

In Ruby, all classes are derived from class Object. Since methods can be added to classes at any time, it is possible to re-open object and add methods, which then belong to every object.

This project involves adding a method `reach` to all objects. The `reach` method is like the standard `each` method, except that it applies to the leaves of container objects. The `each` method operates on each member of the container, but `reach` operates recursively, entering members which are also containers and iterating through their contents. For our purposes, an object is just a container if it has an `each` method. For instance,

```
irb(main):001:0> load("reach2.rb")
=> true
irb(main):002:0> [4, 9, "fred", 11].each { |x| print x, "\n"
4
9
fred
11
=> [4, 9, "fred", 11]
irb(main):003:0> [4, 9, "fred", 11].reach { |x| print x, "\n"
4
9
fred
11
=> [4, 9, "fred", 11]
irb(main):004:0> [4, [ "ding", "bat" ] , "fred", [1, 2, 3]].each { |x| print x, "\n"
4
["ding", "bat"]
fred
[1, 2, 3]
=> [4, ["ding", "bat"], "fred", [1, 2, 3]]
irb(main):005:0> [4, [ "ding", "bat" ] , "fred", [1, 2, 3]].reach { |x| print x, "\n"
4
ding
bat
fred
1
2
3
=> [4, ["ding", "bat"], "fred", [1, 2, 3]]
irb(main):006:0> 17.each { |x| print x, "\n" }
NoMethodError: undefined method `each' for 17:Fixnum
      from (irb):6
      from /usr/bin/irb:11:in `<main>'
irb(main):007:0> 17.reach { |x| print x, "\n" }
17
=> 17
irb(main):008:0> { "mike" => 17, "bill" => 3, "sally" => 25,
["mike", 17]
["bill", 3]
["sally", 25]
["alex", 9]
=> {"mike"=>17, "bill"=>3, "sally"=>25, "alex"=>9}
irb(main):009:0> { "mike" => 17, "bill" => 3, "sally" => 25,
mike
17
bill
3
sally
25
alex
9
=> {"mike"=>17, "bill"=>3, "sally"=>25, "alex"=>9}
```

```

["mike", 17]
["bill", 3]
["sally", 25]
["alex", 9]
=> {"mike"=>17, "bill"=>3, "sally"=>25, "alex"=>9}
irb(main):009:0> { "mike" => 17, "bill" => 3, "sally" => 25,
mike
17
bill
3
sally
25
alex
9
=> {"mike"=>17, "bill"=>3, "sally"=>25, "alex"=>9}
irb(main):010:0> { "mike" => [ 3, 9, 8, 7 ], "bill" => 3, "sa
["mike", [3, 9, 8, 7]]
["bill", 3]
["sally", [4, 7, 1]]
["alex", []]
=> {"mike"=>[3, 9, 8, 7], "bill"=>3, "sally"=>[4, 7, 1], "ale
irb(main):011:0> { "mike" => [ 3, 9, 8, 7 ], "bill" => 3, "sa
mike
3
9
8
7
bill
3
sally
4
7
1
alex
=> {"mike"=>[3, 9, 8, 7], "bill"=>3, "sally"=>[4, 7, 1], "ale
irb(main):012:0>[[[],4,5,[3,[],4,["hi","there"]]].each { |x| p
[]
4
5
[3, [], 4, ["hi", "there"]]
=> [[[], 4, 5, [3, [], 4, ["hi", "there"]]]
irb(main):013:0> [[[],4,5,[3,[],4,["hi","there"]]].reach { |x|
4
5
3
4
hi
there
=> [[[], 4, 5, [3, [], 4, ["hi", "there"]]]

```

Place your code in a file that can be loaded as shown above. It should re-open class Object and define the reach method. A method added to Object is inherited by every class, and so becomes part of every object in the Ruby system, including ones already created.

This assignment is a Jedi mind trick. My reach method is eight lines (excluding comments), and contains one if and no loops. And here's how you do it. Begin with this not-too-fancy partial solution:

```

class Object
  def reach
    yield(self)
    return self
  end
end

```

so

```

irb(main):001:0> load("foo.rb")
=> true
irb(main):002:0> 17.reach { |x| print x, "\n" }
17

```

```

mike
3
9
8
7
bill
3
sally
4
7
1
alex
=> {"mike"=>[3, 9, 8, 7], "bill"=>3, "sally"=>[4, 7, 1], "ale:
irb(main):012:0>[[],4,5,[3,[],4,["hi","there"]]].each { |x| p
[]
4
5
[3, [], 4, ["hi", "there"]]
=> [[], 4, 5, [3, [], 4, ["hi", "there"]]]
irb(main):013:0> [[],4,5,[3,[],4,["hi","there"]]].reach { |x|
4
5
3
4
hi
there
=> [[], 4, 5, [3, [], 4, ["hi", "there"]]]

```

Place your code in a file that can be loaded as shown above. It should re-open class Object and define the reach method. A method added to Object is inherited by every class, and so becomes part of every object in the Ruby system, including ones already created.

This assignment is a Jedi mind trick. My reach method is eight lines (excluding comments), and contains one if and no loops. And here's how you do it. Begin with this not-too-fancy partial solution:

```

class Object
  def reach
    yield(self)
    return self
  end
end

```

so

```

irb(main):001:0> load("foo.rb")
=> true
irb(main):002:0> 17.reach { |x| print x, "\n" }
17
=> 17
irb(main):003:0> [[],4,5,[3,[],4,["hi","there"]]].reach { |x|
[[], 4, 5, [3, [], 4, ["hi", "there"]]]
=> [[], 4, 5, [3, [], 4, ["hi", "there"]]]

```

This is a partial solution, since it works correctly for classes which are not collections. For classes that are, use their each method, and run reach on each member. How do you know if the current class is a collection? By definition, a collection is any object which has an each method. To find out, ask yourself. The code is:

```

if self.respond_to?(:each)

```

So, the complete body of reach should find out if its object has an each method. If so, use it to run reach on each member, otherwise just yield yourself. That's all it takes, young Padawan.

#### **Submission**

When your function works, is nicely formatted and documented, submit it using [this form](#).