UW Ruby Programming 110

Winter 2015

Michael Cohen

Lecture 4

Jan 29, 2015

Lecture 4

1. Assignment Observations
2. Review: Blocks
3. Building Data Structures
4. Nested Methods
5. Recursions
6. Module
7. Code beyond your file
8. Additional Resources
9. Assignment #4

Section 1
Assignment Observations

Section 1: Assignment Observations

if is an expression

```
# don't do this:
if x == y
  value = some_expression
else
  value = some_other_expression
end
```

Section 1: Assignment Observations

if is an expression

```
# do this instead:
value = if x == y
  some_expression
else
  some_other_expression
end
```

Section 1: Assignment Observations

if as return value

```
# don't do this:
def some_method
  # ...
  if x == y
    value = some_expression
  else
    value = some_other_expression
  end
  return value
end
```

Section 1: Assignment Observations

if as return value

```
# don't do this either:
def some_method
  # ...
  value = if x == y
    some_expression
  else
    some_other_expression
  end
  return value
end
```

Section 1: Assignment Observations

if as return value

```
# do this:
def some_method
  # ...
  if x == y
    some_expression
  else
    some_other_expression
  end
end
```

Section 1: Assignment Observations

if then true else false

```
# don't do this:
value = if x == y
  true
else
  false
end
```

Section 1: Assignment Observations

if then true else false

```
# do this instead:
value = x == y
```

Section 1: Assignment Observations

if then true else false

```
# don't do this:
def some_method
  # ...
  value = if x == y
    true
  else
    false
  end
  return value
end
```

Section 1: Assignment Observations

if then true else false

```
# don't do this either:
def some_method
  # ...
  if x == y
    true
  else
    false
  end
end
```

Section 1: Assignment Observations

if then true else false

```
# do this instead:
def some_method
  # ...
  x == y
end
```

Section 1: Assignment Observations

puts vs return value

```
# don't do this:
def some_method
  # ...
  puts result
end
```

Section 1: Assignment Observations

puts vs return value

```
# do this:
def some_method
  # ...
  result
end
```

Section 1: Assignment Observations

puts vs return value

```ruby
# this is OK for debugging:
def some_method
  # ...
  puts result
  result
end
```

Section 1: Assignment Observations

each vs map

```
# don't do this:
new_ary = []
ary.each do |item|
  new_ary << some_method(item)
end
```

Section 1: Assignment Observations

each vs map

```
# do this:
new_ary = ary.map do |item|
  some_method(item)
end
```

Section 1: Assignment Observations

each vs map

```ruby
# don't do this:
def sample_method(ary)
  # ...
  new_ary = []
  ary.each do |item|
    new_ary << some_method(item)
  end
  new_ary
end
```

Section 1: Assignment Observations

each vs map

```
# don't do this either:
def sample_method(ary)
  # ...
  new_ary = ary.map do |item|
    some_method(item)
  end
  new_ary
end
```

Section 1: Assignment Observations

each vs map

```
# do this:
def sample_method(ary)
  # ...
  ary.map do |item|
    some_method(item)
  end
end
```

Section 1: Assignment Observations

each vs map

```
# when do you use each?

# 1. when you only care about
#    the side-effect (such as I/O)

# 2. each_with_index
```

Section 1: Assignment Observations

side-effects

```
# what is a side-effect?

# when a method does something
# other than return the result

# Examples:
# 1. I/O, such as puts
# 2. modify input args
```

Section 1: Assignment Observations

side-effects

```
# Avoid side-effects

# => don't do any I/O
# => don't modify input args
```

Section 2

Review: Blocks

What is a block?

1. a callback
2. mechanism to inject code
3. mechanism to provide specialized behavior

Using Blocks

```
# iteration:
[1,2,3].each {|item| puts item}

# transactions:
File("name.txt") do |file|
  lines = file.readlines
end
```

Section 2: Blocks

Using Blocks

```
# customization:
render_body "Bank Statement" do
  render_records
end

render_body "News Article" do
  render_article
end
```

Section 2: Blocks

Implementing Methods that take blocks

```
def render_body(title)
  <<-BODY
    <header>...</header>
    <nav>...</nav>
    <h1>#{title}</h1>
    <main>#{yield}</main>
    <footer>...</footer>
  BODY
end
```

Section 3

Nested Methods

Section 3: Nested Methods

Method

```ruby
def render_html(title)
  # ...
end
```

Section 3: Nested Methods

Nested Method

```
def render_html

  def render_body
    #...
  end

  # ...
  render_body
  # ...
end
```

Section 4

Recursion

Section 4: Recursion

What is recursion?

