CSE 341 Section 8 Worksheet

1. What do the following Ruby expressions do?

For example: x+2 means sending message + to x with argument 2

a. `octopus.swim("fast")`

b. `octopus.swim "fast"`

c. `octopus.tentacles = 8`

d. `Aquarium.new("clownfish")`

e. `["clown", "fish"].each {|s| puts s}`

f. `[1,2,3].map {|j| j*10}`

g.
```
sum=0
4.times {sum=sum+10}
```

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.

Then write a method called **bookinfo** that takes a block which takes two arguments, first argument is the title of the book, second is the name of the author. Then the block prints (using puts) a string saying the name and author of the Book object. **BookInfo** should call the block with its title and author fields.
For example, if we have a book object `bk` title “Ruby intro” and author “Alan”, calling

```
bk.bookinfo {...}
```

should generate the output:

```
title is Ruby intro, author is Alan
```

Suppose we are going to write **bookinfo_closure**, which requires passing in something that represents a closure (recall blocks are not closures). What are the lines that we should change to make it work?

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

```
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.

4.

```
class Beverage
  def initialize(price)
    @price=price
  end

  def more_expensive(a)
    return @price > a.price
  end
  attr_reader :price
end
class Soda < Beverage
  def initialize(sugar,price)
    @sugar = sugar
    @price = price
  end

  def healthier(a)
     return @sugar < a.sugar
  end
  attr_reader :price, :sugar
end
```

```
class Beer
  def initialize(price)
    @price = price
  end

  def mix(a)
    @sugar = a.sugar
    @price += a.price
  end
  attr_reader :price, :sugar
end
```

Suppose we have following bindings:

```
beer = Beer.new(5)
soda = Soda.new(10, 2.5)
beverage = Beverage.new(0.5)
```

Decide whether each of the following statement is valid or not:

```
a) beverage.more_expensive(beer)
b) soda.healthier(beverage)
c) beer.more_expensive(soda)
d) beer.sugar
e) soda.healthier(beer)
f) beer.mix(soda)
g) soda.healthier(beer)
```

5. The following deal with the `MyList` linked-list class, which you should download/copy into a text file from the "blocks_inheritance" file from the course webpage.

a. Write a `filter_block` method that acts like Haskell's filter function, using blocks. In other words, write a method that takes in a block and returns a new `MyList` object with all the elements of the self object that return true when passed into the block.
b. Write a `filter_proc` method that acts like filter_block, but explicitly takes in a proc instead of a block.
c. Write some code to test each of the above methods in the same file. Don't write actual unit tests; instead, write a to_s method for `MyList` and then use `puts` to print the result of multiple tests when running the file. (In other words, this is just practice for writing/using procs and blocks in ruby.)

6. This will again use the blocks_inheritance file, but will involve the `Point` classes. Define a `PointCircle` class that represents a circle using a point object that represents a circle whose center is at the origin and which takes in a point which lies somewhere on the edge of the circle (in other words, the radius of the circle is the distance from the origin to the point). Clients

should be able to access the radius of the circle with a method in `PointCircle`. What kind of Object do you need to pass into `PointCircle`? Is `PointCircle` a `Point`?