Ruby Language Basics I

Sang Shin
JPassion.com
“Code with Passion!”

Topics

- What is Ruby?
- Ruby naming convention
- Interactive Ruby (IRB)
- Ruby object
- Ruby types
  - String, Hash, Symbol
- Ruby class
- Inheritance
- 3 ways of creating a Ruby object

What is Ruby?

Ruby is...

A dynamic, open source programming language with a focus on **simplicity** and **productivity**. It has an elegant syntax that is natural to read and easy to write.

http://www.ruby-lang.org

Ruby as a Programming Language

- Dynamically typed
- Optimized for people
  - > Easy to read and write
  - > Powerful
  - > Fun
- Everything is an object
  - > There are no primitives - Java language has primitives which are not objects

Ruby Language History

- Created 1993 by Yukihiro “Matz” Matsumoto
  - “More powerful than Perl and more OO than Python”
- Ruby 2.1.2 is current (as of July 2014)

Ruby Naming Conventions

Ruby Naming Conventions

- Ruby file - *.rb* suffix
  - *myprog.rb*
- Class & Module names – MixedCase
  - *MyClass*
- methods - lower case with underscores
  - *my_own_method*
- local variables – lower case with underscores (same as methods)
  - *my_own_variable*
  - One reason why should use lower case is that if you Capitalize the first letter, Ruby will define a constant instead of a variable
    - *number = 1   # regular variable*
    - *Pi = 3.14159 # constant*

Ruby Naming Conventions

- Instance variables - @ prefix to variable name
  - *@my_instance_variable*
- Class variables – @@ prefix to variable name
  - *@@my_class_variable*
- Global variables - $ prefix to variable name (rarely used)
  - *$my_global_variable*
- Constants
  - *UPPER_CASE*

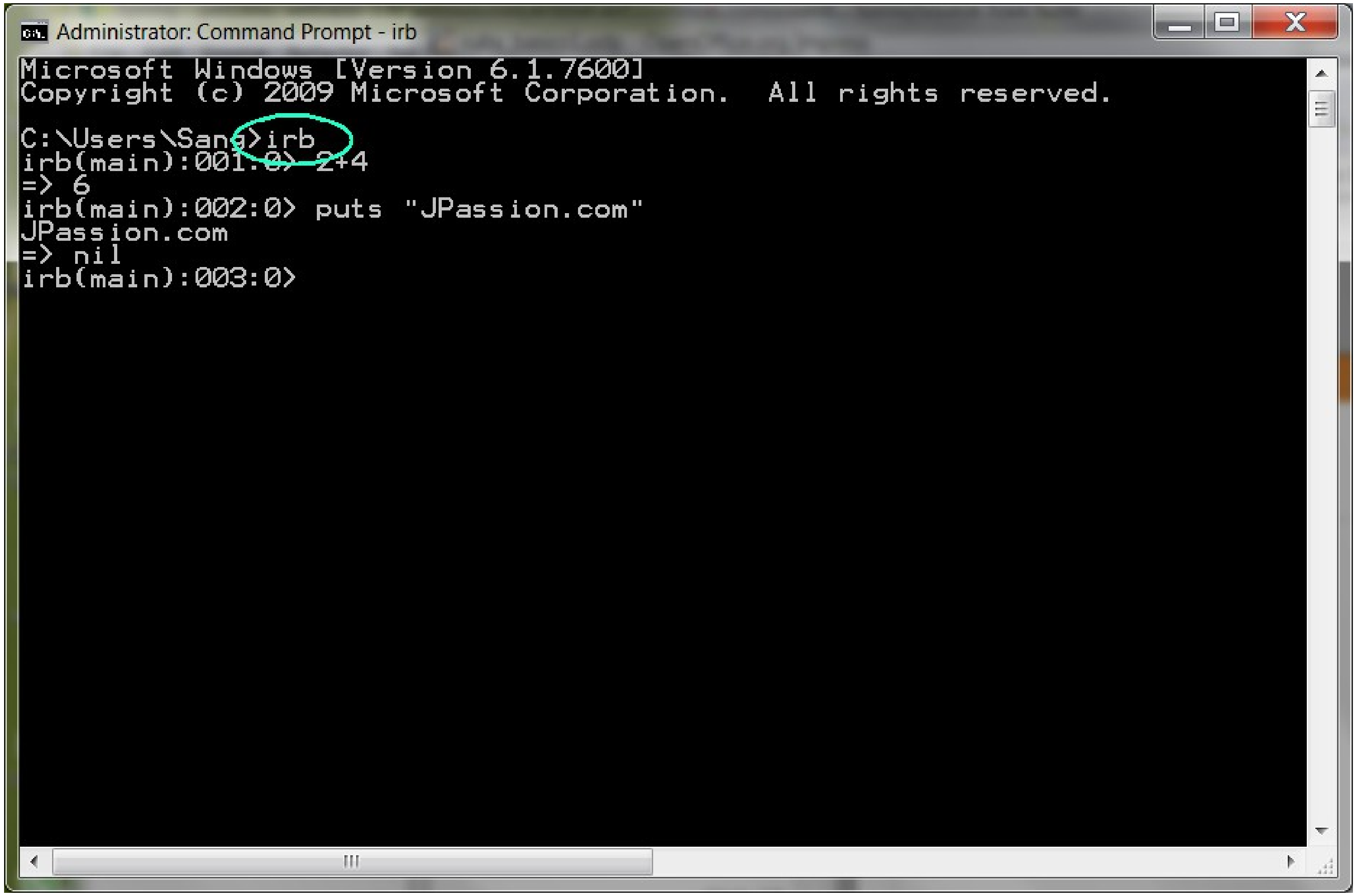
IRB (Interactive Ruby)

