Crash Course in Ruby

Summer Institute 2018

Data Types in Ruby

- Integers

```
my_int = -25                # -25
large_number = 1_000_000    # 1000000
```

- Floats

```
my_float = 17.2             # 17.2
sum = 16 + 14.0             # 30.0
```

- Boolean

```
result = 6 > 10             # false
```

- Strings

```
my_str = "Hello World!"     # "Hello World!"
literal_str = 'Hello back!' # "Hello back!"
```

- Arrays

```
sea_creatures = ["shark", "squid"]
sea_creatures[1]            # "squid"
sea_creatures << "guppy"    # ["shark", "squid", "guppy"]

my_ary = [true, 5, "string"]
my_ary.size                 # 3
```

- Symbols (acts like a label or identifier in Ruby)

```
format = :json              # :json
is_completed = (job_state == :completed)       # true
```

- Hashes (dictionary-like collection of keys and values)

```
user = {"first_name" => "John", "last_name" => "Smith"}
user["first_name"]                             # "John"

user = {:first_name => "Jane", :last_name => "Smith"}
user[:last_name]                               # "Smith"

user = {first_name: "Sammy", last_name: "Shark"}
user[:first_name]                              # "Sammy"

crazy = {1 => "abc", "abc" => 1, completed: true}
```

- Nil (absence of a value, this is falsey)

```
my_var = nil
result = "success" if my_var                   # nil
```

- Regexp (regular expression used to match patterns)

```
/ell/ =~ "hello"            # 1
/needle/ =~ "haystack"      # nil
```

Conditional Tests

- Ruby has reserved keywords (if, else, elsif, end):

```
a = 5
if a < 0
  puts "Negative number"
elsif a < 10
  puts "Small positive number"
else
  puts "Large positive number"
end
```

- Comparison operators are: `<=, <, >, >=, ==, !=`
- You can combine expressions using `&&` (and) and `||` (or) operators as well as use `!` (not):

```
(1 == 1) && (2 == 2)       # true
(1 == 2) && (2 == 2)       # false
(1 == 2) || (2 == 2)       # true
(1 == 2) || !(2 == 2)      # false
```

- For simple false cases you can use the keyword unless:

```
unless a.even?
  puts "Your number is odd"
end
```

- You can append the conditional for simple statements:

```
set_value = true
value = 5 if set_value                # 5
value = value + 1 unless value.odd?   # nil
value                                 # 5
```

- For simple cases you can use the ternary operator for terseness:

```
true ? "this is true" : "this is false"
# "this is true"

false ? "this is true" : "this is false"
# "this is false"
```

Conditional Quiz

- Determine the output for the following code:

```
if (2 == 3) || ( (4 == 4) && !(5 == 6) )
  puts "You speak the truth"
else
  puts "All hope is lost"
end


value = nil
value ? "value is truthy" : "value is falsey"


value = "octopus"
value ? "value is truthy" : "value is falsey"


nil == false ? "Nil is false" : "Nil is not false"
```

Strings in Depth (https://ruby-doc.org/core-2.2.0/String.html)

- String can be initialized with single or double quotes:

```
user = "bob"      # can interpolate, see below
state = 'texas'   # literal string
```

- Will need to escape quotes similar quotes:

```
book = "The \"Amazing\" Book"
book = 'The "Amazing" Book'
```

- Can interpolate values in strings:

```
x, y = 12, 36
output = "#{x} + #{y} = #{x+y}"
# 12 + 36 = 48
output = '#{x} + #{y} = #{x+y}'
# #{x} + #{y} = #{x+y}
```

- General delimited strings using (, {, <, and more:

```
output = %Q("x" is #{x})
# "x" is 12
output = %q["x" is #{x}]
# "x" is #{x}
```

- Strings are objects:

```
name = "bob"
name.capitalize   # "Bob"
name.length       # 3

name = "    Alice        "
name.strip        # "Alice"

"this is a string".split
# ["this", "is", "a", "string"]

"foot".gsub("o", "e")      # "feet"
```

String Quiz

- Given the following strings, how could I combine them into a new string that displays the full name:

```
first_name = "John"
last_name = "Smith"

#
# code here
#

full_name      # "John Smith"
```

Arrays in Depth (https://ruby-doc.org/core-2.2.0/Array.html)

- Array with nothing in it:

```
empty_array = []
```

- You can put anything into an array:

```
my_array = [1, "hello", true, nil, 220.5]
```

- Arrays start with index 0:

```
my_array[0]      # 1
my_array[1]      # "hello"
my_array[100]    # nil
```

- Arrays are objects:

```
my_array.first   # 1
my_array.last    # 220.5
my_array.size    # 5
my_array.push("fred")
my_array.last    # "fred"
```

- Can get fancy:

```
animals = ["fish", "cat", "dog"]
animals.sort     # ["cat", "dog", "fish"]
animals          # ["fish", "cat", "dog"]
animals.sort!    # ["cat", "dog", "fish"]
animals          # ["cat", "dog", "fish"]

animals.map { |animal| animal.capitalize }
# ["Cat", "Dog", "Fish"]
```

- You can use shorthand to append to arrays:

```
animals << "goat"
animals << "car" if false
animals          # ["fish", "cat", "dog", "goat"]
```

- More helpful methods on Arrays

```
# subarrays
animals.take(2)   # ["fish", "cat"]
animals.drop(2)   # ["dog", "goat"]
animals[1..3]     # ["cat", "dog", "goat"]

# filters
[1, 2, 3, 4, 5].select(&:even?)      # [2, 4]
[1, 2, 3, 4, 5].reject(&:even?)      # [1, 3, 5]

# sorting
["doggy", "cat", "fish"].sort
# ["cat", "doggy", "fish"]
["doggy", "cat", "fish"].sort_by(&:length)
# ["cat", "fish", "doggy"]
```

