Ruby

A review

Programming Basics

What is programming?

- Defining commands
- Issuing them
- Ensuring they get executed

Variables - Algebra

- Remember your algebra!
    - $5 + 9 = 14$
    - $5 + x = 14$
    - Solve for x.

Variables: Containers for Values

- x = 5
- y = “Jonathan”
- q = x + r
- Variables can hold many data types.

Data Types

- Float
  - Numbers with decimal points
  - 10.32, 65.323, .32
- Integer
  - Natural numbers
  - 11, 53, 3

Data Types 2

- Strings
  - A line of text
  - “Jon”, “Elephants are awesome”
- Booleans
  - True or False

Data Types 3

- Arrays
  - A collection of values
  - [5,3,4,"omega", 15.3, "cappa"]
- When assigned to variables, can be accessed with brackets[]
  - a = [5,3,4,"omega", 15.3, "cappa"]
  - a[0] == 5, a[4] == 15.3

Hashes

- Another way to store data - similar to an array
- Used to store key => value pairs
- myHash = {“jan” => “January”, “feb” => “February”}
- To access value: myHash[“January”]

Conditions

- A condition is a test for something

```
if (2 == 2)

        puts “2 really does equal 2!”

end
```

Conditions II

- Important key words:
  - if
  - elsif
  - else
  - end

Loops

- Repetitive conditions where one variable inside a block of code changes
- There are a few different kinds:
  - for
  - while
  - each
  - begin...rescue...end

For Loops

```
for i in 0..4
  puts “loop #{i}”
end
```

While Loops

```
u = 5
while u<10
  puts u
  u = u + 1
end
```

Each Loops

- Loop over an array

```
p = [“This”, “is”, “awesome”]
p.each do |item|
  puts item
end
```

Begin...Rescue...End

- Executes a block if the first block fails

```
begin
  “Oranges” + 4
rescue
  “Oranges” + “Apples”
end
```

Methods

- Methods are shortcuts to a block of code
- They take arguments
- They typically return a value

Method Syntax

```
def addTwo(n)
  n + 2
end
```

Objects

- Objects
  - A representation of something in the real world
  - Has properties and methods
  - For instance - a Car has an engine, and a method to start the engine

Object Syntax

```
class Bar
  attr_accessor :brand
  attr_accessor :wheels
  def turn_on_engine
   puts “engine is on”
  end
end
```

Instance vs Class Methods

- Instance methods require a new instance of the class to work (def addTwo)
- Class methods will work without a new instance (def self.addTwo)

String Information

- Is the string empty?

```
awesomeString.empty?
>true
```

- Length

```
awesomeString.length
```

String Concatenation

- Concatenation: bringing two strings together

```
awesomeString = “Hello” + “World”
```

- No + necessary, as long as it’s just two strings (no variables)

```
awesomeString = “Hello” “World”
```

String Concatenation II

- Chains

```
awesomeString = “Hello” << “World”
```

- Concat method

```
awesomeString = “Hello”.concat(“World”)
```

String Interpolation

- An easy way to put a variable inside

```
awe = “Hello”

run = “World”

u = “Well #{awe}, #{run}!”
```

More String Fun

- Freezing

```
awesomeString = “Hello” << “World”

awesomeString.freeze
```

- Objects cannot be unfrozen once locked, only duplicated
- Once frozen, a string is “immutable”

Accessing String Elements

- Searching

```
awesomeString = “Hello World”

awesomeString[“Hello”]
```

- Parts of the string

```
awesomeString[0]

>”H”

awesomeString[0,3]

>”Hel”

awesomeString[0..6]

>”Hell”
```

String Manipulation

- “Any string” can be manipulated in many ways
- For instance - words can be replaced

```
yourString = “Hello World!”
yourString[“World”] = “Universe”
yourString
```

String Manipulation

- Substitution

```
yourString = “Hello World!”

yourString.gsub “Universe”, “World”
```

- Repeating

```
yourString * 3

Hello World!Hello World!Hello World!
```

String Manipulation

- Inserting text

```
yourString = “Hello World!”
yourString.insert 5, “ to the”
>”Helloto the world!”
```

String Manipulation

- Chomp and chop

```
yourString = “Hello World! H”

yourString.chop

>”Hello World ”

yourString.chomp > “Hello World”
```

- Reverse

```
yourString.reverse

> “!dlroW olleH”
```

Further String Methods

- .upcase
- .downcase
- .swapcase
- .capitalize

Here Documents

- Heredocs are free-format strings.
- They allow you to specify long strings easily

```
yourText = <<DOC

Hello Sir,

I know you enjoy learning about programming.

DOC
```

Gems, Bundler, & Sinatra

Libraries and RubyGems

- Library - collection of methods and classes
- Has an easy way to access them
- Available for many prog. languages
- Ruby's "library system" or "package manager" is known as **RubyGems**

Using Gems

- Installation **gem install mygem**
- Uninstallation **gem uninstall mygem**
- List all installed **gem list --local**
- When using RVM - sudo command is typically not necessary to install

Bundler Gem

- Makes installing other gems super easy
- **gem install bundler**
- Use a Gemfile to describe which gems you'd like
- Resource for great gems:
  - www.therubytoolbox.com

Gemfile

- Contains source of gems at the top
  - source ‘http://rubygems.org’
  - Then all the gems you’d like to use
    - gem ‘sinatra’
    - gem ‘will_paginate’

Gemfile II

- Groups are used for different dev environments
- group :group_name do | gems | end
  - Development
  - Test
  - (Staging)
  - Production
- Run **bundle install** to install all gems at once

Sinatra

- Sinatra is a DSL - "Domain Specific Language"
- DSLs are libraries/packages used to solve a very specific problem
- Sinatra is a web application framework - similar to Ruby on Rails

Using Sinatra

- Install the gem - **gem install sinatra**
- Create your main file - <filename>.rb
- Use the code on the next page

Your First Webapp

```
require ‘sinatra’


get ‘/hi’ do
  “Hello World!”
end
```

To Run It

- ruby yourapp.rb

Make your own!

- Create a Sinatra web application with several different routes
- Use methods to encapsulate functionality
- Sample: sin_class.rb