Ruby Programming

Basics

Overview

- Requirements
- Ruby Overview
- Variables
- Functions
- Arrays and Hashes
- Symbols
- Control Structures
- Blocks and Iterators

Requirements

- Ruby installation
- Text Editor
  - Save in plain text format
  - Recommended editors:
    - Sublime Text 2 (http://www.sublimetext.com/2)
    - TextMate (Mac OS X only)
    - Gedit (http://projects.gnome.org/gedit/)
    - Notepad++
    - Vim
- Command line/shell

Command Line Survival Guide

Listing files

Mac/Linux/*nix:
$ ls

Windows:
C:/> dir

Entering a Directory

Absolute path:
ls /path/to/some/directory

Relative path:
/Users/home$ cd Desktop

Windows Installation

Windows Installation

- Difficult to install in Windows
  - It's not that Windows is terrible...it's just that it's really really terrible (joke lang)
    - Ruby community is biased towards Unix environments
  - Highly recommended to NOT install it from scratch and use pre-built packages:
    - http://rubyinstaller.org/
    - https://github.com/vertiginous/pik

Mac/Linux/*nix Installation

- RVM (Ruby Version Manager)
  - Command line tool to manage multiple ruby installations
  - https://rvm.io/
  - Gems (ruby libraries) are managed easily and are independent of ruby versions

Mac/Linux/*nix Installation

```
user$ \curl -L https://get.rvm.io | bash -s stable --ruby
```

```
user$ rvm list known
# MRI Rubies
...
[ruby-]1.8.7[-p334]
[ruby-]1.8.7-head
```

```
user$ rvm use 1.9.2
Using /Users/user/.rvm/gems/ruby-1.9.2-p180
```

```
user$ ruby -v
ruby 1.9.2p180 (2011-02-18 revision 30909) [i386-darwin9.8.0]
```

Ruby Overview

- Ruby is...
  - Object oriented
  - Dynamic/Interpretted
  - Reflective
- Created in February 24, 1993 by Yukihiro Matsumoto

Ruby Overview

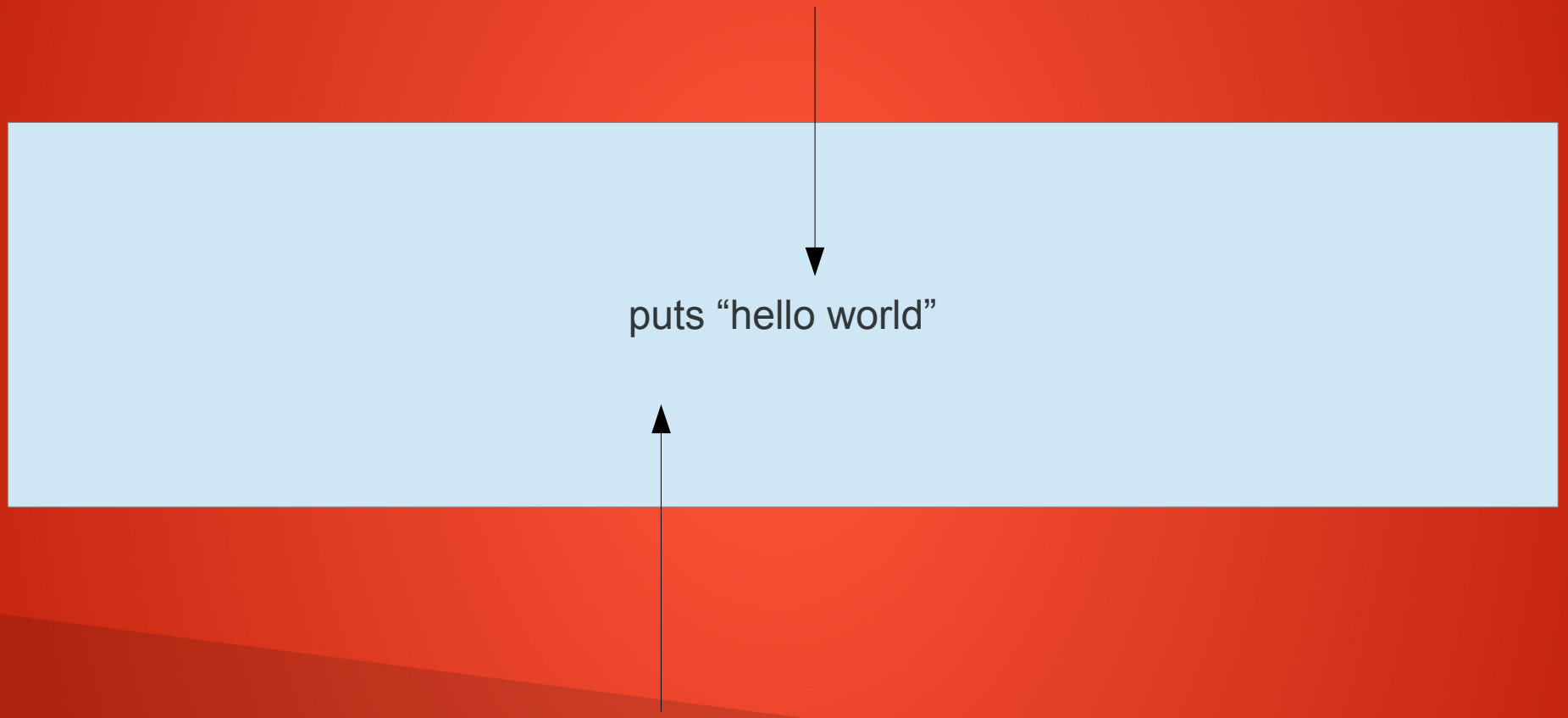
- Ruby is a general purpose language
  - Create games (http://rubygame.org/)
  - Desktop applications (http://en.wikibooks.org/wiki/Ruby_Programming/GUI_Toolkit_Modules)
  - Web applications (Ruby on Rails)
  - Scripting
    - Math homework
    - Drive a car
    - Rocket science

