Worksheet 6

Object Oriented Programming

(OOP)

This form of programming involves creating 'objects' with different attributes. This is a very powerful tool in programming and can be used to make problems simpler in large projects.

Let's look at an example:

```
class Dog:

    num_legs = 4  # A class attribute

    def __init__(self, name, size, color):
        self.name = name
        self.size = size
        self.color = color
        self.sticks_found = 0

    def bark(self):
        print(self.size, 'bark')

    def found_stick(self):
        self.sticks_found += 1
        return self.sticks_found

    def leg_number(self):
        return self.num_legs

    def scare_away(self, intruder):
        if intruder:
                if self.size == 'big':
                        print('Big Growl')
                else:
                        return 'better call the police'
        return 'we are safe now'
```

This example is creating the framework for creating Dog objects. We create something called a **class,** which provides a general framework for creating different types of dogs.

There are two different types of attributes: **instance** and **class**

Instance Attributes are attributes of the object that are specific to each instance. Each instance has different data, and changing one's attribute doesn't change the other.

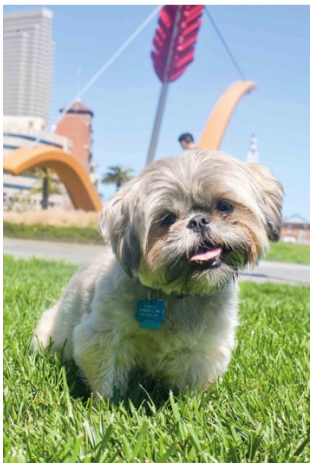
Variables in the above example including name, size, color, and sticks_found are examples of instance attributes.

Class Attributes are attributes of the object that general to the class. All instances of the class have the same attribute and data. Changing the class attribute changes that information for all instances.

The variable num_legs is a class attribute, since all dogs have 4 legs.

The functions (called **methods** when placed inside classes) including bark, found_stick, leg_number, and scare_away are also class attributes.

You probably also noticed the __init__ method, and the 'self' keywords inside. This is called the **constructor.** The __init__ method is IMPLICITLY called whenever we create an instance of the class. Let's look at an example of this.



This is a picture of Professor Denero's dog, Samosa. Let's create an **instance** of the Dog class that represents Samosa!

```
>>> samosa = Dog("samosa", "small", "brown")
```

Here, the **__init__** method is called, and the arguments taken in by Dog, get applied to the arguments 'name', 'size', and 'color' in the __init__.

self refers to the instance itself. In order to access the instance attributes inside the class, we use self.[attribute]

This is called **dot notation,** and is used to access different attributes.

Let's take a look at some of the uses.

```
>>> samosa.size
'small'
>>> samosa.color
'brown'
>>> samosa.sticks_found
0
>>> samosa.bark()
small bark
```

Notice how the method bark takes in one argument: **self**. Yet, we pass in no arguments? How is that possible? **Dot notation** makes the variable before the dot (in this case, *samosa*) to be the first argument in the method.

```
>>> samosa.found_stick()
1
>>> samosa.sticks_found
1
>>> samosa.leg_number()
4
>>> samosa.scare_away(True)
'better call the police'
```

This last example takes in two arguments: **self** and **intruder**. *samosa* is the first argument (self), and the boolean *True* is the second argument (intruder).

Let's make another instance of another dog: Spot

```
>>> spot = Dog("spot", "big", "brown")
```

When we change instance attributes of Spot, instance attributes of Samosa does not change.

```
>>> spot.name
'spot'
>>> samosa.name
'samosa'
>>> samosa.found_stick()
2
>>> spot.found_stick()
1
>>> spot.sticks_found
1
>>> samosa.sticks_found
2
>>> spot.scare_away(True)
Big Growl
'we are safe now'
```

Now, let's say that Spot loses a leg in a tragic battle with a disease.

Now, Spot is not like most dogs, as he has three legs. So, we have to change his num_legs class attribute.

```
>>> Dog.num_legs = 3
>>> samosa.num_legs
3
>>> spot.num_legs
3
```

Now, all dogs created or created originally have 3 legs. We don't want that! Let's try something else.

```
>>> Dog.num_legs = 4
>>> spot.num_legs = 3
>>> samosa.num_legs
4
>>> spot.num_legs
3
>>> Dog.num_legs
4
```

Here, we are changing the class attribute of ONE instance. When we do this, the class attribute gets turned into an instance attribute for the instance.

