Ruby Programming for Beginners

Programming

- Operating system - talk to a computer's hardware
    - e.g., Windows, Mac OS X, Linux
- Software
    - Programs and apps
    - Sends input to operating system
    - Receives output from operating system
- Programming language
    - Code used to create software
    - E.g., Ruby, Java, C++

Terms to Know

- Framework, library
  - Collection of reusable code
  - Contains common features
  - Lets you write an app faster

Ruby - language

Ruby gems - libraries

Ruby on Rails - framework

Ruby versus Ruby on Rails

- Rails is written in the Ruby language
- Rails contains many Ruby gems
- Rails is a framework
- Rails is used to build web apps

Ruby Philosophy

I believe people want to express themselves when they program.

They don't want to fight with the language.

Programming languages must feel natural to programmers.

I tried to make people enjoy programming and concentrate on the fun and creative part of programming when they use Ruby.

-- Yukihiro "Matz" Matsumoto, Ruby creator

Basic Programming Structures

- Variables - labels that hold information
- Types of information - text, numbers, collections
- Methods and operators - do stuff with variables
    - e.g., `+` for addition, `puts` to print output, `reverse` to reverse text
- Loops - do the same action several times
- Printing - display something on the screen (use `puts`), or save it to a file

Let's start coding!

Open your terminal

- A.k.a. "command line", "shell", "prompt"
- In Windows: `git bash`

  Git Bash
- Mac OS X and Ubuntu Linux: `Terminal`

Prompt

- Terminals show a line of text when:
  - You log in
  - A command finishes
- Called the "prompt"
  - Ends with a dollar sign
- When instructions start with "`$ `" or "`> `", just type the rest of the line into the terminal
- Give the terminal a command:

  ```
  $ irb
  ```

  Then hit Enter

irb: Interactive Ruby

- IRB has its own prompt that ends with `>`:

```
$ irb
>
```

- Enter the `exit` command to return to the terminal:

```
> exit
$
```

- Windows users: can't use backspace, delete, or arrow keys in IRB? Try:

```
$ irb --noreadline
```

Variables

- A variable holds information
- We give it a name so we can refer to it
- The info it holds can be changed

```
$ irb
> my_variable = 5.3
=> 5.3
> another_variable = "hi"
=> "hi"
> my_variable = "good morning"
=> "good morning"
```

Variable Assignment

- Assignment - store a value in a variable
  - Done with equals sign:

```
> variable = "my value"
```

- Right side of equals sign is evaluated, then value stuck into left side

```
> sum = 5 + 3
=> 8
> sum = sum + 2
=> 10
```

Naming Variables

- Can't name it just anything
- Choose from:
    - All letters -- `folders = [true, false]`
    - Letters, then numbers -- `data2 = 2.0`
    - Letters and underscores -- `first_variable = 1`
    - All of the above -- `some_value1 = 'morning'`
- Try out:
    - Hyphens -- `bad-name = 2`
    - Starting with a number -- `3var = 'something'`
    - Only numbers -- `123 = "abc"`

Common Types of Information

- String (text)
  - Like name, email, address fields from InstallFest
- Number
  - Like temperature field from InstallFest
- Collection
  - Arrays and hashes
- Boolean (`true`, `false`)

Strings

- A string is text
- Wrap text in a pair of quotation marks:
    - `> 'Single quotes are fine'`
    - `> "So are double quotes"`
- Don't mix and match the quotes:
    - `> "But you have to'`
    - `> 'match the quotes"`

String Exercise

1. Create string variables `first_name`, `last_name`, and `favorite_color`
   - Set the values to whatever you like
2. Create the sentence "Hi, my name is *(first_name) (last_name)* and my favorite color is *(favorite_color)*"

Hint: use the + operator to combine strings:

```
"string 1" + "string 2"
```

Numbers

- Integers - numbers without decimal points, e.g., 5, -2
- Floats - numbers with decimal points, e.g., 3.14, -0.123
- Operators you can use with numbers: `+, -, *, /`

Number Exercise 1

How are these results different?

