruby has *classes*. Classes hold data—in the form of variables and constants—and methods, which are compact collections of code that help you perform operations on data. Classes can inherit information from each other, but only one at a time. This allows you to reuse code—which means you'll spend less time fixing or debugging code—and intermix the code through inheritance.

In Ruby, almost everything is an object. In fact, everything that Ruby can bind to a variable name is an object.

Let’s do an example.

```
class Hello
   def howdy
       greeting=  “Hello, Matz!”
       puts greeting
   end
end

class Goodbye < Hello
    def solong
        greeting = “Goodbye, Matz!”
        puts greeting
     end
end

friendly = Goodbye.new
friendly.howdy
friendly.solong
```

If we run the code above, we will get back:
Hello, Matz!
Goodbye, Matz.

The reason is:

The Hello class defines the howdy method. The Goodbye class likewise contains the definition of a method, solong. The Goodbye class also *inherits* what's in the Hello class (that's what the < symbol is for). This means that the Goodbye class didn't have to redefine the howdy method. It just inherited it.

friendly is an object (an instance of the Goodbye class). You can use the friendly object to call both the howdy and solong methods, because it inherently knows about them. It knows about the solong method because it is defined inside the Goodbye class, and it knows about the howdy method because Goodbye inherited it from Hello.

Topic 5: Declaring Variables

1. Ruby doesn't have type declarations. It just assigns values to variables
   a. Example: months = 12

2. Local Variables:
   a. Must start with a lowercase letter or with an underscore character
   b. Example 1: alpha
   c. Example 2: _beta
   d. When a local variable is defined inside a method or a loop, its scope is within the method or loop where it was defined.

3. Instance Variables
   a. An instance variable is a variable that is referenced via an instance of a class and therefore belongs to a given object.
   b. It is always prefixed by a single at (@) sign
   c. Example: @hello = 6170
   d. You can access instance variables from outside of their class only by accessor methods

4. Class Variables
   a. A instances of a class. Only one copy of a class variable exists for a given class.
   b. It is always prefixed by two at signs (@@)
   c. You have to initialize a class variable before you use it
   d. Example: @@times = 0

5. Global Variables
   a. Global variables are available globally to a program, inside any structure. Their scope is the whole program.
   b. It is always prefixed by a dollar sign ($)
   c. Example: $amount = "0.00"

Topic 6: Math

1. Converting Numbers
   a. Integer(1.4) => 1 // converts a floating-point number
   b. Integer("256") => 256 // converts a string
   c. Integer("0b100") => 8 // converts a binary number from string
   d. Integer(?z) => 122 // converts a character to ASCII

2. Basic Math Operators:
   a. Use + , - , * , / for addition, subtraction, multiplication, and division

b. Use ** for exponent
c. Use % for modulo

3. Division and Truncation
   a. 25 / 2 => 12 // Integer division always rounded down
   b. 25.0 / 2 => 12.5 // Use a float as at least one operand to stop truncation

4. Ranges
   a. The range operator . . means an inclusive range of numbers.
      Example: 1 . . 10 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
   b. The range operator . . . means a range of numbers excluding the last one
      Example: 1 . . . 10 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
   c. Creating an array of a range of numbers:
      (1 . . 9).to_a => [1, 2, 3, 4, 5, 6, 7, 8, 9] //.to_a means “to array”

5. Useful Math methods
   a. Absolute value: -40.abs => 40
   b. Ceiling: 4.65.ceil => 5
   c. Floor: 4.65.floor => 4
   d. Rounding: 100.45.round => 100

Topic 7: String

1. Basic string functions:
   a. title = String.new // sets title = “”
   b. title.empty? // returns boolean showing whether title is empty (“”)
   c. title.length // returns length of title

2. Concatenating Strings
   a. “Hello,” “ “ “Matz” “!” // Creates “Hello, Matz!”
   b. “Hello,” + “ “ + “Matz” + “!” // Similarly, creates “Hello, Matz!”
   c. “Hello, “ << “Matz!” // Again, creates “Hello, Matz!”
   d. “Hello, “.concat “Matz!” // Also creates “Hello, Matz!”