Running Ruby

- 2 main ways in running Ruby:
  - Interactively run ruby code
    - IRB (run irb via command line)
    - http://tryruby.org/ (online irb)
  - Write some ruby code and save it as a *.rb before feeding it to the interpretter
    - Script style
    - chmod u+x yourscript.rb

Hello World

“hello world” is an argument to the function



“puts” is a function

Hello World

- From your common Java/C#/PHP programming styles, what's missing/different?

```
puts “hello world”
```

```
puts(“hello world”);
```

R.I.P.

Variables

variable = “value”

Variables in Strings

Example 1

```
variable = “world”

puts “Hello #{variable}!”
```

Example 2

```
x = 1
y = 1
z = x + y
puts “#{x} plus #{y} is #{z}”
```

Object Oriented Ruby

- Everything in Ruby is an object
  - Instances of an object/class are created
  - Each object/class has some attributes or behaviors
  - Most methods are built into the object as opposed to calling some external method

```
some_word = “Hello world!”
some_word.length
some_word.index(“e”)

num = -1234
positive = num.abs
```

Dynamic Language

- Ruby knows the type of the value being passed to it
- Determined at runtime

```
some_word = “Hello world!” // string
some_num = 1 // integer
```

Functions

- Anatomy of a function (the Java/C# way):

Return type

Function name

```
public [data_type] [function_name] (type a, type b … type n)
{
    // Put some logic here

    return variable;
}
```

Use the return keyword to return a variable with type data_type

Variables local to the function

Functions

- The Ruby way:

May or may not have a return value

Starts with a “def” keyword followed by the function name

```
def my_function(a, b, c … n)
      // some code

      z = a + b + c

      return z
end
```

Optional return statement. The function returns whatever is in the last line

No more {}. Makes use of tabs and the end keyword

Functions

- Example: Adder function

```
def adder (x, y)
      z = x + y
end

x = 2
y = 2

z = adder 2, 2

puts “The sum of #{x} and #{y} is #{z}”
```

Optional parenthesis

Last line is returned

Arrays

- A collection of objects
- Accessed through an integer key
- Instantiated using an array literal [ ]

```
snsd_data = [“SNSD”, 9, 18]
puts “Array snsd_data length: #{snsd_data.length}”
puts “Array snsd_data: #{snsd_data.inspect}”
puts “#{snsd_data[0]} has #{snsd_data[1]} members and #{snsd_data[2]} legs”
```

