Methods

Outline

1. Introduction
2. Method arguments
3. Method returning value
4. Class method and instance method
5. Block, Procs, Lambda

1. Introduction

- Ruby methods are very similar to functions in any other programming language.
- Methods have a name, take some input, do something with it, and return a result.
- Method names should begin with a lowercase letter.
- Execute method by the way call method via object.

1. Introduction(cont.)

Syntax:

```
def method_name [( [arg [= default]]...[, * arg [, &expr ]])]
   expr..
end
```

Example:

```
def print_your_name name
  puts "Your name is " + name
end
```

2. Method arguments

```ruby
def calculate_value(x,y)
  x + y
end

def calculate_value(value='default', arr=[])
  puts value
  puts arr.sum
end

def calculate_value(x,y,*otherValues)
  puts otherValues
end
```

```ruby
def accepts_hash(arguments)
  print "got: ", arguments.inspect #
will print out what it received
end

# >= Ruby 2.0
def calculate_value(a, b, c: true, d:
false)
   puts  a, b, c, d
end
```

3. Method returning value

- Methods return the value of the last statement executed
- An explicit return statement can also be used to return from function with a value, prior to the end of the function declaration

3. Method returning value (cont.)

```ruby
def calculate_value(x,y)
  x / y
end
# calculate_value(3, 4)
def second_calculate_value(x,y)
  return x / y
  # something
end
# calculate_value(3, 4)
def third_calculate_value(x,y)
  return x / y if y > 0
  0
end
# calculate_value(3, 0)
```

```ruby
def method_call
  yield
end

method_call(&someBlock)

# method_call {2*3}
```

4. Class method and instance method

```ruby
class Invoice
  # class method
  def self.print_out
    “Printed out invoice”
  end

  # instance method
  def convert_to_pdf
    “Converted to PDF”
  end
end
```

Execute method

```ruby
# class method
puts Invoice.print_out
# instance method
puts Invoice.new.convert_to_pdf
```

5. Blocks

The use of blocks is fundamental to the use of iterators

```
1.upto(10) {|x| puts x }

1.upto(10) do |x|
  puts x
end

1.upto(10)          # No block specified

{|x| puts x }    # Syntax error: block not after an invocation
```

5. Blocks (cont.)

```
array = [1, 2, 3, 4]

array.collect! do |n|
  n ** 2
end

puts array.inspect
# => [1, 4, 9, 16]
```

```
class Array
  def iterate!
    self.each_with_index do |n, i|
      self[i] = yield(n)
    end
  end
end

array = [1, 2, 3, 4]
array.iterate! do |n|
  n ** 2
end
puts array.inspect
# => [1, 4, 9, 16]
```

5. Procs

```
class Array

  def iterate!(&code)
    self.each_with_index do |n, i|
      self[i] = code.call(n)
    end
  end
end
array = [1, 2, 3, 4]
array.iterate! do |n|
  n ** 2
end
puts array.inspect
# => [1, 4, 9, 16]
```

```
# A block is just a Proc!

def what_am_i(&block)
  block.class
end
puts what_am_i {}
# => Proc
-----------------------------------
square = Proc.new do |n|
  n ** 2
end

square.call (2)

=> 4
```

5. Lambda

Lambda is an anonymous function

1. It has no name (hence anonymous)
2. Is defined inline
3. Used when you don't want the overhead/formality of a normal function
4. Is not explicitly referenced more than once, unless passed as an argument to another function

```
blah = -> {puts "lambda"}
=> #<Proc:0x000000037ea6f0@(pry):129 (lambda)>
blah.call
# lambda
# => nil
```

5. Lambda (cont.)

```
class Array
  def iterate!(code)
    self.each_with_index do |n, i|
      self[i] = code.call(n)
    end
  end
end

array = [1, 2, 3, 4]
array.iterate!(lambda {|n| n ** 2})
puts array.inspect
# => [1, 4, 9, 16]
```

5. Exercise with Block, Proc, Lambda

- Use Block to convert array of string to uper case

input: strings = [“a”, “b”, “c”]

output: up_strings = [“A”, “B”, “C”]

- Write a function that accepts a block and execute the block if the block given

5. Summary Differences

- Procs are objects, blocks are not

- At most one block can appear in an argument list

- Lambdas check the number of arguments, while procs do not

- Lambdas and procs treat the ‘return’ keyword differently

References

- http://ruby-doc.org/
- https://www.tutorialspoint.com/

Thank you for listening!