Collections & Blocks

Enumerable

Enumerable is by far the most important module to master in Ruby. What is a module? Modules are very similar to Classes. In fact a Module is a superclass of a Class:

```
Class.superclass # => Module
```

We use a module to group together a collection of variables and methods (very similar to a class). However, we should use a module if it is meant to be included within a class and we should use a class if we are going to instantiate something. What does it mean by including a module?

Printer Module

Let's create our own module and include it within the String class and the Array class. Go ahead and create a new directory called mod_prac and create a new file called printer_module.rb with the following contents:

```
module Printer
  def print_each_element
    for i in 0...self.length
      puts self[i]
    end
  end
end
```

Now let's create another Ruby file called main.rb with the following contents. We are including this module into the Array class and the String class so that its instances can be sent this message:

```
require_relative 'printer_module'
class Array
  include Printer
end
class String
  include Printer
end
[1,2,3].print_each_element
"hello".print_each_element
```

We could have used a Printer class and have Array and String inherit from it so that the methods are available, but it makes more sense to include a module because a Printer is not really related to an Array or a String.

Enumerable

Enumerable is just a module that is included in Arrays and Hashes. We can verify this by looking at the ancestors. Ancestors contain the list of Classes and Modules that a class inherits/includes.

```
Array.ancestors  # => [Array, Enumerable, Object, Kernel, BasicObject]
Hash.ancestors   # => [Hash, Enumerable, Object, Kernel, BasicObject]
```

Arrays and Hashes both have the Enumerable and the Kernel module included. This means that any methods declared in these two methods are available to the instances of the Array or Hash class. Arrays and Hashes also inherit from the Object class and the BasicObject class. This is what people mean when they say in Ruby everything is an object. If we look at the ancestor chain, it always ends with Object and BasicObject.

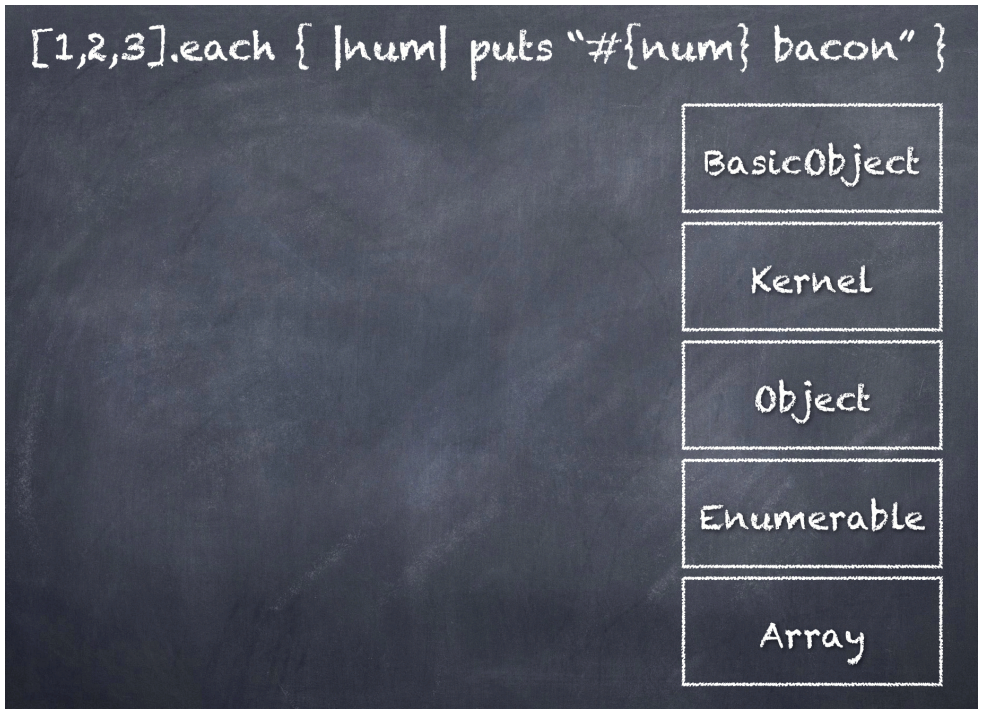
```
Fixnum.ancestors # => [Fixnum, Integer, Numeric, Comparable, Object, Kernel, BasicObject]
```

The Enumerable module has many methods that makes Ruby very fun to work with. The Enumerable module is equivalent to the underscore library for JavaScript except that it comes pre-baked with Ruby.

