MIS 21

R. Alampay, C. Ruiz, N. Victorino

Learning Objectives

- Learn basic ruby syntax
- Understand classes and types
- Execute ruby from the command line
- Execute ruby using a web framework

Ruby

- Designed and developed by Yukihiro "Matz" Matsumoto in the mid-1990s
- Functional, object-oriented and imperative
- Dynamic type system
- Designed for programmer productivity and fun
- Operates under Principle of Least Surprise
  - Language should behave in such a way that minimizes confusion
- Uses a lot of syntactic sugar

Hello World (Ruby Version)

- Create a file called "hello.rb"
  - Ruby files usually have the extension rb
- Enter the following code: `puts 'hello, world'`
- In your command line, navigate to the folder where file exists
- Type "ruby hello.rb"
  - ruby is the ruby interpreter
- We can also run ruby commands via **irb** (*interactive ruby shell)*

Hello World (con'td)

- Try the following lines
  - `puts "hello, world"`
  - `puts('hello,world')`
  - `puts 'hello', ' world'`
- `puts` is a method/function
- Methods have parameters/arguments
  - parenthesis is optional and ruby
- Also try: `p, print`

Variables

- A symbolic name which contains some known or unknown value.
- Try:

```
message = "hello, world"
puts message
```

- Variables can be any text that is not a ruby keyword
- Type is not implied
- Ideally, you want a variable to be descriptive of the value that it contains
- Syntax
  - `variable = value`
- Snake case is used: multiple words separated by underscore i.e. `long_variable_name`

Variables

- In ruby, everything is an object
  - Everything has properties and methods
  - No primitive types
- In ruby, everything is an expression
  - Every line of code returns a value
- Since no type is specified it is the developers responsibility to ensure what is contained in a varible and how it is used
  - Duck Typing (more on this later)

Reflection

- The ability of classes to define themselves is called *reflection*
- Reflection is a key component in metaprogramming – i.e. writing programs that can write other programs
- Reflection can be used to determine properties about a class or variable
  - `[something].methods` will give us a list of methods of something
  - `[something].class` will give us the class of something

String

- `irb> "this is a string".class`
- Represents literal values
- Enclosed by either double quotes or single quotes
- Double quotes are strings that allow *interpolation*

```
message = 'hello'
puts "the message is #{message}"
```

- Interpolation allows you to place variables inside a string (instead of concatenating it)
- http://www.ruby-doc.org/core-2.1.1/String.html

Numbers

- Several classes are used for numbers
- `1.class -> Fixnum`
- `Fixnum.superclass -> Integer`
- `111_000_000.class -> Fixnum`
- `1111111111.class -> Bignum`
- `Bignum.superclass -> Integer`
- `1.2345.class -> Float`

Operations and methods

- +,-,/,*,%
- n ** m -> raises n to the power of m
- 2.even? -> returns true
- 2.odd? -> returns false
- 2.next -> returns 3
- http://www.ruby-doc.org/core-2.1.1/Float.html
- http://www.ruby-doc.org/core-2.1.1/Fixnum.html

Nil

- When referring to something that does not exists, the value of `nil` is used.
- Even nil is an object
  - `nil.class -> NilClass`

Boolean

- Used to represent logical (true or false) values
- Logical Operators
  - `>, <, <=, >=, !=`
  - `and` and `&&`
  - `or` and `||`
  - Generic equality (`obj == other`) -> returns true if `obj` and `other` are the same object
- Most methods that return a boolean value usually have a ? at the end of the method name

Symbols

- Symbols are "special strings"
  - Memory efficient
  - Faster to compare
  - Mutable (cannot be changed)
  - Usually not used for display/printing out
- Typically used to represents symbols and names
- Examples
  - `:test`
  - `:a`
  - `:symbol`
- It is a common convention to use symbols as indexes or keys

Arrays

- Arrays collect objects into one variable
- An array can contain any set of values
- The values in an array need not be the same type
- http://www.ruby-doc.org/core-2.0/Array.html

```
arr = [1,2,'three',:big]
arr.size #returns 4
arr << 5 #insert 5 at end of arr
arr[1] #returns 2
```

Enumerable

- Enumerable is a *mixin* that provides for collection classes (like Array)
  - *mixins* are a way of adding functionality to classes

```
arr.first
arr.each { |x| puts x }
arr.reverse_each do |x|
     puts x
end
```

Hashes

- A collection of unique keys and values
  - Also called associative arrays or dictionaries
  - Similar to arrays but allows you to specify any kind of object as it's index
- http://www.ruby-doc.org/core-2.0/Hash.html

```
hsh = {:name => 'Juan', :age => 15}
hsh[:name] #returns 'Juan'
another = {"blah" => 25, "etc" => 5}
another.each_pair {|key, value|
puts "#{key}:#{value}"}
```

Ranges

- Try the ff:

```
(1..5).each {|i| puts i }

("a"..."z").each do |letter|
     puts letter
end

(10..20).include?(15)
```