UW Ruby Programming 110
Winter 2015
Michael Cohen

Lecture 2
Jan 16, 2015

Administrivia

- Assignment #1
- Lecture 1 recordings
- Dash
- Slack

Assignment #1

Assignment #1

Problem 1

```
def titleize(s)
  words = s.split
  caps = []
  words.each do |word|
    caps << word.capitalize
  end
  caps.join " "
end
```

Assignment #1

Problem 1 - alternate

```ruby
def titleize(s)
  s.split.map {|word| word.capitalize}.join " "
end
```

Assignment #1

Problem 2

```ruby
def my_reverse(s)
  output = ""
  letters = s.split ""
  n = letters.length
  while n > 0
    n = n - 1
    output << letters[n]
  end
  output
end
```

Assignment #1

Problem 3

```ruby
def palindrome?(s)
  stripped = s.delete(" ").delete(",").downcase
  stripped == stripped.reverse
end
```

Lecture 2

1. Review - String
2. Hash
3. Array
4. Basic I/O
5. Putting it all together
6. Rendering HTML
7. Assignments

Section 1
Review - String

Section 1: Review - String

Syntax

```ruby
'this is a string'      # single quote
"another string"        # double-quote

name = "Michael"
"my name is #{name}"    # interpolation

# heredocs:
long_string = <<END
  all the news
  that's fit to print
END
```

Section 1: Review - String

Methods

```
# length, slice (aka []), index
# empty?, include?, start_with?, end_with?
# upcase, downcase, swapcae, capitalize
# chop, chomp, delete
# split
# each_char, each_line
```

Section 2

Hash

Section 2: Hash

Creating a Hash

```
h1 = Hash.new                # => {}
h2 = {}
h2[:random_key]              # => nil

h3 = Hash.new("Go fish")     # => sets a default param
h3[:random_key]              # => "Go fish"

# hash literal syntax:
h4 = {:first => "John", :last => "Doe"}  # classic hash-rocket
h5 = {first: "Jane", last: "Smith"}      # JSON syntax
```

Section 2: Hash

Methods - Basics

```
h = {first: "Jane", last: "Smith"}

# fetch
# provide default value if key is missing
h[:middle]                  # => nil
h.fetch(:middle, "")        # => ""
h.fetch(:first, "")         # => "Jane"
```

Section 2: Hash

Methods - Basics

```
h = {first: "Jane", last: "Smith"}

# delete
# returns value for key, removes key from hash
h.delete :first      # => "Jane"
h                    # {:last => "Smith"}
```

Section 2: Hash

Methods - Basics

```
h = {first: "Jane", last: "Smith"}

# keys - returns array of keys:
h.keys     # => [:first, :last]

# values - returns array of values:
h.values   # => ["Jane", "Smith"]
```

Section 2: Hash

Methods - Basics

```
h = {first: "Jane", last: "Smith"}

# flatten:
h.flatten  # => [:first, "Jane", :last, "Smith"]

# invert:
h.invert   # => {"Jane" => :first, "Smith" => :last}
```

Section 2: Hash

Methods - Basics

```ruby
h = {first: "Jane", last: "Smith"}

# merge:
h.merge({mid: "X"})  # => [:first, "Jane", :last, "Smith", :mid => "X"]

h.merge mid: "X"     # => [:first, "Jane", :last, "Smith", :mid => "X"]

# useful for defaults and overrides:
defaults = {city: "Seattle", state: "WA"}
addr1 = defaults.merge street: "123 Main St", zip: "98112"
addr2 = defaults.merge street: "123 Main St", zip: "94101", state: "CA"
```

Section 2: Hash

Methods - Predicates

```
h = {first: "Jane", last: "Smith"}

h.empty?                    # => false

h.has_key? :first           # => true
# aliases:  key?, include?,  member?

h.has_value? "Pat"          # => false
# alias: value?
```

Section 2: Hash

Methods - Generators

```
h = {first: "Jane", last: "Smith"}

# select:
h.select {|k,v| k == :first}    # => {:first => "Jane"}
h.select {|k,v| v == "Smith"}   # => {:last => "Smith"}

# reject:
h.reject {|k,v| k == :first}    # => {:last => "Smith"}
h.reject {|k,v| v == "Smith"}   # => {:first => "Jane"}
```

Section 2: Hash

Methods - Iterators

```
# each:
h.each {|k,v| puts "#{k} => #{v}"}

# each_key:
h.each_key {|k| puts "#{k} => #{h[k]}"}

# each_value:
h.each_value {|v| puts "#{v}"}
```

