Objectives:

Work within an interactive ruby shell (IRB)

Learn the basics of Ruby

Start irb on a target node

1. Execute:

```
$/opt/chef-workstation/embedded/bin/irb
```

Expected output:

```
irb(main):001:0>
```

1. Variable assignment.
    - Variables are assigned with an equals sign ("=") .
    - The REPL prints the return value of the last statement, with a => in front.
    - "**Puts**" is the ruby equivalent of "print" or "echo" in other languages - it automatically includes a newline.
    - "**pp**" (pretty print) also prints objects in a more visually organized way
    - **NOTE, "puts" is a function with a return value and puts returns nil**

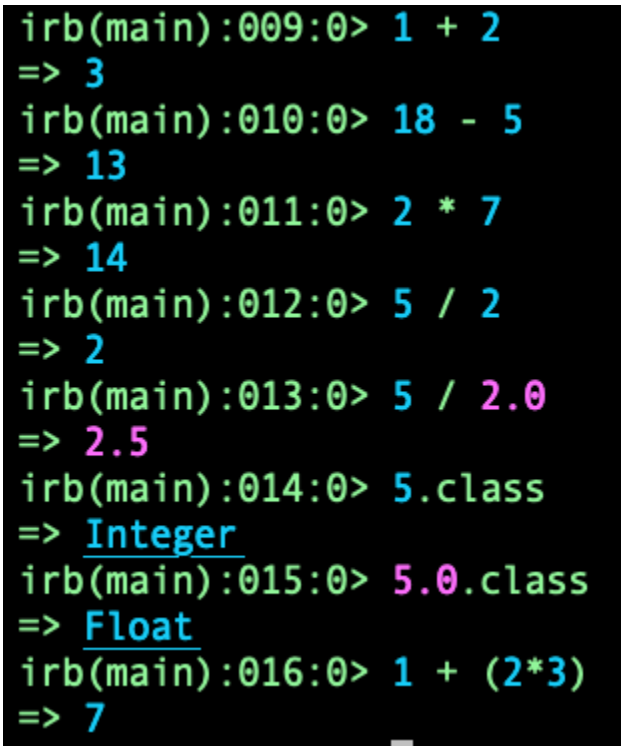
2. **Execute in irb as follows:**

```
irb(main):001:0> x= "Hello"
=> "Hello"
irb(main):002:0> puts x
Hello
=> nil
irb(main):003:0>
```

3. List Methods by executing: pp x.methods:

Arithmetic, subtraction, multiplication, division, decimal places, order of operations:

4. Follow each operation as depicted below.



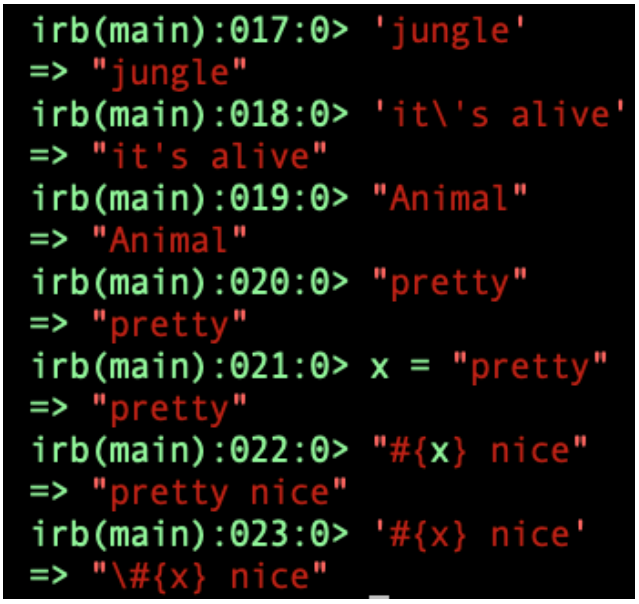
Key Points:

- Plus is addition and numbers are unquoted.
- Addition with quotes are treated as a concatenation and are returned as such:

  "1" + "2" = "12"
- * is multiplication.
- Division using whole numbers means no decimal points. For example 5 / 2 returns => 2.
- Division on floating point numbers means decimal points.
- Ruby has dots to call methods and everything in ruby is an object including a number. We can find out what **class** an object is by calling ".class".
- Expressions can be grouped with parenthesis like 1 + (2*3).