Arrays

- Array of strings can be time consuming to type
- Optional shortcut version %w{ }

```
snsd_top_three = [
        “Seohyun”,
        “Taeyeon”,
        “Yuri”
]

snsd_top_three = %w{ Seohyun Taeyeon Yuri }
```

w is right beside {

No “,” and “ “

Hashes

- A collection of objects that uses an object key
- Instantiated using curly braces { }
- Each element is made up of two objects
  - Key
  - Value

```
snsd_data = {
        “first” => “Seohyun”,
        “second” => “Taeyeon”,
        “third” => “Yuri”
}
puts “First place: #{snsd_data['first']}”
puts “Second place:: #{snsd_data['second']}”
puts “Third place: #{snsd_data['third']}”
```

Symbols

- Unique variables for some significant value
- Actual values are sometimes irrelevant
  - You just need unique identifiers

```
FIRST = 1
SECOND = 2
THIRD = 3

some_function(FIRST)
```

Symbols

- Symbols are constant names that you don't have to predeclare and are guaranteed to be unique
- Ruby manages these values

```
snsd_data = {
        :first => “Seohyun”,
        :second => “Taeyeon”,
        :third => “Yuri”
}
puts “First place: #{snsd_data[:first]}”
puts “Second place:: #{snsd_data[:second]}”
puts “Third place: #{snsd_data[:third]}”
```

Control Structures

- Ruby has all the common control structures (if, else, while, do, for) plus more

```
today = Time.now

if today.saturday?
      Puts “Study Ruby programming”
elsif today.sunday?
      Puts “Go to church”
else
      Puts “Go to work”
end
```

Control structures use dynamic spacing and the “end” keyword to define its block

Not a typo. It's really spelled “elsif”

Blocks

- Chunks of code you can associate with method invocations
  - Creating a capsule of code before executing it
  - “reifying” code
    - Make to a thing
    - Latin: res meaning thing
    - Thing → thingify. O_o

```
do
    puts “Hello world!”
end

some_method(param1, param2) do

end
```

Begins with “do”

Code Blocks

- Example: numfix (or Ruby integer objects) has a method called times which will execute a block passed to it

```
3.times do
    puts “Trouble”
end

puts “Nareul noryeosseo Neoneun”

3.times do
    puts “Shoot!”
end

puts “Neoneun”

3.times do
    puts “Hoot!”
end
```