- Divide an integer by an integer
  - e.g., `5 / 2`
- Divide an integer by a float
  - e.g., `5 / 2.0`

Number Exercise 2

1. Assign your two favorite numbers to two variables, `num1` and `num2`
2. Compute the sum (`+`), difference (`-`), quotient (`/`), and product (`*`) of `num1` and `num2`
3. Assign these values to variables `sum`, `difference`, `quotient`, and `product`

Collections

- Array, a.k.a. list
  - Collection of values
  - `> [1, 3, 5, 7]`
  - `> ["hi", "there", 'folks']`
- Hash, a.k.a. dictionary, map, associative array
  - Collection of keys and values
  - `> {1 => 'one', 2 => 'two'}`
  - `> {'this' => 'that', "who" => 2.5}`

Array Indexing

- Items in an array are stored in the order they were added
- Access an item via its index
- Ruby starts counting at 0, not 1

```
> fruits = ['kiwi', 'strawberry', 'plum']
=> ['kiwi', 'strawberry', 'plum']
> fruits[0]
=> 'kiwi'
> fruits[2]
=> 'plum'
```

Array Exercise

1. Create an array variable called `grocery_list`
2. Include at least five items from your grocery list in the array
    - Use strings to represent groceries

Array Index Exercise

- Using your `grocery_list` array:
  - What do you think will be at index 0?
  - What about index 5?
  - Access index 0
  - Access index 5

Remember: Ruby starts counting at 0, not 1

Hashes

- Refer to values by keys, not indices
- Each member of a hash is a pair:
  key => value

```
> en_to_es = {'one' => 'uno', 'two' =>
'dos', 'three' => 'tres'}
=> {"one"=>"uno", "two"=>"dos", "three"=>"
tres"}
> en_to_es['one']
=> "uno"
```

Booleans

- `true` and `false`
- Some code evaluates to `true` or `false`:
  - Numeric comparison:

```
> 1 < 2                 > 1 == 1
=> true                 => true
> 2 >= 5                > 18.0 != 18.0
=> false                => false
```

  - String equality:

```
> "yellow" == "blue"
=> false
> "yellow" != "blue"
=> true
```

Boolean Exercise

1. Assign your favorite color to a variable named `favorite_color`
2. Assign a different color to a variable named `not_favorite_color`
3. Test to see if these variables are equal

Methods

- *If objects (like strings, integers, and floats) are the nouns in the Ruby language, then methods are like the verbs.*
  -- Chris Pine's "Learn to Program"
- Do stuff to values
- Call a method on a value using dot "."

```
> 'hello there'.reverse
=> "ereht olleh"
```

Method Exercise

1. Create a string variable called `old_string` and assign it the value `"Ruby is cool"`
2. Use string methods to modify `old_string` so that it is now `"LOOC SI YBUR"`
3. Assign this to another variable called `new_string`

Hint: look at the string methods `upcase` and `reverse`

Loops

- Used to do something repeatedly
- Useful with arrays and hashes

```
> cities = ['Lexington', 'Louisville',
'Indianapolis']
> cities.each do |city|
?> puts "I live in " + city
> end
I live in Lexington
I live in Louisville
I live in Indianapolis
=> ["Lexington", "Louisville", "Indianapolis"]
```

Loop Exercise

1. Create an array of four places you would like to visit
2. Print out each of these places using a loop

Example output:

```
"I would like to visit Barcelona"
"I would like to visit Ireland"
"I would like to visit Alaska"
"I would like to visit New Orleans"
```

Hint: use the `each` method on your array

Summary

- Programming in the Ruby language with the Ruby on Rails framework
- Try out Ruby code in IRB
- Use variables to label data and manipulate it
- Data types: strings, integers, floats, arrays, hashes, and Booleans
- Manipulate variables with methods and operators
- Use loops to do something repeatedly, maybe looping over an array or hash

Questions?