Strings:

5. Use strings in irb as shown below:



   - Strings in ruby can use single quotes.
   - REPL prints results in double quotes to explicitly state that the return value is a string.
   - A forward slash escapes from the quote delimiter.
     - REPL gives a double quoted version, so there is no escape character.
   - Strings in ruby can also use double quotes.
   - Interpolation: the value of an expression can use a variable with #{}, as depicted in the example above.

     Note that strings with single quotes can't use interpolated variables.

6. **Use the operators as shown:**

```
irb(main):030:0> 1 == 1
=> true
irb(main):031:0> 1 == true
=> false
irb(main):032:0> 1 != true
=> true
irb(main):033:0> !!1 == true
=> true
```

- == tests for equality.
- 1 evaluates to true *when pressed* - but it is not true.

```
irb(main):034:0> 2 < 1
=> false
irb(main):035:0> 2 > 1
=> true
irb(main):036:0> 4 >= 3
=> true
irb(main):037:0> 4 >= 4
=> true
irb(main):038:0> 4 <= 5
=> true
irb(main):039:0> 4 <= 3
=> false
```

- Above examples depict Greater than, less than, greater than or equal to, less than or equal to.

```
irb(main):042:0> 5 <=> 5
=> 0
irb(main):043:0> 5 <=> 6
=> -1
irb(main):044:0> 5 <=> 4
=> 1
```

7. The spaceship operator returns
    - 0 if it is equal
    - -1 if it is greater
    - and 1 if it is less than.
    - **Note this is very useful for sorting!**

8. Special Iterations:

```
irb(main):045:0> 3.times do puts "3 times" end
3 times
3 times
3 times
=> 3
```

Arrays:

```
irb(main):046:0> x = ["a", "b", "c"]
=> ["a", "b", "c"]
irb(main):047:0> x[0]
=> "a"
irb(main):048:0> x.first
=> "a"
irb(main):049:0> x.last
=> "c"
```

Arrays have methods just like everything else - first and last are two methods that apply specifically to arrays.

9. Continue manipulating arrays in irb:

```
irb(main):050:0> x + ["d"]
=> ["a", "b", "c", "d"]
irb(main):051:0> x
=> ["a", "b", "c"]
irb(main):052:0> x = x + ["d"]
=> ["a", "b", "c", "d"]
irb(main):053:0> x
=> ["a", "b", "c", "d"]
```

   - Adding the element didn't mutate the original array. 'x' still has its original value.
   - The operation was not destructive to the original object.
   - In order to destroy the original value, a new value needs to be assigned to 'x'.

```
irb(main):054:0> x << "e"
=> ["a", "b", "c", "d", "e"]
irb(main):055:0> x
=> ["a", "b", "c", "d", "e"]
```

   - The "<<" append operator will add an item to an array!
   - It also mutates it! - this is more idiomatic than using the + operator
     - **Note: there is no "prepend operator", but there are functions that allow you to prepend.**

```
irb(main):056:0> x.map { |i| "the letter #{i}" }
=> ["the letter a", "the letter b", "the letter c", "the letter d", "the letter e
irb(main):057:0> x
=> ["a", "b", "c", "d", "e"]
```

   - Ruby is full of iterators - like "each" from earlier. Map iterates through the array and takes a block, whose return value becomes the value of an item in a new array.
   - Note: the original value of x is unchanged!

```
irb(main):058:0> x.map! { |i| "the letter #{i}" }
=> ["the letter a", "the letter b", "the letter c", "the letter d", "the letter e"]
irb(main):059:0> x
=> ["the letter a", "the letter b", "the letter c", "the letter d", "the letter e"]
```

A ! on the end of the map means "be destructive to the object I am operating on". This results in a mutated x.

