Developing Web Apps with Ruby and Rails

Prof. Paul Krause, University of Surrey

Chapter 1: Lecture 3

A Little Deeper into Ruby

Objectives for today

- Explore some core Ruby syntax
  - You should take time to experiment a little with what you learn
- Introduce Classes in Ruby
- If you don’t have Ruby installed yet, then take a look at Chapter 1 Lecture 2

Source for today’s material

- The “Pickaxe book”:
    - Programming Ruby 1.9 - The Pragmatic Programmer’s Guide
    - Dave Thomas
    - Pragmatic Bookshelf

Arrays and Hashes

```
a = [1, ‘cat’, 3.14]    # array with 3 elements

puts “The first element is #{a[0]}”

# set the third element

a[2] = nil

puts “The array is now #{a.inspect}”
```

Note that *nil* is an object – it just represents nothing

Short cut to arrays with words

a = [‘ant’, ‘bee’, ‘cat’, ‘dog’, ‘fox’]

Try a[0], a[1] in the irb

Alternative is

a = %w{ ant bee cat dog fox }

Try the above again.

Hashes

- Basically a list of key, value pairs separated by “=>”
- Each Key in a particular Hash must be unique

```
inst_section = {
   ‘cello’ => ‘string’,
   ‘clarinet’ => ‘woodwind’,
   ‘drum’ => ‘percussion’,
   ‘oboe’ => ‘woodwind’,
   ‘trumpet’ => ‘brass’,
   ‘violin’ => ‘string’
}
```

- Try accessing with `p inst_section[‘KEY’]`
  - (what happens if you use a key that is not yet defined?)

Control Structures

```
if count > 10
  puts “Try Again”
elseif tries == 3
  puts “You loose”
else
  puts “Enter a number”
end


while weight < 100 and num_pallets <= 30
  pallet = next_pallet()
  weight += pallet.weight
  num_pallets += 1
end
```

Statements as conditions

`gets` returns `nil` when the end of file is reached, and

`nil` is treated as “false” in conditions, so

```
while line = gets
  puts line.downcase
end
```

will terminate cleanly when the end of file is reached.

Statement modifiers

- Useful if the body of an if or while statement is just a single expression

```
if radiation > 1000
  puts “I suggest you leave now!”
end
```

- Can be rewritten as

```
puts “I suggest you leave now!” if radiation > 1000
```

- Also

```
square = 2
square = square*square while square < 1000
```

Regular Expressions

- To match a string containing either Perl or Python use:

```
/Perl|Python/ or
/P(erl|ython)/
```

- Repetition – one a, followed by one or more b’s and finish with one c:

```
/ab+c/
```

- For zero or more b’s use “*”:

```
/ab*c/
```

- Character classes

`\s` – matches any white space character
\w – matches characters that may appear in words [A-Z,a-z,0-9]
\d – matches any digit
. – matches (almost) any character

Using Regular Expressions

```
if line =~ /Perl|Python/
  puts “Scripting language mentioned: #{line}”
end
```

- Changing history:

```
line.sub(/Perl/, ‘Ruby’)      # Replace first ‘Perl’ with ‘Ruby’
line.gsub(/Python/, ‘Ruby’) # Replace every ‘Python’ with
                             # ‘Ruby’

line.gsub(/Perl|Python/, ‘Ruby’) # Total dominance
```

Blocks and iterators

- Two kinds of delimiter for code blocks

```
{ puts “Hello” }
```

- Or

```
do
  club.enroll(person)
  person.socialize
 end
```

Yield

- What can you do with a block?
- You can associate it with a call to a method

```
greet { puts “Hi” }
```

- The method (‘greet’ in the above case) can then invoke the block using the Ruby `yield` statement
- Try it out...

Blocks and yield

- Inter this into a Ruby file:

```
def call_block
  puts "Start of Block"
  yield
  yield
  puts "End of method"
end

call_block { puts "In the block" }
```

Passing arguments into a block

```
def who_says_what
  yield("Dave", "hello")
  yield("Simon", "goodbye")
end

who_says_what {|person, phrase| puts "#{person} says
                                               #{phrase}"}
```

Using blocks to implement iterators

- You will see this used widely in Ruby and in Rails
- Iterators return successive elements from some kind of collection. E.g.:

```
animals = %w( ant bee cat dog fox )
animals.each {|animal| puts animal}
```

- You might remember this example from the last lecture:

```
3.times {puts “Hello World!”}
```

Writing

- Ruby supports formatted writing in much the same way as C, Java and PERL
- Use `printf` as illustrated below:

```
printf(“Number: %5.2f, \nString: %s\n”, 1.23, “hello”)
```

Classes, Objects and Variables

- We will use a simple example to base this discussion around
  - Following the “Pickaxe Book”
- We want to monitor stock in a bookshop:
  - Scan books to record: Date; ISBN No.; Price
  - Enter each record into a file
  - Analyse the data to find out how many copies of each book we have, and what is the total value of the stock

Class BookInStock

- Create a new (Ruby Project) folder
- Call it BookShop, or something similar
- Create a new Ruby Class - create a new file and call it book_in_stock.rb
- Enter the following skeleton:

```
class BookInStock
  def initialize

  end
end
```

Adding State

- We need to add in instance variables so that objects of class BookInStock actually contain the information we need:

```
class BookInStock
   def initialize(isbn, price)
       @isbn=isbn
       @price=Float(price)
   end
end
```

Adding State

- We need to add in instance vari of class BookInStock actually contain the information

local variables

```
class BookInStock
    def initialize(isbn, price)
        @isbn=isbn
        @price=Float(price)
    end
end
```

Adding State

- We need to add in instance vari of class BookInStock actually contain the information

local variables

```
class BookInStock
    def initialize(isbn, price)
        @isbn=isbn
        @price=Float(price)
    end
end
```

instance variables

Print out some objects

```
class BookInStock
   def initialize(isbn, price)
        @isbn=isbn
        @price=Float(price)
   end
end
b1 = BookInStock.new("isbn1", 3)
p b1
b2 = BookInStock.new("isbn2", 3.14)
p b2
b1 = BookInStock.new("isbn3", “5.67”)
p b3
```

Creating a string representation

```
class BookInStock
  def initialize(isbn, price)
    @isbn=isbn
    @price=Float(price)
  end
  def to_s
    "ISBN: #{@isbn}, price: #{@price}"
  end
end

b1 = BookInStock.new("isbn1", 3)
puts b1
b2 = BookInStock.new("isbn2", 3.14)
puts b2
```

Recap and Next

- We have delved a little deeper into Ruby basics
- Next time we will
  - explore classes in a little more detail
  - start to explore Rails