Ruby-Seminar

February 6, 2017

0.1 Ruby 2.3.3

- Abishek Aditya 14BIT0137

Hello World!

```
In [3]: puts "Hello, World"
        puts  [1,2,3,4]
```

```
Hello, World
[1, 2, 3, 4]
```

Basic Maths and String intepolation

```
In [1]: puts "5+5 is #{5+5}"
        puts 5**2
        puts -7%2
        puts 71/2

        puts 71./ (2)


        puts 71/2.0
```

```
5+5 is 10
25
1
35
35
35.5
```

Some simple methods

```
In [2]: puts "Hello World".downcase
        puts "hello world".split(" ")
        puts "".empty?
        puts -57.12.abs
```

```
hello world
["hello", "world"]
true
57.12
```

Duck typing

```
In [4]: a_num = 5.2  # This is a comment
        b_char = "s"
        number = "My_number"
        puts number.class
        CONSTANT = 25
        a_num = "cake"

        puts a_num
```

```
String
cake
```

Conditionals

```
In [5]: a = "Hello" if 5 < 3
        puts a
```

```
In [6]: a = "G" unless 5 < 2
```

```
Out[6]: "G"
```

```
In [7]: if a=="G"
          puts "Yo"
        end
```

```
Yo
```

```
In [8]: unless a.empty?
          puts "value of a is #{a}"
        end
```

```
value of a is G
```

```
In [9]: a = 20
        if a > 50
          puts "Old Guy"
        elsif a > 40
          puts "Middle Aged"
        else
          puts "Not Old"
        end
```

```
Not Old
```

```
In [6]: x = 12

        case x
        when 10...12
          puts "A"
        when 13..15
          puts "C"
        else puts "AC"
        end
```

```
AC
```

loops

```
In [11]: for i in 1..5 do
           puts i
         end
```

```
1
2
3
4
5
```

```
Out[11]: 1..5
```

```
In [12]: a = 5
         while a >= 0
           puts a
           a -= 1
         end
```

```
5
4
3
```

```
2
1
0
```

```
In [13]: a = 0
         b = -5
         until a < -5 || b > 4
           puts "#{a*b}"
           a -= 1
           b += 2
         end
```

```
0
3
2
-3
-12
```

```
In [14]: [1,2,3,5].each do |i|
           puts i*i
         end
```

```
1
4
9
25
```

```
Out[14]: [1, 2, 3, 5]
```

Data Structures

```
In [7]: array = [1, 2, 3, 4, 5]


        puts [1, 'hello', false]

        puts array[0]
        puts array.first
        puts array[12]

        array.[] 0 #=> same as array[0] Array class uses indexed which is a method

        puts array[-2]
        puts array.last
```

```
[1, "hello", false]
1
1

4
5
```

```
In [8]: arr = [1,5,4,7,9,6]

        puts arr[2, 3] #start, number
        puts arr[2..5] #range
        puts arr[0...2] #range inclusive
        puts
        a=[1,2,3]
        a.reverse!
        puts a
        # Or with a range
        arr[1..3] = 50
        puts arr
        puts
        a << 4
        puts a
        a.push(5)

        puts arr.include?(50)
        puts
        a.each_with_index {|i,k|  puts "#{i**2} => #{k}"}
```

```
[4, 7, 9]
[4, 7, 9, 6]
[1, 5]

[3, 2, 1]
[1, 50, 9, 6]

[3, 2, 1, 4]
true

9 => 0
4 => 1
1 => 2
16 => 3
25 => 4
```

```
Out[8]: [3, 2, 1, 4, 5]
```

```
In [16]: # Hashes are Ruby's primary dictionary with key/value pairs.
```

```
         hash = { 'color' => 'green', 'number' => 5 }

         puts hash.keys


         puts hash['color']
         puts hash['number']

         puts hash['nothing here']


         new_hash = { words: 3, action: nil } #Symbols as keys

         puts new_hash.keys

         puts
         puts new_hash.key?(:words)
         puts new_hash.value?(13)
         puts
         new_hash.each do |key,value|
           puts "#{key} : #{value.nil? ? 'nothing exists' : value}"
         end
```

```
["color", "number"]
green
5

[:words, :action]

true
false

words : 3
action : nothing exists
```

```
Out[16]: {:words=>3, :action=>nil}
```

File Handling

```
In [4]: File.open('../planner', 'r') do |f1|
          while line = f1.gets
            puts line
          end
        end

        # Create a new file and write to it
        File.open('tester', 'w') do |f2|
          # use "\n" for two lines of text
          f2.puts "Created by Abishek\nNext Line"
        end
```