Practice

```
class Avenger:

    alive = True

    def __init__(self, name, punch, kick):
        self.name = name
        self.punch = punch
        self.kick = kick
        self.health = 30

    def flex(self):
        print('I am', self.name)
        self.check_death()

    def fight_evil(self):
        self.health -= 5
        self.check_death()
        print('Take that evil!')

    def fight_good(self, other_avenger):
        other_avenger.fight_evil()
        self.fight_evil()
        self.check_death()
        other_avenger.check_death
        if (self.punch + self.kick > other_avenger.punch +
other_avenger.kick):
            other_avenger.health -= 5
            print(self.name, 'wins.')
            other_avenger.check_death()
        else:
            self.health -= 5
            print(other_avenger.name, 'wins.')
            self.check_death()

    def check_death(self):
        if not self.alive:
            print(self.name, "is dead.")
            return
        if self.health <= 0:
            print(self.name, "is dead.")
            self.alive = False
            return
        print(self.name + " has " + str(self.health) + " health.")
        #str(some number) creates a string of that number
```

```
>>> iron_man = Avenger("Iron Man", 20, 5)
>>> captain_america = Avenger("Captain America", 20, 20)
>>> hulk = Avenger("Hulk", 50, 40)
>>> thor = Avenger("Thor", 100, 10)
>>> hulk.flex()



>>> iron_man.flex()



>>> captain_america.fight_evil()



>>> captain_america.fight_evil()


>>> captain_america.fight_good(hulk)




>>> iron_man.fight_good(captain_america) #civil war



>>> captain_america.fight_good(thor)



>>> thor.check_death()
```

```
>>> thor.flex()


>>> thor.fight_good(iron_man)




>>> hulk.fight_good(iron_man)




>>> hulk.fight_good(captain_america)




>>> thor.fight_evil()


>>> Avenger.health = 40
>>> hulk.health


>>> Avenger.alive = "Thanos wins"
>>> thor.alive



>>> thor.alive = False
>>> Avenger.alive


>>> Avenger.kick
```

Now, we can use something called **Inheritance** to simplify even more complex problems.

Let's say we add a new class to our original Dog example

```
class CuteDog(Dog):
    def bark(self):
        Dog.bark(self)
        print("bark! I am cute!")
    def scare_away(self, intruder):
        if intruder:
            print('Cute Power!')
        return 'we are safe now'
```

As you may notice in the first line, after CuteDog, we have the Dog class in parenthesis. This is to designate the Dog class as a PARENT of the CuteDog. This means that all methods not defined in the CuteDog class will be inherited by the Dog class.

Let's look at this in use.

```
>>> sam = CuteDog("Sam", "big", "white")
>>> sam.found_stick()
1
```

As you can see, even though the CuteDog class doesn't have found_stick defined in it, we went to the parent class and used the method there.

This is how the instance of sam was created, since CuteDog doesn't have an __init__ method.

```
>>> sam.bark()
big bark
bark! I am cute!
```

Here, we used the line Dog.bark(self) to call the parent class's bark method. This way, we don't have to copy and paste the entire method when overriding for one small change inside a child class.

```
>>> sam.color
'white'
>>> sam.scare_away(True)
Cute Power!
'we are safe now'
```

As you can see, the entire scare_away class was overridden, since sam is a CuteDog.

Let's do more practice using the Avenger class from above.

Practice

```
class Avenger:

    alive = True  # A class attribute

    def __init__(self, name, punch, kick):
        self.name = name
        self.punch = punch
        self.kick = kick
        self.health = 30

    def flex(self):
        print('I am', self.name)
        self.check_death()

    def fight_evil(self):
        self.health -= 5
        self.check_death()
        print('Take that evil!')

    def fight_good(self, other_avenger):
        other_avenger.fight_evil()
        self.fight_evil()
        self.check_death()
        other_avenger.check_death
        if (self.punch + self.kick > other_avenger.punch +
other_avenger.kick):
            other_avenger.health -= 5
            print(self.name, 'wins.')
            other_avenger.check_death()
        else:
            self.health -= 5
            print(other_avenger.name, 'wins.')
```

```
            self.check_death()

    def check_death(self):
        if not self.alive:
            print(self.name, "is dead.")
            return
        if self.health <= 0:
            print(self.name, "is dead.")
            self.alive = False
            return
        print(self.name + " has " + str(self.health) + " health.")


class HumbleAvenger(Avenger):

    humblescore = 0

    def flex(self):
        print("I am an Avenger")
        print("Humblebrag score: " + str(self.humblescore))

    def fight_evil(self):
        self.health -= 2
        self.check_death()
        print("Are you ok evil?")
        self.humblescore += 1

    def fight_good(self, other_avenger):
        Avenger.fight_good(other_avenger, self)
        print("I hope you are ok.")
        self.humblescore += 1

class CockyAvenger(Avenger):

    def flex(self):
        print("I am the best")

    def fight_good(self, other_avenger):
        Avenger.fight_good(self, self)
        print("I'm the best Avenger")
```

```
>>> iron_man = CockyAvenger("Iron Man", 20, 5)
>>> captain_america = HumbleAvenger("Captain America", 20, 20)
>>> hulk = Avenger("Hulk", 50, 40)
>>> thor = CockyAvenger("Thor", 100, 10)
>>> hulk.flex()



>>> iron_man.flex()


>>> captain_america.fight_evil()


>>> captain_america.flex()


>>> captain_america.fight_good(hulk)









>>> iron_man.fight_good(captain_america) #civil war




>>> captain_america.fight_good(thor)
```

```
>>> thor.check_death()


>>> thor.flex()


>>> thor.fight_good(iron_man)




>>> hulk.fight_good(iron_man)




>>> hulk.fight_good(captain_america)




>>> thor.fight_evil()


>>> HumbleAvenger.fight_good(thor, iron_man)




>>> captain_america.humblescore
```

```
>>> HumbleAvenger.humblescore


>>> CockyAvenger.flex()



>>> CockyAvenger.flex(hulk)


>>> CockyAvenger.flex(captain_america)
```

Implement a class called EvilAvenger which is the same as a CockyAvenger, except:

- It takes in another instance attribute called *archnemesis* which is an Avenger.
- when it fights evil, it fights its archnemesis.
- when it fights good, it fights evil instead.

```
class EvilAvenger(____________________):
```