CSC 120 **ICA-8**

Work with your neighbor. (This will be graded for participation only.)

1. The code for the Counter class is given here for reference:

```
class Counter:
    def __init__(self, name):
        self._name = name
        self._count = 0

    def click(self):
        self._count += 1

    def count(self):
         return self._count

    def __str__(self):

        return "Counter :" + self._name + "->" + \

                               str(self._count)
```

Discuss these questions with your group:

a) When you call the constructor `Counter`, what method is actually called by the Python interpreter?

b) How many arguments does `Counter()` need when you call it?

c) What do we call the variables `self._name` and `self._count`?

d) The `__init__()` and `__str__()` methods have double underscores in the definitions. What do we call methods with double underscores?

e) Type in the `Counter` class (or download it from the class website—the link is next to the ICA) and write some test code. Create two different `Counter` objects and assignment them to variables. Increment the counter in each object a few times. Then print out each counter object.

2. The + key is broken on your keyboard. Implement `Counter` to use another means to keep track of the count.

3. Given the class definition for `Point` below:

```
import math
class Point:
    def __init__(self, x, y):
        self._x = x
        self._y = y

     def translate(self, dx, dy):
         self._x += dx
         self._y += dy

     def distance_from_origin(self):
         return math.sqrt(self._x**2 + self._y**2)
```

Write an `__eq__()` method for the `Point` class.

4. Write a class named `Fraction` that represents a fraction by defining the following attributes and methods:

Attributes:
- `_num`: an integer representing the numerator
- `_dem`: a non-zero integer representing the denominator (assume it is not zero)

Methods:
- `__init__(self, n, d)`: where `n` is the numerator and `d` is the denominator
- `__str__(self)`: returns a string of the form shown in **Usage** below
- `get_numerator(self)`: returns the numerator attribute
- `get_demoninator(self)`: returns the denominator attribute
- `__eq__(self, other)`: returns `True` if two `Fraction` objects are equal and `False` otherwise.

Usage:
```
>>> x = Fraction(1,2)
>>> str(x)
'1/2'
>>> y = Fraction(4,8)
>>> str(y)
'4/8'
>>> x == y
True
```

Note: The solution to this problem is covered in detail in OCA-4.