IRB (Interactive Ruby)

- When learning Ruby, you will often want to experiment with new features by writing short snippets of code. Instead of writing a lot of small text files, you can use *irb*, which is Ruby's interactive mode.
- You can use *irb* at the command line

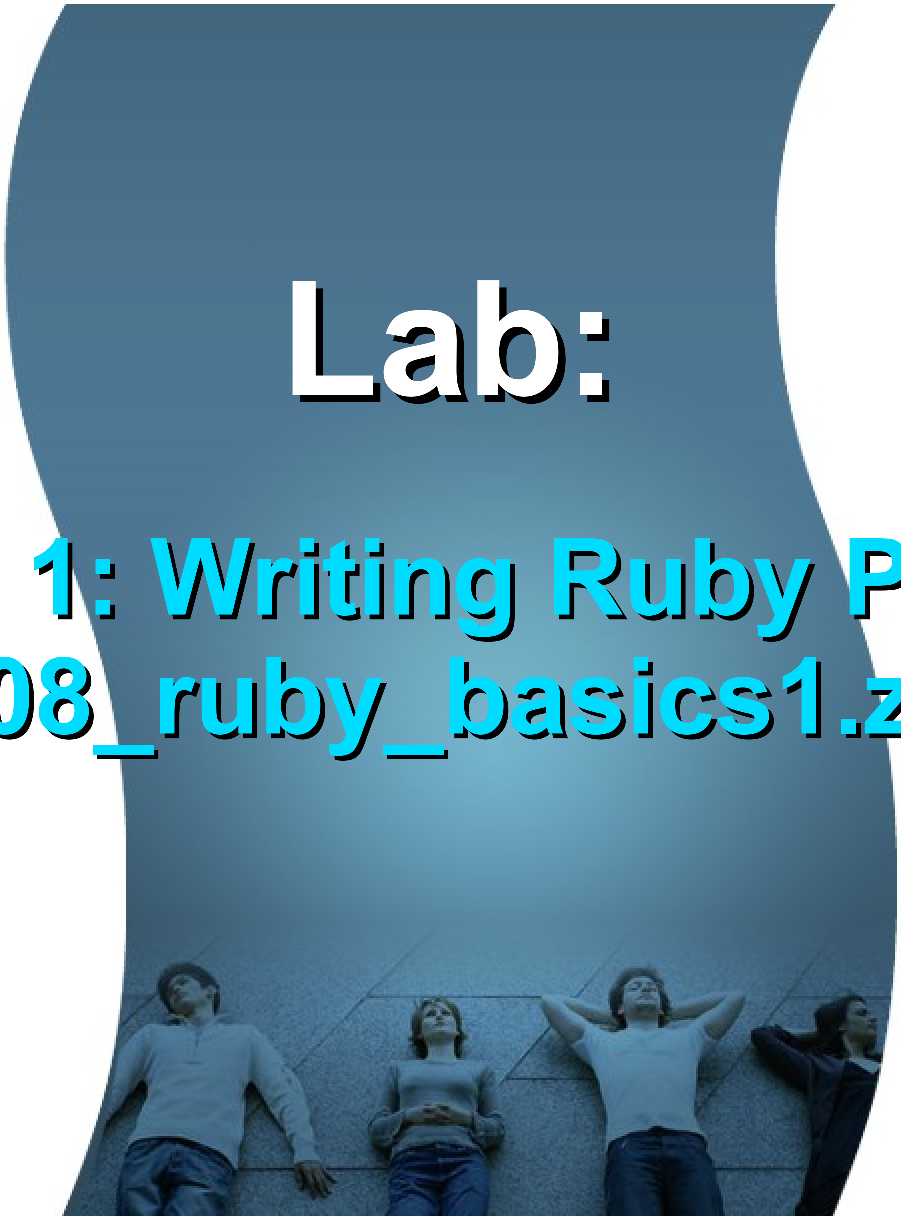
```
$ irb --simple-prompt
>> 2+2
=> 4
>> 5*5*5
=> 125
>> quit
```

