UW Ruby Programming 110

Winter 2015

Michael Cohen

Lecture 6

Feb 12, 2015

Lecture 6

1. Regular Expressions
2. Blocks, Procs and Lambdas
3. DSLs
4. Assignments

Section 1
Regular Expressions

Definition

A regular expression is a pattern matching specification

Patterns

- syntax inherited from Perl
- same syntax used by almost every programming language

Examples

```
abc          #=> match 'abc'
ab(c|d)      #=> match 'abc' or 'abd'
ab+c         #=> match 'abc', 'abbc', 'abbbc', etc.
ab*c         #=> match 'ac', 'abc', 'abbc', 'abbbc', etc.
[a-z]        #=> match any single char between a and z
[^abc]       #=> match any single char other than a, b, c
```

Abbreviations

```
# \d   [0-9]                Digit character
# \D   [^0-9]               Nondigit
# \s   [\s\t\r\n\f]         Whitespace character
# \S   [^\s\t\r\n\f]        Nonwhitespace character
# \w   [A-Za-z0-9_]         Word character
# \W   [^A-Za-z0-9_]        Nonword character
```

Creating a Regular Expression

```
a = Regexp.new('^\s*[a-z]')   #=> /^\s*[a-z]/

b = /^\s*[a-z]/               #=> /^\s*[a-z]/

c = %r{^\s*[a-z]}             #=> /^\s*[a-z]/
```

Matching against a RegExp

```
"Hello World" =~ /ello/          #=> 1
"Hello World" =~ /ll/            #=> 2
"Hello World" =~ /World/         #=> 6

"Hello World" =~ /elo/           #=> nil
```

Section 2
Blocks, Procs and Lambdas

Blocks

```
[1,2,3].map do |elem|
  elem*elem
end
```

Blocks

```
def render_html
  <<-HTML
    <html>
      <head>...</head>
      #{yield if block_given?}
    </html>
  HTML
end
```

Blocks

```
def render_html(&render_body)
  <<-HTML
    <html>
      <head>...</head>
      #{render_body.call if render_body}
    </html>
  HTML
end
```

Blocks

What if you want to accept multiple blocks?

```
def render_html(&render_head, &render_body)
  <<-HTML
    <html>
      #{render_head.call if render_head}
      #{render_body.call if render_body}
    </html>
  HTML
end
```

Blocks

How do you call a method that accepts multiple blocks?
What is a block?

```
def discover_block(&block)
  puts block.class
end
```

Procs

What is a Proc?

```
greeter = Proc.new do |name|
  puts "Hello #{name}"
end

greeter.call "Matz"
```

Shortcuts

```
greeter = proc do |name|
  puts "Hello #{name}"
end

greeter.("Matz")
greeter["Matz"]
```

Lambda

```
greeter_l = lambda do |name|
  puts "Hello #{name}"
end

greeter_l.call "Matz"
```

"Stabby" Lambda

```
# concise:
greeter_l = ->(name){ puts "Hello #{name}" }

greeter_l.call "Matz"
```

Proc vs Lambda

Lambdas check the number of args.
Procs don't.

```
greeter_p = proc {|name| puts "Hello #{name}"}
greeter_p.call  #=> "Hello "

greeter_l = lambda {|name| puts "Hello #{name}"}
greeter_l.call  #=> ArgumentError: wrong number of arguments
```

Proc vs Lambda

Proc and Lambda differ in how they handle return.

Proc vs Lambda

```ruby
def lambda_return
  l = lambda do
    puts "in lambda"
    return
    puts "in lambda after return"
  end
  puts "before lambda"
  l.call
  puts "after lambda"
end
lambda_return
#=> before lambda
#=> in lambda
#=> after lambda
```

Proc vs Lambda

```
def proc_return
  p = proc do
    puts "in proc"
    return
    puts "in proc after return"
  end
  puts "before proc"
  p.call
  puts "after proc"
end
proc_return
#=> before proc
#=> in proc
#
```

Closures

```
def create_greeter(name)
  ->(){ puts name }
end

greeter = create_greeter "Jane Smith"
greeter.call      #=> "Jane Smith"
```

Closures

```
def create_counter
  value = 0
  ->(){ value += 1 }
end

counter = create_counter
counter.call    #=> 1
counter.call    #=> 2
```