Method Lookup

When we send a message to an object, it first looks within its own class if that method exists. If not we move up our ancestor chain to see if that message exists. We keep going up until we either find the message that we know how to respond to or we reach the end of our ancestor chain and say no message (method) was found.

[1, 2, 3].each { |num| puts "#{num} bacon" }



{ hello: "goodbye" }.to_s

Challenge: Useful Methods

Here is a list of useful enumerable methods. Try all of the methods at least once.

Iterators are extremely important for Ruby. Please go through each of these functions through Ruby's official documentation and build a single ruby file where you go through each of the following Ruby function. This will be listed in one of the assignments below.

- **.any? {|obj| block}** -> true or false
  - e.g.

```
["ant", "bear", "cat"].any? {|word| word.length >= 3}
```

    #==>true

- **.each ->** calls *block* once for each element in `self`, passing that element as a parameter.
  - e.g.

```
["ant", "bear", "cat"].each {|word| print word, "--"}
```

#==>ant--bear--cat--

- **.collect** {|obj| block} -> array; returns a new array with the results of running block once for every element in enum
  - e.g.

```
(1..4).collect {|i| i*i}
```

    #=> [1, 4, 9, 16]

  - e.g.

```
(1..4).collect { "cat" }
```

    #=> ["cat", "cat", "cat", "cat"]

- .**map** {|obj| block} -> enumerator; returns a new array with the results of running block once for every element in enum. it's exactly like .collect
- .**detect/.find** -> enumerator**;** returns the first for which block is not false.
  - e.g.

```
(1..10).detect { |i| i %5 == 0 and i % 7 == 0 }
```

    #=> nil

  - e.g.

```
(1..100).detect { |i| i %5 == 0 and i % 7 == 0 }
```

    #=> 35

- **.find_all** {|obj| block} or **.select** {|obj| block} ; returns an array containing all elements of enum for which block is not false
  - e.g.

```
(1..10).find_all { |i| i % 3 == 0 }
```

    #=> [3, 6, 9]

- **.reject** {|obj| block} -> opposite of find_all
  - e.g.

```
(1..10).reject { |i| i % 3 == 0 }
```

#=> [1, 2, 4, 5, 7, 8, 10]

- **.upto(limit)** -> iterates block up to the int number

```
5.upto(10) { |i| print i, " " }
```

#=> 5 6 7 8 9 10

- **.has_key?**(key) -> true or false
- .has_value?(value) -> true or false
- .key(value) -> returns the key of an occurrence of a given value. If the value is not found, returns nil
- .keys -> returns a new array populated with the keys from the hash

Reference: http://www.ruby-doc.org/core-1.9.3/Enumerable.html

Note: Note: Max file size is 5MB.

Files

Choose File No file chosen Upload

Timer | **HTML Helper** | **my_expect** | **Initializer**

Timer

We sometimes want to test how long a block of code takes. We are going to write a method called timer that takes in a block of code. The method will tell us how long it took to execute that block of code.

```
mkdir timer_fun
cd timer_fun
touch timer.rb
touch timer_spec.rb
```

timer_fun/timer.rb

```
def timer
end
```

timer_fun/timer_spec.rb

```ruby
require_relative 'timer'
RSpec.describe "My timer method" do
  it "should time how long the block takes" do
    total_time = timer { sleep(0.5) }
    expect(total_time < 0.509).to eq(true)
    expect(total_time > 0.5).to eq(true)
  end
end
```

Challenge

Make the tests pass.

Next