**4CS001**

**Python Workshop 8: Object Oriented Programming**

These are instructions for your eighth Python workshop. These workshops are designed for you to be able to make a good start on in your weekly lab sessions, but you may need to take them home to finish. There are some tasks that will not have been covered in the lecture, these will require you to do some independent research!

When you have finished this workshop, please upload your work to Canvas. This way you can always check back later to see how you solved a problem.

**Select:** Start > All Programs > IDLE (Python GUI)

**Part 1**

**(a)** What are the two kinds of entities “bundled” in a class?

**(b)** Any number of what kind of entity that can be created from any given class?

**(c)** What are the three fundamental features that an object-oriented programming language has in support of object-oriented programming?

**(d)** What else does encapsulation provide other than the ability to bundle together instance variable and methods?

**(e)** Describe what it means for a member of a class to be defined private.

**(f)** Explain the purpose of getters and setters.

**(g)** Explain what the special identifier self is for in Python.

**(h)** Explain the use of name mangling in Python.

**(i)** Explain when special methods \_\_str\_\_ and \_\_repr\_\_ are used in Python.

**(j)** Explain when a subclass can serve as a subtype of another type.

**(k)** For a given object obj, show how in Python the type of the object may be determined from within a program or the Python shell.

**(l)** Explain the concept of polymorphism in object-oriented programming.

**(m)** Explain the advantages of using polymorphism in program design.

**(n)** What is the meaning of “duck typing” in Python?

**Part 2**

**1a.** Implement a Range class representing a range of integers that contains two integer instance variables, \_\_start and \_\_end. Implement the special method \_\_str\_\_ such that consecutive integers are displayed, for example '10,12,13,14,15,16', when the object is used with print.

**1b.** Implement the special method \_\_lt\_\_ for the Range class in part a above so that range1 < range2 evaluates to True if all the values in range1 are less than all the values in range2, and returns False otherwise.

**2.** Implement a Money class that stores monetary values in pounds and pence.

Special method \_\_init\_\_ should have the following function header:

def \_\_init\_\_ (self, pounds, pence)

Include special method \_\_repr\_\_ (\_\_str\_\_) for displaying values in pounds and pence: £0.45, £1.00, £1.25.

Also include special method \_\_add\_\_, and three getter methods that retrieve the stored monetary value in other currencies (without changing the pound and pence amount). Choose any three currencies to convert to.

**3.** Implement a class named AvgList as a subclass of the built-in list class in Python, able to compute the average of a given list of numeric values. If the list contains any non-numeric types, a ValueError exception should be raised.

**4.** Implement a FootMeasure class that stores a linear measurement of feet and inches. Special method \_\_init\_\_ should have the following function header:

def \_\_init\_\_(self, feet = 0, inches = 0)

Thus, the class should be able to create a FootMeasure object various ways by use of optional keyword arguments:

meas = FootMeasure()  
meas = FootMeasure(feet = 5)  
meas = FootMeasure(feet = 5, inches = 8)  
meas = FootMeasure(inches = 68)

Implement special method \_\_repr\_\_ (\_\_str\_\_) in the class such that measurements are displayed appropriately:

5 ft. **NOT** 5 ft. 0 in.  
5 ft. 8 in. **NOT** 68 in.

When the measurement is 0, it should be displayed as, 0 ft. 0. ins.

Include special method add() for adding two FootMeasure values. Also include all the special methods for implementing the relational operators.

**Part 3**

1. Develop an abstract class named Temperature that stores a temperature.

Special method \_\_init\_\_ should have the following function header:

def \_\_init\_\_ (self, temperature)

The abstract class should contain the following methods:

\_\_str\_\_ – e.g., 75 degrees Fahrenheit  
above\_freezing() - Returns True if temperature above freezing point   
convert\_to\_fahrenheit – Returns a new Temperature object converted to degrees Fahrenheit  
convert\_to\_celsius – returns a new Temperature object converted to degrees Celsius  
convert\_to\_kelvin – returns a new Temperature object converted to degrees Kelvin

**(b)** Develop subclasses Fahrenheit, Celsius and Kelvin to appropriately implement each of the methods in the abstract Temperature class.

Note that when a meaningless conversion method is applied, for example:

temp1.convert\_to\_fahrenheit()

When temp1 is an object of type Fahrenheit, then a copy of the Temperature object should be returned.

Demonstrate the correctness of your classes by doing the following:

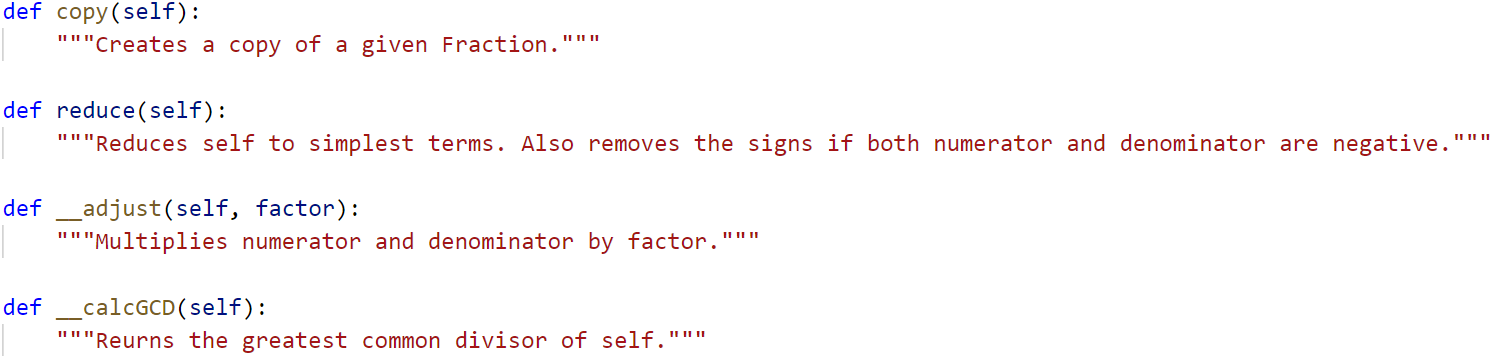
Create a list of Temperature objects of a mix of Temperature types

Print out the value of each temperature and if it is above the freezing point by calls to methods get\_temperature and above\_freezing.

Create a new list of temperatures containing each temperature of the original list converted to a common temperature scale (Fahrenheit, Celsius or Kelvin). For each temperature object in the new list, print out the value of that temperature and if it is above the freezing point.

**Part 4 (Optional)**

**1.** Complete the Fraction class that we looked at in the lecture (on Canvas) by implementing the missing methods below:



**2.** Modify the Fraction class to include a division operator by implementing special method \_\_truediv\_\_.

**3.** Modify the Fraction class to support relational operators  
(<, <=, >, >=, ==, !=) by implementing the special methods:

\_\_lt\_\_, \_\_le\_\_, \_\_gt\_\_, \_\_ge\_\_, \_\_eq\_\_ and \_\_ne\_\_