Ruby-PartII

October 21,2014

Agenda

- Blocks
- method_missing
- Modules
- Regex
- Ruby OOP
  - Inheritance
  - Access Restriction
- Mockups
- Conclusion
- Hmw

Blocks

- A block is a nameless chunk of code that lives inside a control statement, loop, method definition, or method call.
- In Ruby, blocks can be created two ways: with braces or with a do/end statement

Ex: (1..10).each {|x| print x*2," "}

Ex :
```
def block_example
      puts "Beginning of the block"
      yield
      yield
      puts "End of the block"
   end
block_scope{puts " I am in the block"}
```

Blocks(cont.)

-built in to many objects in ruby

a. each

b. detect

```
numbers = [-1,5,6,7,3,41,22]
number = numbers.detect {|x| x > 10}
print number
```

c. select

```
odds= numbers.select{|x| x % 2 == 1}
p odds
```

d. collect

```
arr = ["the","software","engineering", "Course"]
arr_up= arr.collect{|x| x.upcase}
p arr_up
```

method_missing

In most languages when a method cannot be found and error is thrown and the program crashes, but in Ruby ..

Ex: class MathTest

```
  def sum(a,b)
     return a+b
  end

  def sub(a,b)
     return a-b
  end

  def mul(a,b)
     return a*b
  end
end

mt = MathTest.new()
puts mt.sum(3,5)
puts mt.sub(3,5)
puts mt.mul(3,5)
```

method_missing(cont.)

```
def method_missing(name, *args)
  puts "I don't know the method #{name}"
end

mt = MathTest.new()
puts mt.sum(3,5)
puts mt.sub(3,5)
puts mt.mul(3,5)
puts mt.div(3,5)
```

Modules(later ..)

- basically a set of methods
- You can't instantiate a module
- You can include as many modules as you want into one class.
- **Modules are extremely popular in the Ruby community**
- One problem with modules is that in order to unit test them you either test every single method or you need to create a dummy class that includes the module and then you test an object of this class.
- Another problem is that modules introduce hidden dependencies. You can't easily create an object and replace the module with a mock in order to test the collaboration.
- The good thing about modules is that it's very easy to use. When you see a common set of methods in two classes, you extract it to a module and the duplication disappears.

(later ..)

Regular Expressions

- Regular expressions, though cryptic, is a powerful tool for working with text. Ruby has this feature built-in.
- It's used for pattern-matching and text processing.
- A regular expression is simply a way of specifying a pattern of characters to be matched in a string.
- In Ruby, you typically create a regular expression by writing a pattern between slash characters (/pattern/).

Regexes (cont.)

- objects from Regexp class
- Syntax : / pattern/
- =~
- .match method

Ex :

```
puts "Ruby: Regular Expressions" =~ /egu/
puts "Ruby: Regular Expressions".match /egu/
```

Regexes (cont.)

Mostly used :

^ → start of line

$ → end of line

| → or

d → d

\d → digit

\D → non-digit

\s → white space char.

\d+ →matches one or more numerical digits.

\[→ square bracket

\] → closing square bracket

Regexes(cont.)

`[abc]`	A single character of: a, b, or c
`[^abc]`	Any single character except: a, b, or c
`[a-z]`	Any single character in the range a-z
`[a-zA-Z]`	Any single character in the range a-z or A-Z
`^`	Start of line
`$`	End of line
`\A`	Start of string
`\z`	End of string

`(...)`	Capture everything enclosed
`(a|b)`	a or b
`a?`	Zero or one of a
`a*`	Zero or more of a
`a+`	One or more of a
`a{3}`	Exactly 3 of a
`a{3,}`	3 or more of a
`a{3,6}`	Between 3 and 6 of a

`.`	Any single character
`\s`	Any whitespace character
`\S`	Any non-whitespace character
`\d`	Any digit
`\D`	Any non-digit
`\w`	Any word character (letter, number, underscore)
`\W`	Any non-word character
`\b`	Any word boundary

Regexes (cont.)

```
/a/             character 'a'
/\//            character '/' (/\?*+{[.|()^$ need to be escaped with \)
/./             any character (including newline for /.../m)

/a?/            0..1 'a'
/a*/            0..n 'a'
/a+/            1..n 'a'
/a{2,7}/        2..7 'a'
/a{2,}/         2..n 'a'
/a{,7}/         0..7 'a'

/a?bc?/         'b' or 'ab' or 'bc' or 'abc'
/a|bc/          'a' or 'bc'
/(a|b)c/        'ac' or 'bc'

/[abc]/         a or b or c
/[^abc]/        any character except a or b or c
/[a-cF-H]/      a or b or c or F or G or H

/\d/            any digit [0-9]
/\w/            any letters, numbers or underscores [a-zA-Z0-9_]
/\s/            any whitespace character (including newline for /.../m)

/\D/            any character except digits
/\W/            any character except letters, numbers or underscores
/\S/            any character except whitespace

/^abc/          abc after line start
/abc$/          abc before line end
```

Regexes (cont.)

- ^\d+\s+ → first column of numbers

- (\d{2})$ → Any sequence of ***exactly two*** **numerical digits** at the **end of each line**

Just a Ruby Regular Expression Editor, but on the net there are many more.

http://rubular.com/

Regexes (cont.)

