Ruby Programming Language

<http://ruby-doc.org/>

Company: blinq.com

Update Ruby from version 2.0.0 to 2.4.0

Steps:

1. Allow the usr/local writeable

gregorydombchik local $ sudo chown -R $(whoami) /usr/local

1. Update HomeBrew

gregorydombchik local $ brew update

1. Install rvm

curl -L https://get.rvm.io | bash -s stable

1. Install ruby 2.4.0

rvm install ruby-2.4.0

1. Check to make sure the system is using ruby 2.4.0

ruby –v

ri to read ruby documenation

1. Generates all the ruby documentation for ri
2. How to Use
3. ri gets
4. ri chomp

Learn ruby the hard way

<http://learnrubythehardway.org/book/>

**Exercise 0**

<https://atom.io/>

ruby –v

Interactive Ruby Shell:

gregorydombchik ruby $ irb

irb(main):001:0>

**Exercise 1: A Good First Program (ex1.rb)**

puts “puts prints text.”

**Exercise 2: Comments and Pound Characters (ex2.rb)**

# A comment, this is so you can red your program later.

**Exercise 3: Numbers and Math (ex3.rb)**

Calculations :

#Order of Operation

#PEMDAS which stands for Parentheses Exponents Multiplication Division Addition Subtraction.

#3 + 2 + 1 - 5 + 0 - 1 / 4 + 6

#3 + 2 + 1 - 5 + 0 - 0 + 6

#6 - 5 - 0 - 0 + 6

puts 3 + 2 + 1 - 5 + 4 % 2 - 1 / 4 + 6

Answer🡪7

**Exercise 4: Variables And Names (ex4.rb)**

**Exercise 5: More Variables and Printing (ex5.rb)**

**Exercise 6: Strings and Text (ex6.rb)**

**Exercise 7: More Printing (ex7.rb)**

Can I use single-quotes or double-quotes to make a string or do they do different things?

In Ruby the " (double-quote) tell Ruby to replace variables it finds with #{}, but the ' (single-quote) tells Ruby to leave the string alone and ignore any variables inside it.

#Double quotes allows to use variables

puts "Its fleece was white as #{'snow' + hello}."

#Only a string and can not use variable

puts 'This is a test #{hello}'

**Exercise 8: Printing, Printing**

formatter = "first: %{first} second: %{second} third: %{third} fourth: %{fourth}"

puts formatter % {first: 1, second: 2, third: 3, fourth: 4}

This is the result:

first: 1 second: 2 third: 3 fourth: 4

**Exercise 9: Printing, Printing, Printing**

#\n - New Line

months = "\nJan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"

#The % Notation

# %q[ ] ---> Non-interpolated String (except for \\ \[ and \])

puts %q{

There's something going on here.

With the three double-quotes.

We'll be able to type as much as we like.

Even 4 lines if we want, or 5, or 6.

}

**Exercise 10: What was that?**

| Escape | What it does. |
| --- | --- |
| \\ | Backslash () |
| \' | Single-quote (') |
| \" | Double-quote (") |
| \a | ASCII bell (BEL) |
| \b | ASCII backspace (BS) |
| \f | ASCII formfeed (FF) |
| \n | ASCII linefeed (LF) |
| \r | ASCII Carriage Return (CR) |
| \t | ASCII Horizontal Tab (TAB) |
| \uxxxx | Character with 16-bit hex value xxxx (Unicode only) |
| \v | ASCII vertical tab (VT) |
| \ooo | Character with octal value ooo |
| \xhh | Character with hex value hh |

**Exercise 11: Asking Questions**

print "How old are you? "

age = gets.chomp

puts "So, you're #{age} old."

**Exercise 12: Prompting People for Numbers**

# Add to\_i to chomps will convert to an integer

print "Please provide me with cash (dollars): "

number = gets.chomp.to\_f

# Add to\_f to chomps will convert to a float

print "Please provide me with cash (): "

number = gets.chomp.to\_f

**Exercise 13: Parameters, Unpacking, Variables**

The ARGV is the "argument variable," a very standard name in programming, that you will find used in many other languages.

Line 1 "unpacks" ARGV so that, rather than holding all the arguments, it gets assigned to three variables you can work with: first, second, and third. This may look strange, but "unpack" is probably the best word to describe what it does. It just says, "Take whatever is in ARGV, unpack it, and assign it to all of these variables on the left in order."

**Exercise 14: Prompting and Passing**

You should use $stdin.gets.chomp from now on since the action gets.chomp has problems with ARGV.

first1,second2 = ARGV

user\_name\_first = first1 # gets the first argument

user\_name\_second = second2 # gets the second argument

prompt = '> '

**Exercise 15: Reading Files**

ri "File#open"

txt = open(filename)

puts txt.path()

puts "Here's your file #{filename}"

print txt.read

**Exercise 16: Reading and Writing Files**

close -- Closes the file. Like File->Save.. in your editor.

read -- Reads the contents of the file. You can assign the result to a variable.

readline -- Reads just one line of a text file.

truncate -- Empties the file. Watch out if you care about the file.

write('stuff') -- Writes "stuff" to the file.

**Exercise 17: More Files**

Ruby – Cucumber references

<https://www.cs.colorado.edu/~kena/classes/5828/s12/lectures/09-bddcucumber.pdf>

<http://www.rubyinside.com/media/poignant-guide.pdf>

Ruby Gems Using Bundler and installing Cucumber

1. Install Bundler

$ gem install bundler

1. Create Gemfile

$ bundle init

1. Create Gemfile for Cucumber

# frozen\_string\_literal: true

source "https://rubygems.org" do

# gem "rails"

group :test do

gem 'cucumber'

end

end

1. Install the Gemfile

$ bundle install

1. Ruby Paths
   1. Ruby Environment

$ gem env

* 1. Gem Path

/Users/gregorydombchik/.rvm/gems/ruby-2.4.0/gems

1. Initialize Cucumber File using Cucumber Init

gregorydombchik cucumber $ bundle exec cucumber --init

create features

create features/step\_definitions

create features/support

create features/support/env.rb

1. Execute the feature file

gregorydombchik features $ bundle exec cucumber cukes.feature --require step\_definitions/cukes.rb cukes.feature -t @cukes1,@cukes2

Feature: Test File

@cukes1

Scenario: Some cukes # cukes.feature:4

Given I have **48** cukes in my belly # step\_definitions/cukes.rb:1

This is a test --> Cukes: 48

@cukes2

Scenario: More cukes # cukes.feature:8

Given I have **50** cukes and **60** cukes in my belly # step\_definitions/cukes.rb:5

Cukes 1: 50 and Cukes 2: 60

2 scenarios (2 passed)

2 steps (2 passed)

0m0.004s