Section 3

Array

Section 3: Array

Creating an Array

```
# literal syntax:
[1, 2, 3]

Array.new                        # => []
Array.new(3)                     # => [nil, nil, nil]
Array.new(3, "")                 # => ["", "", ""]
Array.new(3) {|i| i**2 }         # => [0, 1, 4]
```

Section 3: Array

Methods - Basics

```
ary = ["John", "Paul", "George", "Ringo"]

# length / count:
ary.length    # => 4
ary.count     # => 4

# first, last:
ary.first     # => "John"
ary.last      # => "Ringo"
```

Section 3: Array

Methods - indexing

```
a = Array.new
a[4] = "4";                      #=> [nil, nil, nil, nil, "4"]
a[0, 3] = [ 'a', 'b', 'c' ] #=> ["a", "b", "c", nil, "4"]
a[1..2] = [ 1, 2 ]               #=> ["a", 1, 2, nil, "4"]
a[0, 2] = "?"                    #=> ["?", 2, nil, "4"]
a[0..2] = "A"                    #=> ["A", "4"]
a[-1]   = "Z"                    #=> ["A", "Z"]
a[1..-1] = nil                   #=> ["A", nil]
a[1..-1] = []                    #=> ["A"]
```

Section 3: Array

Methods - operators

```
# *
[ 1, 2, 3 ] * 3     #=> [ 1, 2, 3, 1, 2, 3, 1, 2, 3 ]
[ 1, 2, 3 ] * ","   #=> "1,2,3"

# +
[ 1, 2, 3 ] + [ 4, 5 ]     #=> [ 1, 2, 3, 4, 5 ]

# -
[ 1, 1, 2, 2, 3, 3, 4, 5 ] - [ 1, 2, 4 ]  #=>  [ 3, 3, 5 ]

# <<
[ 1, 2 ] << "c" << "d" << [ 3, 4 ]  #=>  [ 1, 2, "c", "d", [ 3, 4 ] ]
```

Section 3: Array

Methods - Queue

```ruby
# pop:
ary = ["John", "Paul", "George", "Ringo"]
ary.pop      #=> "Ringo"
ary.pop(2)   #=> ["Paul", "George"]
ary          #=> ["John"]

# push:
ary = ["J", "P", "G", "R"]
ary.push "Yoko", "Linda"  #=> ["J", "P", "G", "R", "Yoko", "Linda"]
```

Section 3: Array

Methods

```
# compact:
["a", nil, "b", nil].compact  #=> ["a", "b"]

# concat:
["a", "b"].concat ["c", "d"]  #=> ["a", "b", "c", "d"]
```

Section 3: Array

Methods

```
# insert:
a = %w{ a b c d }          #=> ["a", "b", "c", "d"]
a.insert(2, 99)            #=> ["a", "b", 99, "c", "d"]
a.insert(-2, 1, 2, 3)      #=> ["a", "b", 99, "c", 1, 2, 3, "d"]

# transpose:
a = [[1,2], [3,4], [5,6]]
a.transpose    #=> [[1, 3, 5], [2, 4, 6]]
```

Section 3: Array

Methods

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# delete:
a.delete 10     #=> nil
a               #=> [1, 2, 3, 4, 5, 6, 7, 8, 9]

a.delete 2      #=> 2
a               #=> [1, 3, 4, 5, 6, 7, 8, 9]

# delete_at:
a.delete_at(0)  #=> 1
a               #=> [3, 4, 5, 6, 7, 8, 9]
```

Section 3: Array

Methods - Predicates

```
beatles = ["John", "Paul", "George", "Ringo"]

# empty?
beatles.empty?  #=> false

# include?
beatles.include? "Paul"  #=> true

# any?
wings = ["Paul", "Linda"]
wings.any? {|e| beatles.include? e}  #=> true

# all?
wings.all? {|e| beatles.include? e}  #=> false
```

Section 3: Array

Methods

```
ary = ["John", "Paul", "George", "Ringo"]

# reverse:
ary.reverse                    #=> ["Ringo", "George", "Paul", "John"]

# sort:
ary.sort                       #=> ["George", "John", "Paul", "Ringo"]
ary.sort {|x,y| y <=> x }      #=> ["Ringo", "Paul", "John", "George"]
```

Section 3: Array

Methods

```
# flatten:
a = [[1,2], [3,4], [5,6]]
a.flatten           #=> [1, 2, 3, 4, 5, 6]

