1 GBA 465 Lab 07 - Skill-Building with Objects (Starter)

1.1 Part A - Who Let The Dogs Out

Actions:

- Create a class named `Dog`.
  - The class has one attribute called `name`.
  - The class has no behaviors (methods).
  - The class uses a constructor method to initialize the name attribute when the object is instantiated.
- Instantiate the Dog class to create an object. Use the constructor to give your dog a name.
- Print the dog's name from the object by referencing the `name` attribute.
- Example output (for Rex):

```
Rex
```

In [1]:
```
# Your implementation:
class Dog:
    def __init__(self,name):
        self.name = name
dog = Dog("Jay")
dog.name
```

Out[1]:
```
'Jay'
```

1.2 Part B - Bark It Out

Actions:

- Modify your `Dog` class (from Problem 10.10) in the cell below to add a new behavior (method) called `bark`.
- When called, the following string is printed: "[*dog's name*]: bark, bark, bark!".
- Example output (for Rex):

```
Rex: bark, bark, bark!
```

- Call the bark method on the dog object.

In [2]:
```
# Your implementation:
class Dog:
    def __init__(self,name):
        self.name = name
    def bark (self):
        print ("{}: Bark, bark, bark!".format (self.name))
dog = Dog("Rex")

dog.bark()
```

```
Rex: Bark, bark, bark!
```

1.3 Part C - Lots of Barking

Actions:

- Create a tuple containing five (5) dog names.
- Iterate over the dog names and instantiate a `Dog` object for each one.
- Call each dog's bark method.
- Example output (for Homer, Audrey, Winnie, Sammy, and Rex):

```
Homer: bark, bark, bark!
Audrey: bark, bark, bark!
Winnie: bark, bark, bark!
Sammy: bark, bark, bark!
Rex: bark, bark, bark!
```

In [3]:
```
# Your implementation:

dogNames= ('Homer','Audrey','Winnie','Sammy','Rex')

for dogName in dogNames:

    dog = Dog(dogName)

    dog.bark()
```

```
Homer: Bark, bark, bark!
Audrey: Bark, bark, bark!
Winnie: Bark, bark, bark!
Sammy: Bark, bark, bark!
Rex: Bark, bark, bark!
```

1.4 Part D - Barking at Another Dog

Actions:

- Modify your `Dog` class (from Problem 10.11) in the cell below:
  - Change the `bark` method so that another dog object can be passed in as an optional parameter called `targetDog`.
  - Use `None` for the default value.
  - Check for this value and adjust business logic accordingly.
- Create two dog objects called dog1 and dog2.
- Have dog1 bark at no one. The output should read "[*dog1 name*]: bark, bark, bark!".
- Example output (for Rex):

```
Rex: bark, bark, bark!
```

- Have dog1 bark at dog2. The output should read "[*dog1 name*] barks at [*dog2 name*]: bark, bark, bark!".
- Example output (for Rex and Homer):

```
Rex barks at Homer: bark, bark, bark!
```

In [4]:
```
# Your implementation:

class Dog:
    def __init__(self,name):
        self.name = name
    def bark(self,targetDog = None):
        if(targetDog == None):
            print(self.name+": bark, bark, bark!")
        else:
            print(self.name+"barks at " + targetDog.name + ": bark, bark, bark!")
dog1 = Dog("Homer")
dog2 = Dog("Rex")

dog2.bark(dog1)
```

```
Rexbarks at Homer: bark, bark, bark!
```

1.5 Part E - Puppies

Actions:

- Modify your `Dog` class (from Problem 10.13) in the cell below so that it has an attribute called `puppies`. This value is not a parameter of the constructor, but should be initialized in the constructor to an empty list.
- Create a method called `addPuppy` which takes a dog object as a parameter. In this method, append the dog parameter to the puppies list.
- Create a method called `hasPuppies`, which returns `True` if the dog has puppies or `False` if the dog has no puppies.
- Instantiate 4 dog objects:
  - The first will be the parent.
  - The second, third, and fourth will be puppies. Add each puppy to the parent.
- Modify the `bark` method to that puppies also bark when the parent barks.
- Call `hasPuppies` on the parent, and print the returned result to the screen. For example:

```
True
```

- Create another dog object.
- Have the parent dog bark at this other dog. You should see the bark output for both the parent and the children.
- Example output (for Scrooge McDog, with 3 puppies: Huey, Duey, and Louie):

```
Scrooge McDog barks at Homer: bark, bark, bark!
Huey barks at Homer: bark, bark, bark!
Duey barks at Homer: bark, bark, bark!
Louie barks at Homer: bark, bark, bark!
```