IRB



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sang>irb
irb(main):001:0> 2+4
=> 6
irb(main):002:0> puts "JPassion.com"
JPassion.com
=> nil
irb(main):003:0>
```



Lab:

Exercise 1: Writing Ruby Programs
5508_ruby_basics1.zip

Ruby Object

In Ruby, Everything is an Object

- Like Smalltalk, Ruby is a pure object-oriented language — everything is an object
- In contrast, languages such as C++ and Java are hybrid languages that divide the world between objects and primitive types
  - > The hybrid approach results in better performance for some applications, but the pure object-oriented approach is more consistent and simpler to use

What is an Object?

- Using Smalltalk terminology, an object can do exactly three things.
  - Hold state, including references to other objects.
  - Receive a message, from both itself and other objects.
  - In the course of processing a message, send messages, both to itself and to other objects.
- If you don't come from Smalltalk background, it might make more sense to rephrase these rules as follows:
  - An object can contain data, including references to other objects.
  - An object can contain methods, which are functions that have special access to the object's data.
  - An object's methods can call/run other methods/functions.

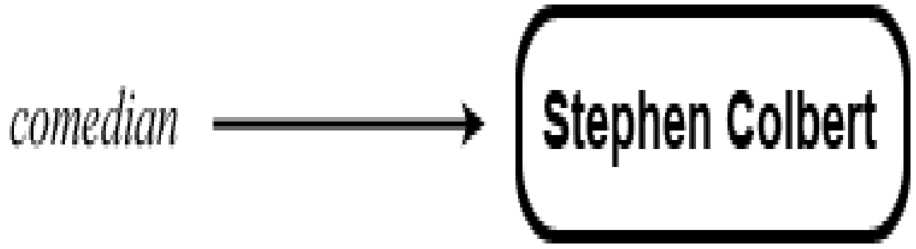
In Ruby, Everything Is An Object

- 'Primitives' are objects
  - > *-1.abs   # Invoke abs() method of an object*
- nil is an object
  - > *nil.methods # Invoke methods() method of an object – display all methods available from nil object*
- Classes are objects
  - > *Song.new* – invoking the "*new*" class method on "*Song*" class - Create instances of themselves
- Code blocks can be converted into objects (Proc's)
  - > They can be pass around, even as parameters or return value
  - > Basis for enabling closure

Variables and Objects

- Create a String object containing the text "Stephen Colbert". We also told Ruby to use the variable *comedian* to refer to this object. (Works the same as in Java)

```
>> comedian = "Stephen Colbert"
=> "Stephen Colbert"
```



Ruby Types

Ruby Types

- String
- Number
- Symbol
  - New concept if you are coming from Java background
- Array
- Hash

Ruby Types:
Strings

String Literals

- One way to create a String is to use single or double quotes inside a Ruby program to create what is called a string literal

```
puts 'Hello world'
puts "Hello world"
```

- Double quotes allow you to embed variables or Ruby code inside of a string literal – this is commonly referred to as interpolation