Section 2: Blocks, Procs and Lambdas

Closures

```
def create_counter
  value = 0
  inc_lambda = ->(){ value += 1 }
  dec_lambda = ->(){ value -= 1 }
  value_lambda = ->(){ value }
  {inc: inc_lambda, dec: dec_lambda, value: value_lambda}
end

counters = create_counter
counters[:inc].call   #=> 1
counters[:value].call #=> 1
counters[:inc].call   #=> 2
counters[:dec].call   #=> 1
```

Section 3

DSLs

Section 3: DSLs

Definition

DSL = Domain Specific Language

Section 3: DSLs

Examples

Rake

```
task :brush_teeth do
  puts "brushing teeth"
end

task :eat_breakfast do
  puts "eating breakfast"
end

task :go_to_bed => [:brush_teeth] do
  puts "going to bed"
end

task :go_to_school => [:brush_teeth, :eat_breakfast] do
  puts "going to school"
end
```

Section 3: DSLs

Examples

RSpec

```
describe Book do
  before :each do
    @book = Book.new "Title", "Author"
  end
  describe "#title" do
    it "returns the correct title" do
      @book.title.should eql "Title"
    end
  end
  describe "#author" do
    it "returns the correct author" do
      @book.author.should eql "Author"
    end
  end
end
```

Section 3: DSLs

Examples

Markaby

```
html do
  head do
    title 'my page'
  end

  body do
    h1 'my page'
    p  'lorem ipsum'
  end
end
```

Section 3: DSLs

Recipe DSL

Let's create a simple DSL for recipes

Section 3: DSLs

Recipe DSL

What does a recipe look like?

```
recipe "Scrambled Eggs"
  ingredients:
    - 2 eggs
    - 1/4 cup of milk
    - 1 tbsp butter
    - dash of salt
    - dash of pepper
  steps:
    1. crack eggs into medium mixing bowl
    2. whisk eggs
    3. add milk
    4. add salt & pepper to taste
    5. heat pan to medium high heat
    6. melt butter in pan
    7. once hot, add eggs to pan
```

Section 3: DSLs

Recipe DSL

How do we create a DSL?

Two questions:
1. what syntax do we want?
2. what objects do we need?

Recipe DSL

Let's define our syntax:

```
recipe "Scrambled Eggs" do
end
```

Recipe DSL

```
recipe "Scrambled Eggs" do
  ingredients do
    ...
  end
  steps do
    ...
  end
end
```

Section 3: DSLs

Recipe DSL

Specify ingredients:

```
recipe "Scrambled Eggs" do
  ingredients do
    - 2 eggs
    - 1/4 cup of milk
    - 1 tbsp butter
    - dash of salt
    - dash of pepper
  end
  steps do ... end
end
```

Recipe DSL

Convert ingredients to ruby syntax:

```
recipe "Scrambled Eggs" do
  ingredients  [
    "2 eggs",
    "1/4 cup of milk",
    "1 tbsp butter",
    "dash of salt",
    "dash of pepper"
  ]
  steps do ... end
end
```

Section 3: DSLs

Recipe DSL

Specify steps:

```
recipe "Scrambled Eggs" do
  ingredients do ... end
  steps do
    1. crack eggs into medium mixing bowl
    2. whisk eggs
    3. add milk
    4. add salt & pepper to taste
    5. heat pan to medium high heat
    6. melt butter in pan
    7. once hot, add eggs to pan
  end
end
```

Section 3: DSLs

Recipe DSL

Convert steps to ruby syntax:

```
recipe "Scrambled Eggs" do
  ingredients do ... end
  steps [
    "crack eggs into medium mixing bowl",
    "whisk eggs",
    "add milk",
    "add salt & pepper to taste",
    "heat pan to medium high heat",
    "melt butter in pan",
    "once hot, add eggs to pan"
  ]
end
```

Section 3: DSLs

Recipe DSL

```
class RecipeBuilder
  def recipe(name)
    @recipe = Recipe.new name
    yield
    @recipe
  end

  def ingredients(is)
    @recipe.ingredients = is
  end

  def steps(ss)
    @recipe.steps = ss
  end
end
```

Recipe DSL

```
class Recipe
  attr_accessor :ingredients, :steps, :name
  def initialize(name)
    @name = name
  end
end
```

