Object Oriented Programming

Quick Review -- Basics

Every **person** (Object) is a **Human** (Class)
    Classes are basic templates for objects
An **object** is an instance *of a* **class**
A **person** is an instance *of a* **human**
A **single person** (instance) has a **name** (Instance variable) and **age** (Instance variable)
Objects and instance variables have a "*has-a*" relationship
An instance variable is an attribute specific to an instance.
An **object** *has an* **instance variable**.
A **person** *has a* **brain**
A single person can **eat** (method) and **sleep** (method)
    Methods describe a certain "behavior" of an object
The **population** (class variable) of the Earth is 7 billion
    Class variable: attributes for the class as a whole, shared by all instances of a class

Warm Up -- Animals

1. An animal may be distinguished by number of legs, number of eyes, a sound, and color. Design an Animal class that keeps track of these features, as well as the total number of animals that have been created.

2. Now, let's differentiate a bit between these animals. Design two classes -- one for a dog, and one for a cat.

Queue

Cross out as incorrect and unnecessary lines in the following code so that the doctests pass for both classes.

```
class Queue(object):
    """Creates a Queue, which is like a list that supports 2 operations:
    enqueue (adding an item to the back of the queue) and
    dequeue (removing an item from the front of the queue).
    The queue cannot enqueue past maximum capacity.

    >>> q = Queue(2)
    >>> q.enqueue(5)
    >>> q.enqueue(3)
    >>> q.dequeue()
    5
    >>> q2 = Queue(1)
    >>> q2.enqueue(90)
    >>> q2.enqueue(2)
    Out of space
    """
    self.items = []
    def __init__(self, capacity, items):
    def __init__(self, capacity):
        self.capacity = capacity
        self.enqueue(capacity)
        self.items = []
    def enqueue(self, item):
        if len(self.items) == self.capacity:
        if len(self.items) == capacity:
            items.append(self, item)
            self.items.append(item)
        else:
            print('Out of space')
    def dequeue():
    def dequeue(self):
        self.items.pop(0)
        return self.items.pop(0)
        return self.items.pop()

class PriorityQueue(Queue):
    """A PriorityQueue is like a sorted list that supports two operations:
    enqueue (adding an item to the PriorityQueue) and
    dequeue (removing the smallest item from the PriorityQueue).

    >>> p = PriorityQueue(2)
    >>> p.enqueue(5)
    >>> p.enqueue(3)
```

```
>>> p.dequeue()
3
>>> p2 = PriorityQueue(1)
>>> p2.enqueue(90)
>>> p2.enqueue(2)
Out of space
"""
def __init__(self, capacity):
    Queue.__init__(capacity)
    Queue.__init__(self, capacity)
    self.items = []
    self.items.sort()
def enqueue(self, item):
    self.enqueue(item)
    Queue.enqueue(self, item)
    if len(self.items) == Queue.capacity:
        print('Out of space')
    self.items.sort()
def dequeue(self):
    return self.dequeue()
    return Queue.deqeue(self)
```