```
def my_method(name)
    puts "Your name is #{name}"
end
```

String Literals with Interpolation

- Notation
  - #{expression}
- Expression can be an arbitrary Ruby expression
- If variable that is referenced by #{expression} is not available (has not been assigned), a NameError exception will be raised:

  "trying to print #{undefined} variable"

  NameError: undefined local variable or method `undefined' for main:Object

Escape Sequences

- \" – double quote
- \\ – single backslash
- \a – bell/alert
- \b – backspace
- \r – carriage return
- \n – newline
- \s – space
- \t – tab

Escape Sequences

```
puts "Hello\t\tworld"
puts "Hello\b\b\b\b\bGoodbye world"
puts "Hello\rStart over world"
puts "1. Hello\n2. World”
```

puts and print

- *puts* automatically prints out a newline after the text

```
>> puts "Say", "hello"
Say
hello
```

- *print* function only prints out a newline if you specify one

```
>> print "Say", "hello", "\n"
Sayhello
```

% Notation

- *%w* causes breaks in white space to result in a string array
  - %w(a b c)
  - => ["a", "b", "c"]

Ruby Types:
Symbols

What is Symbol?

- A Ruby symbol is the internal representation of a name
- It is a class in Ruby language

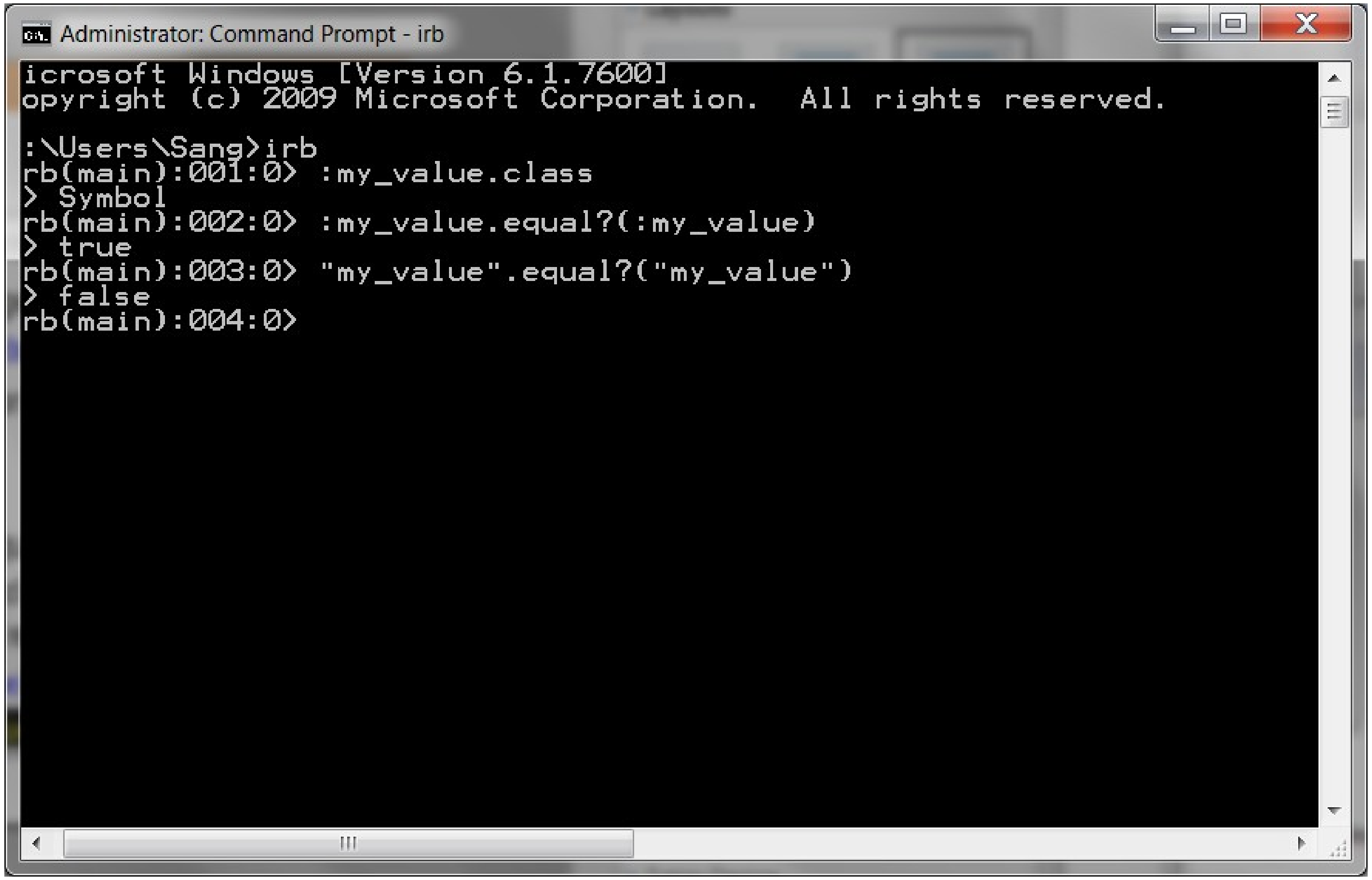
  :my_value.class   #=> Symbol
- You construct the symbol for a name by preceding the name with a colon.

  :my_symbol
- Atomic, immutable and unique
  - Can't be parsed or modified
  - All references to a symbol refer to the same object

    :my_value.equal?(:my_value)   #=> true

    "my_value".equal?("my_value") #=> false

Symbols



