CSE 4256: Homework 10 - Due Friday, July 16

In this class, they will learn the basics of object oriented programming in Python. Below is a simple Dog class.

```
class Dog:
    def __init__(self, age, name):
        self.age = age
        self.name = name

    def bark(self):
        return "arf"

    def is_old(self):
        if self.age >= 10:
            return True
        else:
            return False
```

Unlike in Java, we don't need to declare variable names separately. Also, in Python every variable is public and the `self` keyword acts like `this` in Java.

The `__init__()` method is equivalent to a constructor in Java: it is code that runs when a new object is created. The name of the variables we pass to the constructor don't need to be the same as the names in the method. So for, example, self.a = age will also work.

`__init__()` is what is called a **dunder** method, for "double underscore". We'll see other dunder methods shortly.

Now we create some dogs and run some methods:

```
> d1 = Dog(11, "Jake")
> d2 = Dog(5, "Clyde")
> d1.age
> 11
> d2.age
> 5
> d1.bark()
> 'arf'
> d1.is_old()
> True
> d2.is_old()
```

```
> False
```

Next, we'll modify the __init__ method so that name defaults to "Fido" if no name is passed.

```
def __init__(self, age, name = "Fido"):
Then create some new dogs:
> d1 = Dog(12)
> d2 = Dog(7, "Lassie")
> d1.name
> 'Fido'
> d2.name
> 'Lassie'
```

Now what if we try to print one of the dogs?

```
> print(d2)
> <__main__.Dog object at 0x7f041c882ee0>
```

While this is technically correct, it's probably not what we want. So we'll create another dunder method. Note that we are using the format() method of the string class.

```
def __repr__(self):
    return "{} is {} years old.".format(self.name, self.age)

> d1 = Dog(10, "Jake")
> print(d1)
> Jake is 10 years old.
```

Homework
For the assignment, you'll be building a class to represent fractions. We'll get you started below.

```
Class Fraction:
    def __init__(self, num, den):
        self.num = num
        self.den = den

    def __repr__(self):
        return str(self.num) + "/" + str(self.den)
```

We can now build a Fraction and then print it:

```
> f = Fraction(1, 3)
> print(f)
1/3
```

What if one of the arguments is negative?

```
> f = Fraction(-1, 3)
> print(f)
-1/3
```

This look good, but what about:

```
> g = Fraction(1, -3)
> print(g)
1/-3
```

This is probably not what we want. It should print -1/3.

This is even worse:

```
> h = Fraction(-1, -3)
> print(h)
-1/-3
```

You'll need to update the `__init__()` method to fix this problem. Also, modify the `__init__()` method so the the fraction is in reduced form. You can do this by using the `gcd` class from the math package.

Here is how you would write a dunder method for multiply.

```
def __mul__(self, other):
    return Fraction(self.num * other.num, self.den * other.den)
```

```
> f = Fraction(2, 3)
> g = Fraction(3, 4)
> f * g
```

If f and g are Fractions, the dunder methods allows us to write f * g instead of using a method.

Assignment
Download the file "homework10.py" and complete the methods listed in the file. Submit your Python file on Carmen.