# uniq:
a = [1, 1, 2, 2, 1, 3, 2, 1]
a.uniq              #=> [1, 2, 3]
```

Section 3: Array

Methods

```
ary = ["John", "Paul", "George", "Ringo"]

# join:
ary.join            #=> "JohnPaulGeorgeRingo"
ary.join " "        #=> "John Paul George Ringo"
ary.join ", "       #=> "John, Paul, George, Ringo"
```

Section 3: Array

Methods

```
ary = ["John", "Paul", "George", "Ringo"]

# shift:
ary.shift              #=> "John"
ary                    #=> ["Paul", "George", "Ringo"]

# unshift:
ary.unshift "John" #=> ["John", "Paul", "George", "Ringo"]
```

Methods

```
ary = [3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5]

# max:
ary.max    #=> 9

# min:
ary.min    #=> 1
```

Section 3: Array

Methods - Iterators

```ruby
ary = ["John", "Paul", "George", "Ringo"]

# each:
ary.each {|item| puts item}

# each_with_index:
ary.each {|item, index| puts "#{index}: #{item}"}
```

Section 3: Array

Methods - Iterators

```
a = ["John", "Paul", "George", "Ringo"]

# map:
b = a.map {|item| item[0]}
b   #=> ["J", "P", "G", "R"]
```

Section 3: Array

Methods - Reduce

```ruby
a = [1, 2, 3, 4]

# sum:
a.reduce(0) {|item, acc| acc + item}  #=> 10

# product:
a.reduce(1) {|item, acc| acc * item}  #=> 12
```

Section 3: Array

Methods - Reduce

```ruby
a = [1, 2, 3, 4]

# max:
a.reduce {|item, acc| item > acc ? item : acc}

# min:
a.reduce {|item, acc| item < acc ? item : acc}
```

Section 3: Array

Methods - Finders

```ruby
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# select (aka find_all):
a.select {|item| item % 2 == 0}  #=> [2, 4, 6, 8]

# reject:
a.reject {|item| item % 2 == 0}  #=> [1, 3, 5, 7, 9]

# find (aka detect):
a.find {|item| item % 2 == 0}    #=> 2
```

Section 3: Array

Methods - Delete

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# delete_if:
a.delete_if {|item| item % 2 == 0}  #=> [1, 3, 5, 7, 9]
a                                   #=> [1, 3, 5, 7, 9]
```

Section 4

Basic I/O

Section 4: Basic I/O

Basics

```ruby
# open for reading:
input = File.open("input", "r")

# open for writing:
output = File.open("output", "w+")

# do stuff

# close:
input.close
output.close
```

Section 4: Basic I/O

File modes

```
"r"  read-only
     # starts at beginning of file (default mode).

"r+" read-write
     # starts at beginning of file.

"w"  write-only
     # truncates existing file to zero length or creates new file for writing.

"w+" read-write
     # truncates existing file to zero length or creates new file for reading and writing.
```

Section 4: Basic I/O

Using Blocks

```
# open for reading:
File.open("input", "r") do |input|

  # open for writing:
  File.open("output", "w+") do |output|

    # do stuff

  end

end
```

Reading

```
# read everything into a single string:
contents = file.read_all

# read everything into an array of strings:
ary = file.readlines

# iterate:
file.each_line do |line|
  # do stuff with line
end
```

Section 4: Basic I/O

Writing

```
# print -- without newline:
file.print "some string"
file.print " "
file.print "another string"
#=> file contents: "some string another string"

# puts -- includes newline:
file.puts "some string"
file.puts "another string"
#=> file contents: "some string\nanother string"
```

Section 4: Basic I/O

Standard in/out/err

```ruby
# globals:
$stdin
$stdout
$stderr
```

Section 5
Putting it together

Section 5: Putting it together

Hashes as records

```ruby
def create_address(street, city, state, zip)
  {street: street, city: city, state: state, zip: zip}
end

def create_person(fname, lname, age, addr)
  {fname: fname, lname: lname, age: age, addr: addr}
end
```

Section 5: Putting it together

creating records from input files

```
# line format: fname, lname, age, street, city, state, zip

File.open("input_file") do |input|
  records = input.readlines.map do |line|
    fields = line.split ","
    addr = create_address fields[3], fields[4], fields[5], fields[6]
    create_person fields[0], fields[1], fields[2], addr
  end
end
```