In [5]:
```
# Your implementation:
class Dog:
    def __init__(self,name):
        self.name = name
        self.puppies = []
    def addPuppy(self,puppy):
        self.puppies.append(puppy)
    def hasPuppies():
        if(len(self.puppies)==0):
            return False
        else:
            return True
    def bark(self,targetDog = None):
        if(targetDog == None):
            print(self.name+": bark, bark, bark!")
        else:
            print(self.name+"barks at " + targetDog.name + ": bark, bark, bark!")
        for puppy in self.puppies:
            puppy.bark(targetDog)
parentDog = Dog("Scrooge McDog")

childPuppy1 = Dog("Huey")
childPuppy2 = Dog("Duey")
childPuppy3 = Dog("Louie")

parentDog.addPuppy(childPuppy1)
parentDog.addPuppy(childPuppy2)
parentDog.addPuppy(childPuppy3)

otherDog = Dog("Homer")

parentDog.bark(otherDog)
```

```
Scrooge McDogbarks at Homer: bark, bark, bark!
Hueybarks at Homer: bark, bark, bark!
Dueybarks at Homer: bark, bark, bark!
Louiebarks at Homer: bark, bark, bark!
```

1.6 Part F - Add Some Attributes

Actions:

- Modify your `Dog` class (from Problem 10.14) in the cell below so that has an `age` attribute.
- Initialize the `age` attribute in the constructor as an optional parameter with a default value of `0`.
- Instantiate 4 dog objects:
  - The first will be the parent. Add an age of `5` using the constructor.
  - The second, third, and fourth will be puppies. Add an age of `1` for each puppy using the constructor. Add each puppy to the parent.
- Create a new method called `barkMyAge`. The method outputs the following: "[*dog name*]: bark, bark, bark, bark, bark!", except that the number of barks should be equal to the number of years old.
- Puppies will also `barkMyAge` when the parent does.
- Call `barkMyAge` on the parent.
- Example output (for Scrooge McDog, 5 years old, with 3 puppies: Huey, Duey, and Louie):

```
Scrooge McDog: bark, bark, bark, bark, bark!
Huey: bark!
Duey: bark!
Louie: bark!
```

In [12]:
```
# Your implementation:

class Dog:

    def __init__(self,name,age=0):
        self.name = name
        self.puppies = []
        self.age = age

    def addPuppy(self,puppy):
        self.puppies.append(puppy)

    def hasPuppies(self):
        if(len(self.puppies)==0):
            return False
        else:
            return True

    def barkMyAge(self):
        s = self.name+":"
        for i in range(self.age):
            s = s + "bark"

            if(i<self.age-1):
                s = s+","
            else:
                s = s+"!"
        print(s)

        for puppy in self.puppies:
            puppy.barkMyAge()

    def bark(self,targetDog = None):
        if(targetDog == None):
            print(self.name +": bark, bark, bark!")
        else:
            print(self.name+ "barks at " + targetDog.name + ": bark, bark, bark!"

        for puppy in self.puppies:
            puppy.bark(targetDog)

parentDog = Dog("Scrooge McDog",5)

childPuppy1 = Dog("Huey",1)
childPuppy2 = Dog("Duey",1)
childPuppy3 = Dog("Louie",1)

parentDog.addPuppy(childPuppy1)
parentDog.addPuppy(childPuppy2)
parentDog.addPuppy(childPuppy3)



parentDog.barkMyAge()
```

```
Scrooge McDog:bark,bark,bark,bark,bark!
Huey:bark!
Duey:bark!
Louie:bark!
```

1.7 Part G - Dog Age Revisited

Actions:

- Modify your `Dog` class (from Problem 10.15) in the cell below so that it has a new method called `getMyHumanAge`.
- This method has no parameters, but returns the calculation of the dog's age multipled by 7.
- Create a dog, and be sure to pass the dog's age to its constructor.
- Call the `getMyHumanAge` method on one of your dogs, and print it to the screen.
- Example output (Homer, age 8):

```
Homer's human age is 56.
```

In [7]:
```
# Your implementation:
```

1.8 Part H - List of Objects

Actions:

- Create a tuple of five (5) names called `dogNames`.
- Create an empty list called `dogs`.
- Iterate over the names. For each name:
  - Generate a random number from `1` to `7` to represent the dog's age.
  - Instantiate a dog object, passing in the name and age to the constructor
  - Add the dog object to the `dogs` list.
- Iterate over the `dogs` list. For each dog:
  - Call the `barkMyAge` method.
- Example output for Homer, Audrey, Winnie, Sammy, and Rex:

```
Homer: bark, bark, bark!
Audrey: bark, bark!
Winnie: bark, bark, bark, bark, bark, bark!
Sammy: bark, bark, bark, bark, bark!
Rex: bark, bark, bark!
```

In [8]:
```
# Your implementation:
```