- Ruby String Substitution:gsub, gsub! Methods

```
my_string = "Welcome to Java!"
my_string.gsub!("Java", "Ruby")
puts my_string
```

! operator is used for in-place operations.

Regexes (cont.)

- Accessing captures

Ex : puts "a123 456 789" =~ /(\d\d)(\d)/

Ex: puts "a123 456 789" =~ /(\d\d)(\d)\s(\d)/

puts [$1, $2, $3]

$n contains the n-th (...) capture of the last match, $~ contains MatchData object

- Accessing all matches

puts "123 456 789".scan(/\d+/)

Object Oriented Programming

In Ruby, a class can only inherit from a single other class. Some other languages support multiple inheritance, a feature that allows classes to inherit features from multiple classes, but Ruby *doesn't* directly support this! (but modules!)

Ex:
```
class Person
  def initialize(name, surname)
   @name = name
   @surname = surname
  end
end

person = Person.new("Arzum"," Karatas")
print person
```

OOP- to String method

- The "ToString" method, to_s

```
class Person
  def initialize(name, surname)
      @name = name
      @surname = surname
  end
  def to_s
      "Person: #@name #@surname"
  end
end

person = Person.new("Arzum"," Karatas")
print person
```

OOP – Inheritance

```
class Employee < Person

  def initialize(name, surname, title)
    super(name,surname)
    @title= title
  end
  def to_s
    super + ", #@title"
  end
end

employee = Employee.new("Arzum", "Karatas", "TA")
print employee
```

OOP – Inheritance (cont.)

Let's try to use reach name attribute

```
print employee.name
```

- To grant access to read a variable we declare it after "attr_reader "

Ex: attr_reader :name, :surname

- To grant access to write a variable we declare it after "attr_writer"

Ex: attr_writer :title

OOP – Inheritance(cont.)

```
class Employee < Person

  def initialize(name, surname, title)
    super(name,surname)
    @title= title
  end
  def to_s
    super + ", #@title"
  end

  attr_reader :name, :surname
  attr_writer :title

end

employee = Employee.new("Arzum", "Karatas", "TA")
puts employee
puts employee.name
employee.title = "Teaching Assistant"
puts employee
```

OOP – Inheritance(cont.)

- Assume that you have a .rb file for each class. How you can handle with this situation ?

Remember from PHP ?

OOP – Inheritance(cont.)

```
require "Person"

require_relative "Person"
```

OOP – Inheritance(cont.)

Question : Create a Vehicle class, then add a class variable named "no_of_vehicles" that can keep track of the number of objects created that inherit from Vehicle.

Next, create a method to print out the value of this class variable as well.

OOP – Inheritance(cont.)

vehicle.rb

```
class Vehicle
  @@no_of_vehicles = 0

  def no_of_vehicles
    puts "This program has created #{@@no_of_vehicles}
          vehicles"
  end

  def initialize
    @@no_of_vehicles += 1
  end
end
```

OOP – Inheritance(cont.)

- car.rb

```
require_relative "Vehicle"
class Car < Vehicle

end
```

- bike.rb

```
require_relative "Vehicle"
class Bike < Vehicle

end
```

OOP – Inheritance(cont.)

- **test.rb**

```
require_relative "Car"
require_relative "Bike"

my_car = Car.new()
my_bike = Bike.new()

puts my_car.no_of_vehicles
puts my_bike.no_of_vehicles
```

OOP –Access Modifiers

- **public, private, protected**

Question: Create a class titled as ‘Person’ with attributes name and age. Do NOT make the age getter public, so jack.age will raise an error. Create a older_than? method, that you can call like in the following.

```
puts "Jack is older than Sally !" if  jack.older_than?(sally)
```

OOP –Access Modifiers(cont.)

```
class Person
  def initialize(name, age)
    @name = name
    @age = age
  end

  def older_than?(other_person)
    age > other_person.age
  end

  protected

  def age
    @age
  end
end

jack = Person.new("Jack", 43)
sally = Person.new("Sally", 24)
puts "Jack is older than Sally !" if jack.older_than?(sally)
```

OOP-Method Overriding

```
class Animal
  def move
    "I can move"
  end
end

class Bird < Animal
  def move
    super + " by flying"
  end
end

class Fish < Animal
  def move
    super + " by swimming"
  end
end
```

OOP-Method Overriding(cont.)

```
class Snake < Animal
  def move
    super + " by slithering"
  end
end

twitty = Bird.new()
twitty.move
fishy = Fish.new()
fishy.move
dui = Snake.new()
dui.move
```

OOP –Private Methods

```
class Animal

   def move
     "I can move"
   end

   def secret
     puts "this method is private"
   end
   private :secret
end

class Snake < Animal
  def move
    puts super + " by slithering"
  end
end

sammy= Snake.new()
sammy.move
sammy.secret
```

Conclusion

- OOP in Ruby
- Blocks
- method_missing
- Regexes

Mock-ups

Last year's mockup examples

https://sites.google.com/site/amsteamproject/file-cabinet

http://software-engineering4.webnode.com.tr/projenin-isleyisi/

http://fibilgisayar.weebly.com/proje-304351leyi351i.html

Hmw

- You will have an homework!

Due Date : November 02, Monday at 8 pm

Acknowledgements

This slides is collected from many sources.
Thanks to all of their authors.