10. Hashes :

```
irb(main):060:1* h = {
irb(main):061:1*   "first_name" => "Gary",
irb(main):062:1*   "last_name" => "Gygax"
irb(main):063:0> }
=> {"first_name"=>"Gary", "last_name"=>"Gygax"}
```

A key value pair - like a dict in python, or a hash in perl.

```
=> {"first_name"=>"Gary", "last_name"=>"Gygax"}
irb(main):064:0> h.keys
=> ["first_name", "last_name"]
irb(main):065:0> h["first_name"]
=> "Gary"
irb(main):066:1* h[
irb(main):067:0> "age"] = 33
=> 33
irb(main):068:0> h.keys
=> ["first_name", "last_name", "age"]
```

Items can be added to the hash by putting the key in “square brackets” “[]”, and using assignment on that key name.

```
irb(main):069:0> h.values
=> ["Gary", "Gygax", 33]
```

All of the key values in the hash are returned with .values

```
irb(main):070:0> h.each { |k, v| puts "#{k}: #{v}" }
first_name: Gary
last_name: Gygax
age: 33
=> {"first_name"=>"Gary", "last_name"=>"Gygax", "age"=>33}
```

11. A hash can be iterated over with the .each method - here it is printing out the keys and values.

12. Conditionals

```
irb(main):071:0> x = "happy"
=> "happy"
irb(main):072:1* if x == "happy"
irb(main):073:1*   puts "Sure am!"
irb(main):074:1* elsif x == "sad"
irb(main):075:1*   puts "Boo!"
irb(main):076:1* else
irb(main):077:1*   puts "Therapy?"
irb(main):078:0> end
Sure am!
=> nil
```

If/elseif/else is a commonly used conditional.

What is uncommon - this construct has a return value of the return value of the last expression in the leg that matches.

```
irb(main):079:1* case x
irb(main):080:1* when "happy"
irb(main):081:1*   puts "Sure Am!"
irb(main):082:1*   1
irb(main):083:1* when "sad"
irb(main):084:1*   puts "Boo!"
irb(main):085:1*   2
irb(main):086:1* else
irb(main):087:1*   puts "Therapy?"
irb(main):088:1*   3
irb(main):089:0> end
Sure Am!
=> 1
```

- Case is a shorter way to evaluate more than two conditionals on the same variable.
- Return values are added here to illustrate the point.
- Rule of thumb: If possibilities >=3, use if and elsif. If possibilities > 3, use a case conditional.

13. Method Definitions:

```
irb(main):090:1* def metal(str)
irb(main):091:1*   puts "!!#{str} is metal!!"
irb(main):092:0> end
=> :metal
irb(main):093:0> metal("ozzy")
!!ozzy is metal!!
=> nil
```

Methods - "def" defines a new method with a string input. The return code of the last expression is the return code of the method - in this case, the last expression is puts, and as we have seen a many times: puts returns nil.

14. Classes:

```
irb(main):094:1* class Person
irb(main):095:1*   attr_accessor :name, :is_metal
irb(main):096:2*   def metal
irb(main):097:3*     if @is_metal
irb(main):098:3*       puts "!!#{@name} is metal!!"
irb(main):099:2*     end
irb(main):100:1*   end
irb(main):101:0> end
=> :metal
```

- Ruby is object oriented - everything has a class.
- A class is a collection of properties with methods attached.
- attr_accessor creates two "getter/setter" (aka read/write) methods for a person - their name, and if they are metal. There also exists attr_reader & attr_writer
- The metal method is updated to use the @ variable. These are instance variables, they are different based on the object we create.

```
irb(main):102:0> p = Person.new
=> #<Person:0x0000000001827ef8>
irb(main):103:0> p.name = "Adam Jacob"
=> "Adam Jacob"
irb(main):104:0> p.is_metal = true
=> true
irb(main):105:0> p.metal
!!Adam Jacob is metal!!
=> nil
```

- Create a new instance of a class with ".new".
- Use the accessors with .name = ""

```
irb(main):104:0> p.is_metal = true
=> true
irb(main):105:0> p.metal
!!Adam Jacob is metal!!
=> nil
irb(main):106:0> p.is_metal = false
=> false
irb(main):107:0> p.metal
=> nil
```

15. Exit IRB

```
irb(main):108:0> exit
/home/ubuntu/chef-repo/cookbooks/apache $
```

Notify your instructor that you are done with the lab

END OF LAB