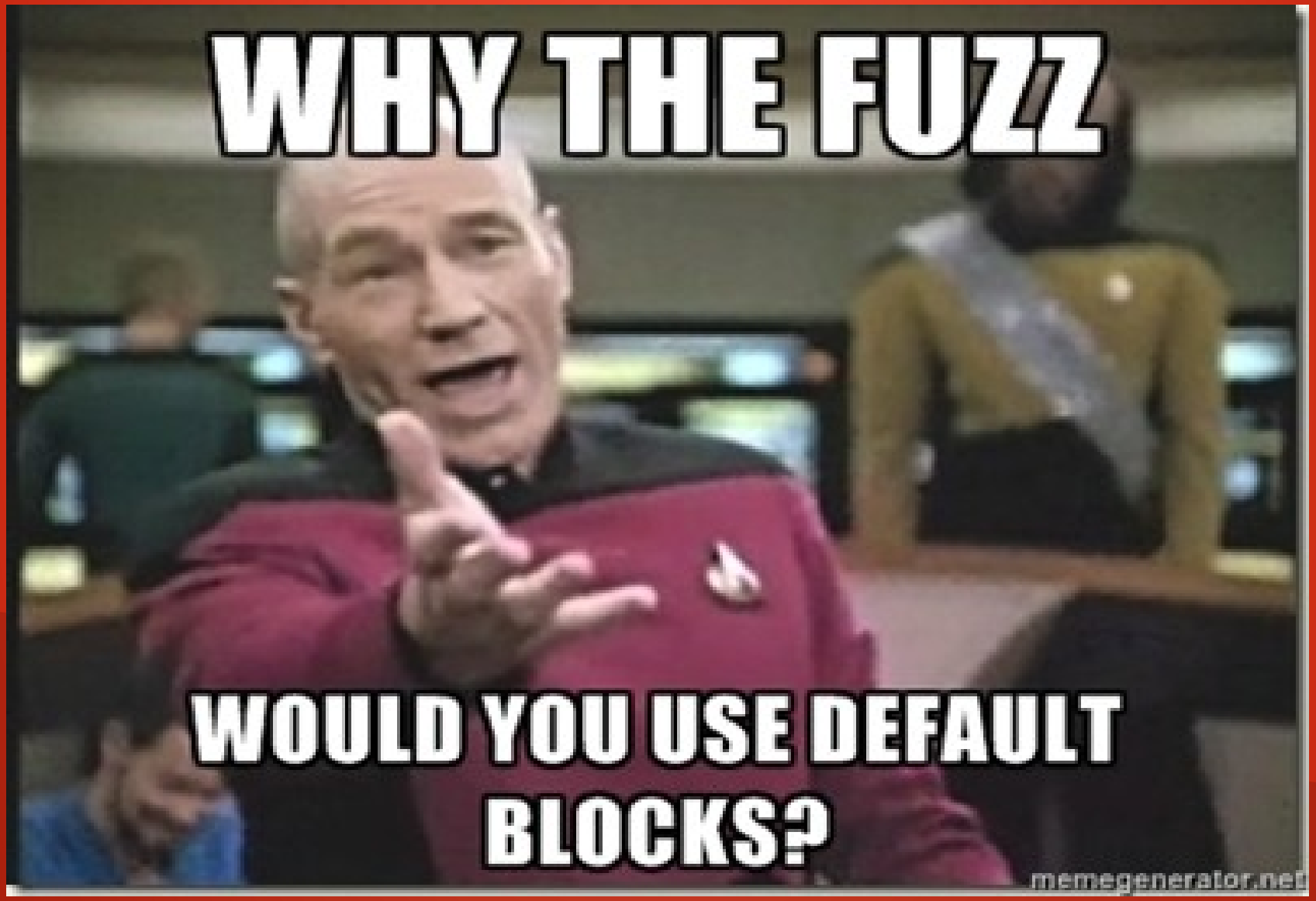
Default Block

- Your first Ruby magic
- Invisible parameter passed to a method
- The anonymous block
- Uses the keyword “yield” to...yield the block content

```
def thrice
    puts “Neoneun”
    yield
    yield
    yield
end

thrice do
    puts “Shoot!”
end
```

Default Block



Default Block

- Metaprogramming
- You allow functions to act as if they were keywords in Ruby
- Allows you to “define” your own “keywords”
- Lets the language grow
- Lets the syntax for methods approach the syntax for keywords
- Rails and other Ruby gems uses code blocks extensively

Default Block Parameters

- You may pass parameters to a block by providing it inside | |

```
def twice
      yield 0
      yield 1
end

twice do |i|
      puts “#{i + 1} Mississippi”
end
```

Blocks and Iterators

- Execute blocks when iterating through a collection

```
names = %w{ Seohyun Taeyeon Yuri }

names.each do |name|
    puts name
end
```

Classes

- Objects that contain different properties/attributes and methods
- User defined

```
class Book
end

book1 = Book.new
book2  = Book.new
```

Constructors

- Define an initialize method
- Instance variables
  - Variables instantiated for an instance of a class
  - Starts with “@”
  - Accessed only within the class

```
class Book
	def initialize(title)
		@title = title
	end
end

book1 = Book.new(“Fifty shades of gray”)
puts book1
```

Accessor Methods

- Getter/Setters
  - Allows getting and setting values for instance variables
  - Declare a getter setter method for each instance variable

```
class Book
	def initialize(title)
		@title = title
	end

	def title
		@title
	end
end

book1 = Book.new("Fifty shades of gray")
puts book1.title
```

attr_reader

- Allows you to use symbols to read instance variables from a class

```
class Book
	attr_reader :title

	def initialize(title)
		@title = title
	end

	def title=(title)
		@title = title
	end
end

book1 = Book.new("Fifty shades of gray")
puts book1.title
```

attr_accessor

- Allows you to use symbols to read and write instance variables from a class

```
class Book
	attr_accessor :title

	def initialize(title)
		@title = title
	end
end

book1 = Book.new("Fifty shades of gray")
puts book1.title
```

Exercise: Credit Card Approval

- Download the files crx.data and crx.names
  - Crx.data: credit card data in a text file
  - Crx.names: name of the attributes
- Create a class to represent a record of each instance of credit card approval record
- Remove the last three records and save it to a separate text file without the last column (+/-)
- Perform KNN to predict the status of the three other instances