6.S189 Homework 4

`http://web.mit.edu/6.s189/www/materials.html`

What to turn in

Checkoffs 8 & 9 are due at **5 PM** on **Tuesday, January 22nd**. Checkoff 8 is over Exercises 8.1 - 8.3 and Checkoff 9 is over Exercises 9.1 - 9.2.

Exercise 8.1 – Intro to Object Oriented Programming

For the following questions, write your answers in the spaces provided.

1. What is the difference between a local variable and an object's attribute?

2. What method is called when the object is created?

3. Write the line of code you would use to call the `do_something` method of and object instance `obj`. Assume that the `do_something` method for `obj` exists.

8.2 – Understanding Objects

For the following questions, write your answers in the spaces provided.

1. Define a class called `Address` that has two *attributes*: `number` and `street_name`. Make sure you have an `__init__` method that initializes the object appropriately. You do not need to define any other methods.

2. Consider the following code:

```
class Clock(object):

    def __init__(self, time):
        self.time = time

    def print_time(self):
        time = '6:30'
        print self.time

clock = Clock('5:30')
clock.print_time()
```

   (a) What does the code print out? Guess first, and then create a Python file and run it.

   (b) Why does the code print this?

3. Consider the following code:

```
class Clock(object):

    def __init__(self, time):
        self.time = time

    def print_time(self, time):
        print time

clock = Clock('5:30')
clock.print_time('10:30')
```

(a) What does the code print out? Guess first, and then create a Python file and run it.

(b) What does this tell you about giving parameters the same name as object attributes?

4. Consider the following code:

```
class Clock(object):

    def __init__(self, time):
       self.time = time

    def print_time(self):
       print self.time


boston_clock = Clock('5:30')
paris_clock = boston_clock
paris_clock.time = '10:30'
boston_clock.print_time()
```

(a) What does the code print out? Guess first, and then create a Python file and run it.

(b) Why does it print what it does? (Are `boston_clock` and `paris_clock` different objects? Why or why not?)

Exercise 8.3 – Your First Class

For this exercise, you will be coding your very first class, a Queue class. Queues are a fundamental computer science data structure. A queue is basically like a line at Disneyland - you can add elements to a queue, and they maintain a specific order. When you want to get something off the end of a queue, you get the item that has been in there the longest (this is known as 'first-in-first-out', or FIFO). You can read up on queues at Wikipedia if you'd like to learn more.

Create a new file called `queue.py` to make your Queue class. In your Queue class, you will need three methods:

- `__init__`: to initialize your Queue (think: how will you store the queue's elements? You'll need to initialize an appropriate *object attribute* in this method)
- `insert`: inserts one element in your Queue
- `remove`: removes one element from your Queue and returns it. If the queue is empty, return a message that says it is empty (without throwing an error that halts your code).

When you're done, you should test your implementation. Your results should look like this:

```
>> queue = Queue()
>> queue.insert(5)
>> queue.insert(6)
>> queue.remove()
5
>> queue.insert(7)
>> queue.remove()
6
>> queue.remove()
7
>> queue.remove()
The queue is empty
```

Be sure to handle that last case correctly - when popping from an empty Queue, **print a message** rather than throwing an error.

Exercise 9.1 – Designing Your Own Inheritance

For this exercise, we want you to describe a generic superclass and at least three subclasses of that superclass, listing at least two attributes that each class would have. Remember what classes will inherit (will subclasses inherit attributes from their parent class? Will parent classes inherit from their subclasses? Will subclasses that share a parent inherit from one another? Be sure you're clear on this before continuing with this exercise.)

It's easiest to simply describe a real-world object in this manner. An example of what we're looking for would be to describe a generic Shoe class and some specific subclasses with attributes that they might have, as shown here:

```
class Shoe:
   # Attributes: self.color, self.brand

class Converse(Shoe): # Inherits from Shoe
   # Attributes: self.lowOrHighTop, self.tongueColor, self.brand = "Converse"

class CombatBoot(Shoe): # Inherits from Shoe
   # Attributes: self.militaryBranch, self.DesertOrJungle

class Sandal(Shoe): # Inherits from Shoe
   # Attributes: self.openOrClosedToe, self.waterproof
```

You can use any real-world object *except a shoe* for this problem :)

Exercise 9.2 – More Inheritance

Consider the following code:

```
class Spell(object):
    def __init__(self, incantation, name):
        self.name = name
        self.incantation = incantation

    def __str__(self):
        return self.name + ' ' + self.incantation + '\n' + self.get_description()

    def get_description(self):
        return 'No description'

    def execute(self):
        print self.incantation


class Accio(Spell):
    def __init__(self):
        Spell.__init__(self, 'Accio', 'Summoning Charm')

class Confundo(Spell):
    def __init__(self):
        Spell.__init__(self, 'Confundo', 'Confundus Charm')

    def get_description(self):
        return 'Causes the victim to become confused and befuddled.'

def study_spell(spell):
    print spell

spell = Accio()
spell.execute()
study_spell(spell)
study_spell(Confundo())
```

1. What are the parent and child classes?

2. What does the code print out? (Try figuring it out without running it in Python first)

3. Which `get_description` method is called when '`study_spell(Confundo())`' is executed? Why?

4. What do we need to do so that '`print Accio()`' will print the appropriate description?

```
Accio Summoning Charm
This charm summons an object to the caster, potentially over a significant distance.
```

   Write down the code that we need to add and/or change.