Rails - done

C# - done

C - done

C++ - done

Python - done

SQL - done

Redis - done

Java - done

ASP (MVC + Core) - done

Prolog - done

Ruby - done

Javascript - done

jQuery - done

Swift

Scala

Clojure

Rust

WPF

Elixir

FSharp

Ember

MIPS

```
In [5]: File.open('tester', 'r') do |f1|
          while line = f1.gets
            puts line
          end
        end
```

```
Created by Abishek

Next Line
```

Methods and functions

```
In [18]: def double(x)
           x * 2
         end

         # Functions (and all blocks) implicitly return the value of the last state
         double(2)

         double 3 #optional paranthesis

         double double 3

         def sum(x, y)
           x + y
         end

         sum 3, 4

         sum sum(3, 4), 5


         def surround
           puts '{'
           yield
           puts '}'
         end

         surround { puts 'hello world' }


         #you can use destructuring assignment
         def foods
             ['pancake', 'sandwich', 'quesadilla']
```

```
end

breakfast, lunch, dinner = foods
puts breakfast
puts dinner

# By convention, all methods that return booleans end with a question mar
puts 5.even?
puts 5.odd?

company_name = "Smaller Company"
puts company_name.upcase
puts company_name
puts company_name.upcase! # we're mutating
puts company_name
```

```
{
hello world
}
pancake
quesadilla
false
true
SMALLER COMPANY
Smaller Company
SMALLER COMPANY
SMALLER COMPANY
```

Classes and Modules

```
In [19]: class Human

           # A class variable
           @@species = 'H. sapiens'

           # Basic initializer
           def initialize(name, age = 0) # default age is 0, no need to add it
             @name = name
             @age = age
           end

           # Basic setter method
           def name=(name)
             @name = name
           end

           # Basic getter method
```

```
  def name
    @name
  end

 # attr_accessor :name

 # attr_reader :name
 # attr_writer :name

  # A class method uses self to distinguish from instance methods.
  def self.say(msg)
    puts msg
  end

  def species
    @@species
  end
end


abishek = Human.new("Abishek",19)
puts abishek
abishek.name = "Abishek Aditya"
puts abishek.name
puts abishek.species
puts Human.say "Hello World"  #Brackets not important
```

```
#<Human:0x005626524f6248>
Abishek Aditya
H. sapiens
Hello World
```

In [20]:
```
#Inheritance

class Human
  @@foo = 0

  def self.foo
    @@foo
  end

  def self.foo=(value)
    @@foo = value
  end

end
```

```
         # derived class
         class Worker < Human  # worker derives from human
         end

         Human.foo # 0
         Worker.foo # 0

         Human.foo = 2 # 2
         Worker.foo # 2

         # Class instance variable is not shared by the class's descendants.

         class Human
           @bar = 0

           def self.bar
             @bar
           end

           def self.bar=(value)
             @bar = value
           end
         end

         class Doctor < Human
         end

         Human.bar # 0
         Doctor.bar # nil
```

Modules are like classes but don't have initializers and can not be assigned to an object. They can be imprted and mixed in with other classes.

```
In [21]: module ModuleExample
           def foo
             'foo'
           end
         end

         # Including modules binds their methods to the class instances

         class Person
           include ModuleExample
         end

         # Extending modules binds their methods to the class itself
```

```
        class Book
          extend ModuleExample
        end

        # Person.foo      => NoMethodError: undefined method `foo' for Person:Clas
        p = Person.new
        puts p.foo     # => 'foo'


        puts Book.foo       # => 'foo'
        # Book.new.foo    => NoMethodError: undefined method `foo'
```

```
foo
foo
```

Splatting

```
In [22]: def guests(*array)   #Put host, in front
           array.each { |guest| puts guest }
         end

         guests('abishek','abhilasha','arkansas')

         def hasher(**args)
           args.each do |k,v|
             puts "#{k} is of type #{v}"
           end
         end

         hasher(cake: "chocolate",biscuit: "Orange")

         hash = {a: 5,b: 10}

         def hacker(a: 5,b: 5)
           puts a*b
         end

         hacker a: 20, b: 20
         hacker **hash
         hacker b: 500
```

```
abishek
abhilasha
arkansas
cake is of type chocolate
biscuit is of type Orange
400
```

```
50
2500
```

Blocks, Procs and Lambdas

```
In [2]: def multiplier(num,&block)
          block.call num
        end

        mul4 = Proc.new {|i| puts i*4 } #or lambda {|i| puts i*4 }

        multiplier(5,&mul4)

        multiplier(2) {|i| puts (i-1)*13}
```

```
20
13
```