A method that calls itself

An elegant why to leverage a divide & conquer strategy to solving problems

Section 4: Recursion

Factorial

```
4! = 4 x 3 x 2 x 1 = 24

0! = 1
1! = 1

n! = (n-1)! x n
```

Section 4: Recursion

Factorial - Iterative Solution

```ruby
def factorial(n)
  acc = 1
  while n > 1
    acc *= n
    n -= 1
  end
  acc
end
```

Section 4: Recursion

Factorial - Recursive

```ruby
def factorial(n)
  if n < 2
    1
  else
    factorial(n-1) * n
  end
end
```

Section 4: Recursion

Factorial - Recursive

```ruby
def factorial(n)
  (n < 2) ? 1 : factorial(n-1) * n
end
```

Section 4: Recursion

Towers of Hanoi

```
def move(num_disks, start=:L, target=:M, using=:R)
  if num_disks == 1
    @towers[target] << @towers[start].pop
    puts "Move disk from #{start} to #{target} : #{@towers}"
  else
    move(num_disks-1, start, using, target)
    move(1,           start, target, using)
    move(num_disks-1, using, target, start)
  end
end

@towers = {
  L: [3, 2, 1],
  M: [],
  R: []
}
move(3)
```

Section 5
Building Data Structures

Section 5: Building Data Structures
Stack

```
s = Stack.new
s.empty?               #=> true
s.push "first"
s.empty?               #=> false
s.peek                 #=> "first"
s.push "second"
s.length               #=> 2
s.pop                  #=> "second"
s.pop                  #=> "first"
s.pop                  #=> nil
```

Section 5: Building Data Structures

Stack

```ruby
class Stack
  def initialize
    @items = []
  end
  def push(item)
    @items << item
    self
  end
  def pop
    @items.pop
  end
  def empty?
    @items.empty?
  end
  def peek
    @items.last
  end
  def length
    @items.length
  end
end
```

Section 5: Building Data Structures

Stack - without Array

```
class Stack
  class Node
    attr :item, :link
    def initialize(item, link)
      @item = item
      @link = link
    end
  end

  def initialize
    @nodes = nil
  end
  def empty?
    @nodes.nil?
  end
  def push(item)
    @nodes = Node.new item, @nodes
    self
  end
```

Section 5: Building Data Structures

Stack - without Array

```
  def pop
    node = @nodes
    @nodes = node.nil? ? nil : node.link
    node.nil? ? nil : node.item
  end
  def peek
    @nodes.nil? ? nil : @nodes.item
  end
  def length
    count = 0
    node = @nodes
    while node
      count += 1
      node = node.link
    end
    count
  end
end
```

Section 5: Building Data Structures

Queue

```ruby
q = Queue.new
q.empty?                #=> true
q.enqueue "first"
q.empty?                #=> false
q.length                #=> 1
q.enqueue "second"
q.length                #=> 2
q.dequeue               #=> "first"
q.dequeue               #=> "second"
q.dequeue               #=> nil
```

Section 5: Building Data Structures

Queue

```ruby
class Queue
  def initialize
    @items = []
  end
  def enqueue(item)
    @items << item
    self
  end
  def dequeue
    @items.shift
  end
  def empty?
    @items.empty?
  end
  def peek
    @items.first
  end
  def length
    @items.length
  end
end
```

Section 5: Building Data Structures

Set

```
s = Set.new
s.empty?          #=> true
s << 1
s.empty?          #=> false
s.length          #=> 1
s << 2
s.length          #=> 2
s << 2
s.length          #=> 2
s.include? 3      #=> false
```

Section 5: Building Data Structures

Set

```ruby
class Set
  def initialize
    @items = []
  end
  def length
    @items.length
  end
  def <<(item)
    @items << item
    @items.uniq!
    self
  end
  def empty?
    @items.empty?
  end
  def include?(item)
    @items.include? item
  end
  def each(&block)
    @items.each(&block)
  end
end
```

Section 6

Modules

Section 6: Modules

What is a Module?