Section 3: DSLs

Recipe DSL

```
rb = RecipeBuilder.new

rb.recipe "Scrambled Eggs" do
  rb.ingredients [
    "2 eggs",
    "1/4 cup of milk",
    "1 tbsp butter",
    "dash of salt",
    "dash of pepper"
  ]
  rb.steps [
    "crack eggs into medium mixing bowl",
    "whisk eggs",
    "add milk",
    "add salt & pepper to taste",
    "heat pan to medium high heat",
    "melt butter in pan",
    "once hot, add eggs to pan"
  ]
end
```

Section 3: DSLs

Recipe DSL

`instance_eval`: some magic to improve the syntax

```
class RecipeBuilder
  def recipe(name, &block)
    @recipe = Recipe.new name
    self.instance_eval &block
    @recipe
  end

  def ingredients(is)
    @recipe.ingredients = is
  end

  def steps(ss)
    @recipe.steps = ss
  end
end
```

Section 3: DSLs

Recipe DSL

```
rb = RecipeBuilder.new

rb.recipe "Scrambled Eggs" do
  ingredients [
    "2 eggs",
    "1/4 cup of milk",
    "1 tbsp butter",
    "dash of salt",
    "dash of pepper"
  ]
  steps [
    "crack eggs into medium mixing bowl",
    "whisk eggs",
    "add milk",
    "add salt & pepper to taste",
    "heat pan to medium high heat",
    "melt butter in pan",
    "once hot, add eggs to pan"
  ]
end
```

Section 3: DSLs

Recipe DSL

```
rb.recipe "Scrambled Eggs" do
  ingredients do
    x "2 eggs"
    x "1/4 cup of milk"
    x "1 tbsp butter"
    x "dash of salt"
    x "dash of pepper"
  end
  steps do
    x "crack eggs into medium mixing bowl"
    x "whisk eggs"
    x "add milk"
    x "add salt & pepper to taste"
    x "heat pan to medium high heat"
    x "melt butter in pan"
    x "once hot, add eggs to pan"
  end
end
```

Section 3: DSLs

Recipe DSL

```
class RecipeBuilder
  def recipe(name, &block)
    @recipe = Recipe.new name
    self.instance_eval &block
    @recipe
  end

  def ingredients(&block)
    @recipe.ingredients = []
    @receiver = @recipe.ingredients
    self.instance_eval &block
  end

  def steps(&block)
    @recipe.steps = []
    @receiver = @recipe.steps
    self.instance_eval &block
  end

  def x(value)
    @receiver << value
  end
end
```

Recipe DSL

```
def recipe(name, &block)
  rb = RecipeBuilder.new
  rb.recipe(name, &block)
end
```

Section 3: DSLs

Recipe DSL

```
recipe "Scrambled Eggs" do
  ingredients do
    x "2 eggs"
    x "1/4 cup of milk"
    x "1 tbsp butter"
    x "dash of salt"
    x "dash of pepper"
  end
  steps do
    x "crack eggs into medium mixing bowl"
    x "whisk eggs"
    x "add milk"
    x "add salt & pepper to taste"
    x "heat pan to medium high heat"
    x "melt butter in pan"
    x "once hot, add eggs to pan"
  end
end
```

Recipe DSL

Add additional properties:

```
recipe "Scrambled Eggs" do
  category "breakfast"
  prep_time "10 mins"
  rating 4
  ingredients do ... end
  steps do ... end
end
```

Recipe DSL

Add attributes to Recipe class:

```
class Recipe
  attr_accessor :steps, :ingredients, :name, :category, :prep_time, :rating
  def initialize(name)
    @name = name
  end
end
```

Recipe DSL

Add method to builder:

```
class RecipeBuilder
  def category(value)
    @recipe.category = value
  end

  def prep_time(value)
    @recipe.prep_time = value
  end

  def rating(value)
    @recipe.rating = value
  end
end
```

Section 4

Assignment #6

Section 4: Assignment #6

Problem 1: PriorityQueue

```
# implement a PriorityQueue
# validate using MiniTest unit tests

# expected results:
pq = PriorityQueue.new
pq.empty?       #=> true

pq.enqueue "first"
pq.empty?       #=> false

pq.enqueue "top", :high
pq.enqueue "last", :low
pq.enqueue "second"
pq.enqueue "another top", :high

pq.dequeue     #=> "top"
pq.dequeue     #=> "another top"
pq.dequeue     #=> "first"
pq.dequeue     #=> "second"
pq.dequeue     #=> "last"
```

Section 4: Assignment #6

Problem 2: Recipe to DSL

```
# render a Recipe object to Recipe DSL

# given Recipe class we defined above
# add render_dsl method which generates the DSL
```