3. Accessing Strings
   a. line = “A horse! a horse! my kingdom for a horse!”
   b. line[18, 23] => “my kingdom for a horse!” // 18 = starting index location,
      // 23 = number of letters to retrieve
   c. line.index(“k”) => 21 // returns 1st index location where “k” appears

4. Comparisons
   a. hay == nicolay // Compares hay with nicolay. Returns false.
   b. hay.eql? nicolay // Compares hay with nicolay. Returns false.
   c. “a” <=> “v” // <=> is spaceship operator. Returns 0 if equal,
      -1 if less than, and 1 if greater than

5. Inserting a String in a String
   a. “Be carful .” .insert 6, “e” => “Be careful.” //Insert “e” at index 6
6. Substitute a Substring
   a. “That’s alll fokes”.gsub “alll”, “all” => “That’s all folks”

7. Reversing a String:
   a. “abcdefghijkl”.reverse => “lkjihgfedcba”

Topic 8: Arrays

1. The Array class is one of Ruby’s built-in classes. Ruby arrays can hold objects such as String, Integer, Fixnum, Hash, Symbol, even other Array objects. Any object that Ruby can create, it can hold in an array.

2. Adding elements to a Ruby array will cause it to automatically increase in size

3. A Ruby array can hold objects of all different types, not just one type

4. Basic Array Functions
   a. months = Array.new // Creates a new array [ ]
   b. months.empty? // Returns boolean of whether months is empty
   c. months.size // Returns size of array
   d. months.length // Also returns size of array
   e. months.clear // Clears the array to [ ]
   f. months = Array.[ ] (“jan”, “feb”, “mar”, “dec”) //Creates array months
   g. months = [nil, “Jan”, “Feb”, “Mar”, “Dec”] //Another way to create array
   h. digits = Array(0 . . 9) //Creates array [1, 2, 3, 4, 5, 6, 7, 8, 9]

5. Iterating over Array
   a. sample = [“a”, “b”, “c”, “d”, “z”, 1, 5]
      sample.each { |e| print e } //prints out each element of sample

6. Accessing elements of array
   a. months = [“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Dec”]
   b. months[0] // returns Jan
   c. months[-1] // returns Dec
   d. months[0, 2] // start index = 0, number of elements to extract = 2
      // Returns [“Jan”, “Feb”]
   e. months.include? “Nov” //Returns whether “Nov” is present in months

7. Concatenation
   a. Let q1, q2 be two arrays. You can concatenate both arrays into one array called q3 using: q3 = q1 + q2
   b. yrs = [2010, 2011, 2012]

```
yrs << 2013              // yrs is now [2010, 2011, 2012, 2013]
```

8. Set Operations
   a. Intersection (&) creates a new array, merging the common elements of two arrays
      but removing uncommon elements and duplicates
      Example: year = semester1 & semester2
   b. Difference (-) creates a new array, removing elements that appear in both arrays
      Example: year = semester1 – semester2
   c. Union (|) joins two arrays together, removing duplicates
      Example: year = semester1 | semester2

9. Extracting Unique Elements
   a. shopping_list = [“apples”, “oranges”, “apples”, “bananas”, “oranges”]
      shopping_list.uniq! //Returns [“apples”, “oranges”, “bananas”]

10. Deletions
    a. months = [“Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Dec”]
    b. months.delete(“Jan”) //Deletes element “Jan” from array
    c. months.delete_at(2) //Deletes element at index 2 from array

11. Iterations
    a. Array has an each method that allows you to iterate over every element in the array
    b. Ex: months.each { |e| print e }//prints out every element inside months

12. Sorting an array of numbers
    a. num_list = [2, 5, 1, 7, 23, 99, 14, 27]
    b. num_list.sort! //sorts num_list to [1, 2, 5, 7, 14, 23, 27, 99]

Topic 9: Conditionals

1. If Statements
   a. Example:
   ```
   if x == 256
      puts “x equals 256”
   end
   ```
   b. The operator && means “and”
      Example: if a == 10 && b == 27...
      Alternatively, you can do: if a == 10 and b == 27...
   c. The operator || is used to represent “or”
      Example: if a == 10 || b ==27
      Alternatively: if a == 10 or b == 27...