1. Namespace
2. Mixin

Section 6: Modules

Namespace

```ruby
module UwRuby110
  class BankStatement
    # ...
  end
end

statement = UwRuby110::BankStatement.new
```

Nested Namespace

```ruby
module UwRuby110
  module Assignment03
    class BankStatement
      # ...
    end
  end
end

statement = UwRuby110::Assignment03::BankStatement.new
```

Section 6: Modules

Nested Namespace

```ruby
module UwRuby110::Assignment03
  class BankStatement
    # ...
  end
end

statement = UwRuby110::Assignment03::BankStatement.new
```

Section 6: Modules

Mixins

```ruby
module Motorized
  def motor=(new_motor)
    @motor = new_motor
  end
  def motor
    @motor
  end
  def motorized?
    @motor != nil
  end
end
```

Section 6: Modules

Mixins

```
class Vehicle
  # ...
end

class MotorizedVehicle < Vehicle
  include Motorized
end

car = MotorizedVehicle.new
car.motorized?          #=> false
car.motor = "Hemi"
car.motorized?          #=> true
```

Section 7
Code beyond your file

Section 7: Code beyond your file

What is require?

`require` **is how you load code from other files**

Section 7: Code beyond your file

Example

```ruby
require "motorized"

class MotorizedVehicle < Vehicle
  include Motorized
end
```

What is RubyGems?

RubyGems is a package manager for Ruby:
- standard format for distributing programs & libs:
  - self-contained format called a "gem"
- tool to easily manage installation of gems
- server for distributing them

Section 7: Code beyond your file

Installing Gems

```
# show installed gems:
gem list

# install a gem:
gem install <gem-name>
```

Ruby Version Manager

Ruby Version Manager - rvm
https://rvm.io/

A command-line tool which allows you to easily install, manage, and work with multiple ruby environments from interpreters to sets of gems.

Section 7: Code beyond your file

Bundler

http://bundler.io/

Bundler provides a consistent environment for Ruby projects by tracking and installing the exact gems and versions that are needed.

Section 7: Code beyond your file

Bundler

sample gemfile:

```
source 'https://rubygems.org'
gem 'nokogiri'
gem 'rack', '~>1.1'
gem 'rspec', :require => 'spec'
```

Bundler

Bundler commands:

```
bundle install
bundle update
bundle exec <some-program>
```

Section 8
Additional Resources

Section 8: Additional Resources

1. http://ruby-doc.com/docs/ProgrammingRuby/
2. http://www.gotealeaf.com/books/ruby/read/introduction
3. http://rubymonk.com/learning/books/1-ruby-primer
4. http://learnrubythehardway.org/book/

Section 9

Assignment #4

Section 8: Assignment #4

Problem 1 - Fibonacci

```ruby
# 1, 1, 2, 3, 5, 8, 13, 21, ...

# F[0] -> 1
# F[1] -> 1
# F[n] -> F[n-2] + F[n-1]

def fib(n)
  # your implementation here
end

# expected behavior:
fib(0)     #=> 1
fib(1)     #=> 1
fib(5)     #=> 8
fib(4)     #=> 5
fib(12)    #=> 233
```

Section 8: Assignment #4

Problem 2 - Queue

```
# implement a Queue class that does not use Array.

# expected behavior:
q = Queue.new
q.empty?                  #=> true
q.enqueue "first"
q.empty?                  #=> false
q.enqueue "second"
q.dequeue                 #=> "first"
q.dequeue                 #=> "second"
q.dequeue                 #=> nil
```

Section 8: Assignment #4

Problem 2 - Queue

```ruby
class Queue
  def initialize
    # your implementation here
  end
  def enqueue(item)
    # your implementation here
  end
  def dequeue
    # your implementation here
  end
  def empty?
    # your implementation here
  end
  def peek
    # your implementation here
  end
  def length
    # your implementation here
  end
end
```

Section 8: Assignment #4

Problem 3 - LinkedList

```
# implement a LinkedList class that does not use Array.

# expected behavior:
ll = LinkedList.new
ll.empty?               #=> true

ll << "first"
ll.empty?               #=> false
ll.length               #=> 1
ll.first                #=> "first"
ll.last                 #=> "first"

ll << "second"
ll.length               #=> 2
ll.first                #=> "first"
ll.last                 #=> "second"

ll << "third"           #=> 3
ll.each {|x| puts x} #=> prints out "first", "second", "third"

ll.delete "second"      #=> "second"
ll.length               #=> 2
ll.each {|x| puts x} #=> prints out "first", "third"
```

Section 8: Assignment #4

Problem 3 - LinkedList

```ruby
class LinkedList
  def initialize
    # your implementation here
  end
  def empty?
    # your implementation here
  end
  def length
    # your implementation here
  end
  def <<(item)
    # your implementation here
  end
  def first
    # your implementation here
  end
  def last
    # your implementation here
  end
  def each(&block)
    # your implementation here
  end
end
```