Array Quiz

- Solve the quizzes below:

```
numbers = [1, 2, 3, 4, 5]

# create array of elements that say whether value is even
# [false, true, false, true, false]
```

```
manly = ["batman", "manbot", "mace", "tulip", "nah man, nah"]

# create array of elements in manly array that contain word "man"
# ["batman", "manbot", "nah man, nah"]
```

Iteration (https://ruby-doc.org/core-2.2.0/Enumerable.html)

- Can iterate over each element of an array:

```
[1, 2, 3].each do |value|
  puts value
end

[1, 2, 3].each { |value| puts value }
```

- Can iterate over elements with index:

```
["a", "b", "c"].each_with_index do |value, idx|
  puts "#{value} located at index #{idx}"
end
```

- Can transform array values (use `map!` to transform in-place):

```
[1, 2, 3].map { |v| v + 1 }
# [2, 3, 4]
```

- Can iterate over ranges:

```
(1..10).each { |v| puts v }
('a'..'e').each { |char| puts char }
```

- Shorthand for performing single method on object:

```
["bob", "steve"].map(&:capitalize)
# ["Bob", "Steve"]

# Equivalent to…
["bob", "steve"].map { |obj| obj.capitalize }
# ["Bob", "Steve"]
```

Hashes in Depth (https://ruby-doc.org/core-2.2.0/Hash.html)

- Hash with nothing in it:

```
homes = {}
```

- You can associate a value with a key:

```
homes["jim"] = "texas"
homes["betty"] = "alaska"
```

- You can retrieve a value, given a key:

```
homes["jim"]              # "texas"
```

- Hashes are objects:

```
homes.length              # 2
homes.empty?              # false
homes.keys                # ["jim", "betty"]
homes.values              # ["texas", "alaska"]
```

- Can get fancy:

```
homes.each do |key, value|
  puts "#{key} is from #{value}"
end
# jim is from texas
# betty is from alaska

homes.fetch("billy", "unknown")   # "unknown"

default_config = {a: 1, b: "string"}
config = {a: 2, c: false}
default_config.merge(config)
# {a: 2, b: "string", c: false}
```

Hash Quiz

- Given the following hash, how can I generate a new hash with all the keys symbolized (you can use the `String#to_sym` method) and dropping any keys with `nil` values:

```
user = {"name" => "bob", group: "PZS0001", permissions: nil}

# Create compact hash with symbolized keys
# {name: "bob", group: "PZS0001"}
```

More at http://www.codequizzes.com/ruby/intermediate/hash-class

Functions

- Function arguments can be called with or without parenthesis, but best to use them when in doubt:

```
puts "hello"
puts("hello")
```

- You can define your own functions:

```
def add(x, y)
  puts "x + y = #{x+y}"
end
```

- The return value of a function is the return value of the last evaluated statement:

```
def make_positive(number)
  number < 0 ? –number : number
end

new_number = make_positive(-5)        # 5
new_number = make_positive(10)        # 10
```

Classes

- Create simple objects:

```
class Dog
  def bark
    puts "Bark! Bark!"
  end
end

dog = Dog.new
dog.bark
# "Bark! Bark!"
```

- Initialize our object:

```
class Dog
  def initialize(breed)
    @breed = breed
  end

  def bark
    puts "#{@breed}: Bark! Bark!"
  end
end

dog = Dog.new("terrier")
dog.bark
# terrier: Bark! Bark!
```

- Create getters/setters for breed:

```
class Dog
  def initialize(breed)
    @breed = breed
  end

  def breed
    @breed
  end

  def breed=(new_breed)
     @breed = new_breed
  end

  def bark
    puts "#{breed}: Bark! Bark!"
  end
end

dog = Dog.new("terrier")
dog.bark
# terrier: Bark! Bark!

dog.breed = "labrador"
dog.bark
# labrador: Bark! Bark!
```

- Use Ruby helpers for getters/setters:

```
class Dog
  # creates "def breed" and "def breed="
  attr_accessor :breed

  def initialize(breed)
    @breed = breed
  end

  def bark
    puts "#{breed}: Bark! Bark!"
  end
end
```

Class Quiz

- Create a Ruby script with a class that can be initialized with a string:

```
word_counter = WordCount.new("the day is sunny the the the sunny is is")
```

- Then create a method on that class that allows a user to view the frequency of each word in that string as a list of hashes:

```
word_counter.word_count
# [
#   { word: "the", count: 4 },
#   { word: "is", count: 3 },
#   …
# ]
```

- Bonus: Add a `#to_s` method that prints the frequency of the words to the screen in a tabular format.
- Bonus: Read in the string from STDIN:

```
$ echo "this is is the new string" | ruby my_script.rb
```

Other topics

- The require and require_relative keywords
- Passing and calling blocks in functions
- The ||= operator
- The backtick operator
- Scope of local, instance, class, and global variables
- Regular expressions in more detail
  - Metacharacters: ., \w, \W, \d, \D, \s, \S
  - Character classes: […], [^…]
  - Repetition: *, +, ?, {n}, …
  - Capturing: (…)
  - Anchors: ^, $, \A, \Z, \b

Sinatra App

- Copy base Sinatra App (https://github.com/OSC/si-18-sinatra-base)
- Create a class that calls the `` `ps aux` `` command and parses it into a simple data structure (e.g., array of hashes, bonus if array of process objects)
- Output to a table on the index page
- Add simple filters, e.g., https://..../?user=jnicklas
  - &sort=cpu:desc or sort=mem:asc