```
icrosoft Windows [Version 6.1.7600]
opyright (c) 2009 Microsoft Corporation.  All rights reserved.

:\Users\Sang>irb
rb(main):001:0> :my_value.class
> Symbol
rb(main):002:0> :my_value.equal?(:my_value)
> true
rb(main):003:0> "my_value".equal?("my_value")
> false
rb(main):004:0>
```

Symbols vs. Strings

- Symbols are always interchangeable with strings
  - In any place you use a string in your Ruby code, you can use a symbol
- Important reasons to use a symbol over a string
  - If you are repeating same string many times in your Ruby code, let's say 10000 times, it will take 10000 times of memory space of the string while if you are using a symbol, it will take a space for a single symbol
- Minor reasons to use a symbol over a string
  - Symbol is easier to type than string (no quotes to type)
  - Symbol stands out in the editor
  - The different syntax can distinguish keys from values in hash

```
:name => 'Sang Shin'
```

Ruby Types:
Hash

Hash

- Hashes are basically key-value pairs (like a Map in Java)
  - Each key should be unique
- A hash object is created by writing *Hash.new* or by writing an optional list of comma-separated key => value pairs ("=>" is called Rocket) inside curly braces (most common form of creating a hash)

```
hash_one   = Hash.new # Similar to Java
hash_two   = {}                # Shorthand for Hash.new
hash_three = {"a" => 1, "b" => 2, "c" => 3} # Most common form
```

Hash and Symbol

- Instead of

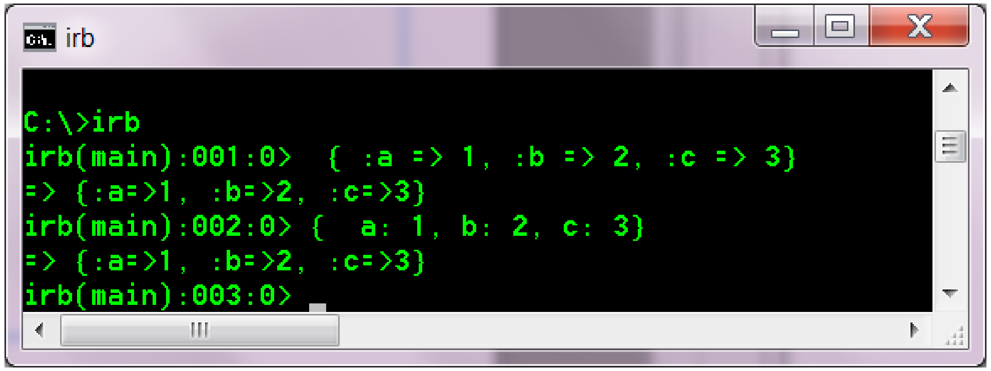
  hash_sym = { "a" => 1, "b"=> 2, "c" => 3}

- Usually Symbols (instead of Strings) are used as Hash keys, so you will see hashes declared like following:

  hash_sym = { :a => 1, :b => 2, :c => 3}

- From Ruby 1.9, a simplified notation for =>

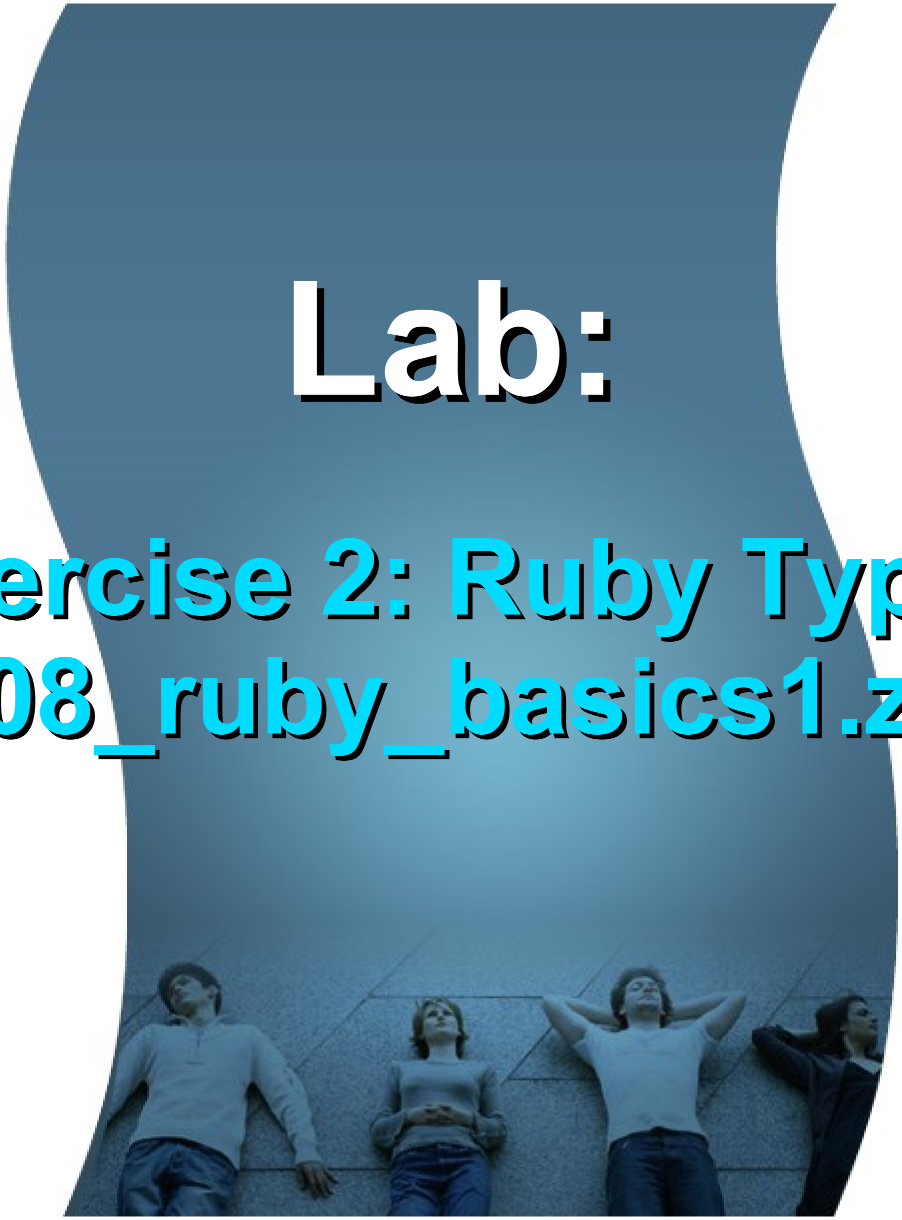
  hash_sym = { a: 1, b: 2, c: 3}


irb

