1. What do the following Ruby expressions do?

- `x+2`
  
  This sends the message `+` to the object bound to `x` with the argument `2`.

- `octopus.swim("fast")`
  
  This sends the argument `swim` to the object bound to `octopus` with the argument "fast" (a string)

- `octopus.swim "fast"`
  
  Same thing - parens optional for one-argument methods.

- `octopus.tentacles = 8`
  
  This sends the argument `tentacles=` to the object bound to `octopus` with the argument `8` (it looks like an assignment, but it’s just a message send)

- `Aquarium.new("clownfish")`
  
  This creates a new instance of the class `Aquarium`. The argument "clownfish" is sent to the initialize method of the newly-created instance.

- `"clown", "fish".each {|s| puts s}`
  
  This sends the each message to a 2-element array with the contents "clown" and "fish". The each message takes a block (in curly brackets). The result is that each string is printed.

- `[1,2,3].map { |j| j*10}`
  
  This sends the map message to the 3-element array with the elements 1,2,3. The map message takes a block, and returns a new array with the results from evaluating the block for each element. The result is [10,20,30]

- `sum=0
  4.times {sum=sum+10}
  `
  
  The number 4 (an instance of Fixnum) gets the message times with the given block. The block is evaluated 4 times, so that `sum` becomes 40.

2. Write a Ruby class `Book`, which has fields for title and author. When you create a new instance of book you should give values for those fields. Also define getters (but not setters) for them. Finally, write a statement that makes a new instance of `Book` with a suitable author and title.

```ruby
class Book
  def initialize(author, title)
    @author = author
    @title = title
  end

  attr_reader :author, :title
end

b = Book.new("Robert Heinlein", "Methuselah’s Children")
```

3. Write a class `Delay` that implements delays (like the delay function in Scheme). The following code shows how it should work:

```ruby
n = 0
D = Delay.new {n=n+1; 3+4}
```
d.force
d.force
v = d.force
e = Delay.new {1/0}

After we evaluate these statements v should be 7, but n should only be 1 (since we only evaluate the block once). Further, since we never force e, we shouldn’t get a divide-by-zero error.

Solution:

class Delay
  def initialize(&p)
    @p = p
    @value = nil
    @unevaluated = true
  end
  def force
    if @unevaluated
      @value = @p.call
      @unevaluated = false
    end
    return @value
  end
end

4. Write a min method for the Enumerable mixin. You’ll need to decide how to handle finding the minimum of an empty collection. Bonus points for handling this in the same way Ruby itself does!

Hint: look at the implementation of map at the end of the inheritance.rb handout.

def min
  m = nil
  each { |x| m = x if m.nil? or x<m }
  return m
end

5. Consider the class and module definitions in self_super.rb linked from the 341 Ruby web page. Suppose we define a class C5 as follows:

class C5 < C1
  include M2
end

What is the result of evaluating these expressions?
```ruby
>> x = C5.new
=> #<C5:0x35f520>
>> x.test1
in mixin M2 test1
=> nil
>> x.test2
in mixin M2 test2
in C1 test2
=> nil
>> x.kind_of?(C5)
=> true
>> x.kind_of?(M2)
=> true
>> x.kind_of?(M1)
=> false
>> C5.ancestors
=> [C5, M2, C1, Object, Kernel]
>> C5.superclass
=> C1
>> C5.superclass.superclass
=> Object
>> C5.superclass.superclass.superclass
=> nil
```