Section 5: Putting it together

Querying, Aggregating

```ruby
# how many people from Washington?
records.select {|person| person[:addr][:state] == "WA"}.length

# how many different states?
records.map {|p| p[:addr][:state]}.uniq.length

# count for each state:
states = records.map {|p| p[:addr][:state]}.uniq
states.reduce({}) do |state, acc|
  acc[state] = records.select {|p| p[:addr][:state] == state}.length
end
```

Section 5: Putting it together

Querying, Aggregating

```
# how many people named "Michael":
records.select do |person|
  person{:fname] == "Michael"
end.length

# how many people named "Michael" from WA:
records.select do |person|
  person[:fname] == "Michael" and person[:addr][:state] == "WA"
end.length
```

Section 5: Putting it together

Querying, Aggregating

```
# calculate average age:
ages = records.map { |person| person[:age] }
sum = ages.reduce {|age, acc| age + acc}
average_age = sum.to_f / ages.length
```

Section 5: Putting it together

Querying, Aggregating

```ruby
# calculate average age by state:
states = records.map {|p| p[:addr][:state]}.uniq
records_by_state = states.reduce({}) do |state, acc|
  acc[state] = records.select {|person| person[:addr][:state] == state}
end
avg_age_by_state = states.reduce({}) do |state, acc|
  ages = records_by_state[state].map {|person| person[:age]}
  sum = ages.reduce {|age, acc| age + acc}
  acc[state] = sum.to_f / ages.length
end
```

Section 6

Rendering HTML

Section 6: Rendering HTML

rendering HTML pages

```
def render_html(title, records)
<<HTML
    <!doctype html>
    <html>
      #{render_head title}
      #{render_body title, records}
    </html>
HTML
end
```

Section 6: Rendering HTML

rendering HTML pages

```
def render_head(title)
<<HEAD
  <head>
    <title>#{title}</title>
  </head>
HEAD
end
```

Section 6: Rendering HTML

rendering HTML pages

```
def render_body(title, records)
<<BODY
  <body>
    <h1>#{title}</h1>
    #{render_records records}
  </body>
BODY
end
```

Section 6: Rendering HTML

rendering HTML pages

```
def render_records(records)
<<RECORDS
  <table>
    #{render_table_header}
    #{records.map {|r| render_record r}.join "\n"}
  </table>
RECORDS
end
```

Section 6: Rendering HTML

rendering HTML pages

```
def render_record(r)
<<RECORD
  <tr>
    <td>#{r[:fname]}</td>
    <td>#{r[:lname]}</td>
    <td>#{r[:age]}</td>
    <td>#{r[:addr][:street]}</td>
    <td>#{r[:addr][:city]}</td>
    <td>#{r[:addr][:state]}</td>
    <td>#{r[:addr][:zip]}</td>
  </tr>
RECORD
end
```

Section 6: Rendering HTML

writing HTML reports

— read input from file
— create records
— transform records
— render output

Section 7

Assignments

Section 7: Assignments

Problem 1 - to_sentence

```
# implement method `to_sentence`

# creates an english string from array

# Your method should generate the following results:
to_sentence []                          #=> ""
to_sentence ["john"]                    #=> "john"
to_sentence ["john", "paul"]            #=> "john and paul"
to_sentence [1, "paul", 3, "ringo"]     #=> "1, paul, 3 and ringo"
```

Section 7: Assignments

Problem 2 - mean, median

```
# implement methods "mean", "median" on Array of numbers

# Your method should generate the following results:
mean [1, 2, 3]     #=> 2
mean [1, 1, 4]     #=> 2

median [1, 2, 3]   #=> 2
median [1, 1, 4]   #=> 1
```

Section 7: Assignments

Problem 3 - pluck

```
# implement method `pluck` on array of hashes

# Your method should generate the following results:
records = [
  {name: "John",   instrument: "guitar"},
  {name: "Paul",   instrument: "bass"  },
  {name: "George", instrument: "guitar"},
  {name: "Ringo",  instrument: "drums" }
]
pluck records, :name        #=> ["John", "Paul", "George", "Ringo"]
pluck records, :instrument  #=> ["guitar", "bass", "guitar", "drums"]
```

Section 7: Assignments

Problem 4 - monthly bank statement

```
# given a CSV file with bank transactions for a single account
# generate an HTML file with a monthly statement

# assume starting balance is $0.00

# the monthly statement should include the following sections:
# - withdrawals
# - deposits
# - daily balance
# - summary:
#   - starting balance, total deposits, total withdrawals, ending balance
```