```
C:\>irb
irb(main):001:0>  { :a => 1, :b => 2, :c => 3}
=> {:a=>1, :b=>2, :c=>3}
irb(main):002:0> {  a: 1, b: 2, c: 3}
=> {:a=>1, :b=>2, :c=>3}
irb(main):003:0> _
```

Where Do Symbols Typically Used?

- Symbols are often used as
  - Hash keys (:name => 'Brian', :hobby => 'golf')
  - Arguments of a method (:name, :title)
  - Method names (:post_comment)
- Symbols are used in Rails pervasively



Lab:

Exercise 2: Ruby Types 5508_ruby_basics1.zip

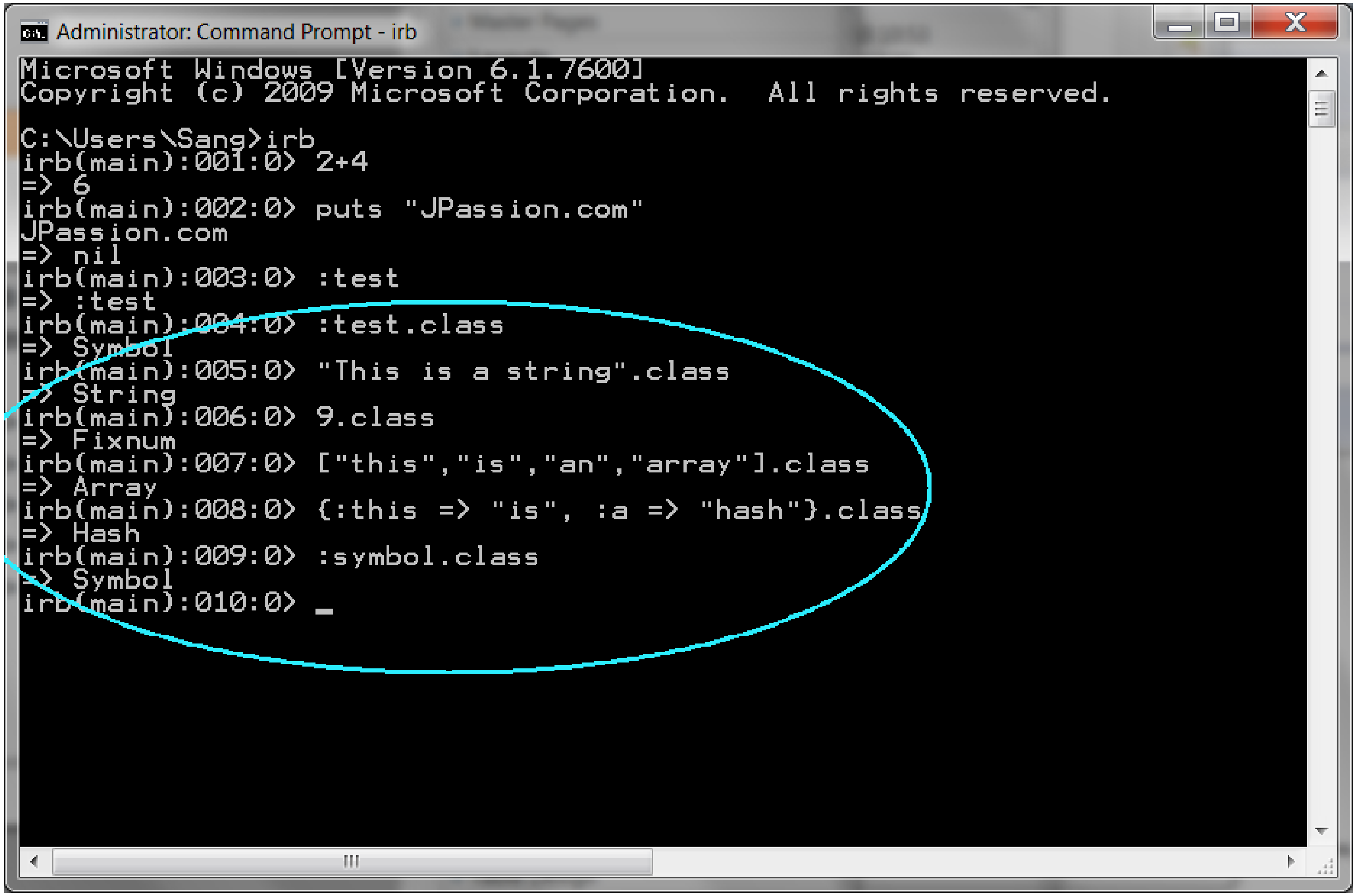
Ruby Class and Objects

Ruby Classes

- Every object in Ruby is created from a class. To find the class of an object, simply call that object's *class* method.