2. The Ternary Operator:

a. Example: label = length == 1? "argument" : "arguments"
b. This expression assigns a string value to label based on the value of length. If the value of length is 1, then the string value “argument” will be assigned to label; but if it is not true—that is, if length has a value other than 1—then the value of label will be the string “arguments”.

3. While Loops
   a. A while loop executes the code it contains as long as its conditional statement remains true.
   b. Ex: while i < breeds.size do
      temp << breeds[i].capitalize
      i += 1
      end
   c. You can break out of a while loop using the keyword break.
      Ex: while i < breeds.size
      temp << breeds[i].capitalize
      break if temp[i] == “Arabian”
      i += 1
      end

4. For Loops
   a. A for loop executes the body of the code a certain number of times
   b. Ex: for i in 1. . 5 do
      print i, “ “
      end //Prints 1 2 3 4 5
   c. Note: You need to keep the “end” at the end of the loop. However, the “do” is optional
   d. Alternatively, you can use the times method:
      10.times { | i | print i, “ “ } //Prints 1 2 3 4 5 6 7 8 9 10
   e. Use the upto method to count up to a certain number from a starting num:
      1.upto(10) { |i| print i, “ “ } //Prints 1 2 3 4 5 6 7 8 9 10
   f. Use the downto method to count down to a certain num from a start num:
      5.downto(1) { |i| print i, “ “ } //Prints 5 4 3 2 1

Topic 10: Using Blocks

1. What are blocks?
   a. A block consists of chunks of code
   b. You assign a name to a block
   c. The code in the block is always enclosed within braces { }
   d. A block is always invoked from a function with the same name as that of the block. This means that if you have a block with the name test, then you can use the function test to invoke this block.

e. If you are familiar with python, a block in Ruby is similar to a lambda function in python.

f. You invoke a block by using the *yield* statement

2. Syntax

a. Ex:
```
block_name {
    statement1
    statement2
    …..
}
```

3. Executing a block

a. Again, you invoke a block by using a simple yield statement.

b. Example:
```
class Hello
    def test
        puts “you are in the method”
        yield
        puts “you are again back inside the method”
        yield
    end
    test { puts “You are in the block” }
end
```
Executing the method test above will produce the following results:
```
You are in the method
You are in the block
You are again back inside the method
You are in the block
```

4. Passing parameters to the block

a. You can also pass parameters to the block with the yield statement.

b. Ex:
```
class Hello
    def test
        yield 5
        puts “You are in the method test”
        yield 100
    end
    test { | i | puts “You are in block #{ i }” }
end
```
Executing the method test above will generate the following output:
```
You are in block 5
You are in the method test
You are in the block 100
```

c. If you want to pass more than one parameters, simply append the extra

parameters to the end of yield (i.e. yield param1, param2, … param100}

Topic 11: Embedding Ruby in HTML

1. <% Ruby Code %> The Ruby code between the delimiters is replaced with its output.

   Ex: <HTML>
   ....
   <Body>
   <% if item_quantity – item_count > 0 %>
    This item is in stock
   <% end %>
   </Body>
   </HTML>

2. <%= Ruby Expression %> The Ruby expression between the delimiters is replaced with its value.

   Ex: <HTML>
   ...
   <Body>
   The value of x is : <%= x %>
   </Body>
   <HTML>

3. <%# Ruby Code %> The Ruby code between the delimiters is commented out and ignored.

Next Tutorial: Sessions and Authentication

6.170 Software Studio
Spring 2013