```
"This is a string".class                  #=> String
9.class                                   #=> Fixnum
["this","is","an","array"].class          #=> Array
{:this => "is", :a => "hash"}.class       #=> Hash
{this: "is", a: "hash"}.class             #=> Hash (from Ruby 1.9)
:symbol.class                             #=> Symbol
```

Ruby Classes



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sang>irb
irb(main):001:0> 2+4
=> 6
irb(main):002:0> puts "JPassion.com"
JPassion.com
=> nil
irb(main):003:0> :test
=> :test
irb(main):004:0> :test.class
=> Symbol
irb(main):005:0> "This is a string".class
=> String
irb(main):006:0> 9.class
=> Fixnum
irb(main):007:0> ["this","is","an","array"].class
=> Array
irb(main):008:0> {:this => "is", :a => "hash"}.class
=> Hash
irb(main):009:0> :symbol.class
=> Symbol
irb(main):010:0> _
```

Defining a Class

- Use *class* keyword

```
# Define Chocolate class
class Chocolate
    def eat
        puts "That tasted great!"
    end
end
```

Instantiation of an Object

- An object instance is created from a class through the a process called instantiation (like in Java)
- In Ruby, this takes place through a Class method *new* (similar to Java)

```
an_object = MyClass.new(parameters)
```

- The above sets up the object in memory and then delegates control to the *initialize* function of the *MyClass* class if it is present. Parameters passed to the new function are passed into the *initialize* function.

```
class MyClass
  def initialize(parameters)
  end
end
```

Class Example

- Simple *RocketShip* Class – We will study this code in the following slides

```
class RocketShip  < Object
  attr_accessor :destination

  def initialize(destination)
    @destination = destination
  end

  def launch()
    "3, 2, 1 Blast off!"
  end
end
```

Class Example

- Single Inheritance

```
class RocketShip  < Object  #  < Object is optional
                            # like in Java
  attr_accessor :destination

  def initialize(destination)
    @destination = destination
  end

  def launch()
    "3, 2, 1 Blast off!"
  end
end
```

Class Example

- Constructors in Ruby are named *initialize*

```
class RocketShip  < Object
  attr_accessor :destination

  def initialize(destination)
    @destination = destination
  end

  def launch()
    "3, 2, 1 Blast off!"
  end
end

# new() allocates a RocketShip instance and initialize()
# is called for initializing that instance
r = RocketShip.new('Netptune')
```

Class Example

- Attributes are easily defined with *attr_accessor*

```
class RocketShip  < Object
  #  No need to define getter and setter for an attribute
  attr_accessor :destination

  def initialize(destination)
    @destination = destination
  end

  def launch()
    "3, 2, 1 Blast off!"
  end
end

r = RocketShip.new
r.destination = 'Saturn'  # Set the attribute with value
```

Ruby Class: Inheritance

Inheritance

- A class can inherit functionality and variables from a super class, sometimes referred to as a parent class or base class. (Like in Java)
- Ruby does not support multiple inheritance and so a class in Ruby can have only one super class. (Like in Java)
- All non-private variables and functions are inherited by the child class from the super class. (Like in Java)

Overriding a method

- If your class overrides a method from parent class (super class), you still can access the parent's method by using 'super' keyword

```
class ParentClass    # This is a parent classs
  def a_method       # This parent class has a_method method
    puts 'b'
  end
end

class ChildClass < ParentClass  # This is a child class
  def a_method                  # and it overrides a_method method
    super                       # Call a_method of a parent class
    puts 'a'
  end
end

instance = ChildClass.new
instance.a_method
```

3 Ways of Creating Ruby Object

Ruby Object Can Be Created in 3 Different Ways

- Constructor parameters in a hash

```
user = User.new(:name => "Shin", :occupation => "Daydreamer")
user = User.new :name => "Shin", :occupation => "Daydreamer"
```

- Create a bare object and then set attributes

```
user = User.new
user.name = "Shin"
user.occupation = "Daydreamer"
```

- Use block initialization (we will learn about block later on)

```
user = User.new do |u|
  u.name = "Shin"
  u.occupation = "Daydreamer"
end
```

Lab:

Exercise 3: Ruby Class & Objects
5508_ruby_basics1.zip

Code with Passion!
JPassion.com