Lab 9

Learning Objectives:

I. **What characters do I type next?** *How do I use objects oriented programming concepts in Python?* In this lab, you will be using object-oriented programming techniques in Python and will get experience writing classes, along with importing and inheriting.

E. **Don't Reinvent the Wheel** *Don't write code that you don't need to.* Object oriented programming approach allows you to write very reusable code to reduce the amount of separate programs you need to write. In addition you will be using python built-in objects and functions to internally store data in your class.

F. **Write and Test, Write and Test....** *Write code in small bits that you can easily test.* If you code an enormous project and then test it all at once, how do you know where your error is? This lab gives you the opportunity to practice testing your code very modularly. Object-oriented programming makes it particularly easy to organize code in ways that can be tested.

Part 1 will have you create a class and get practice with object oriented programming concepts
Part 2 is for you to test out functionalities of your class
Part 3 if you have time lets you implement some more advanced functionality to your class.

Part 1: Write a Set Class

For this part of the lab you will be constructing a class that acts as a Set object. A Set is a collection of arbitrary objects, with the caveat that no object is repeated, i.e. it can't show up more than once in the Set. From this, we have the usual collection methods, like adding an object, removing an object, and asking for the size of the collection, but Sets also have special operations called union, intersection, and set difference.

The union of two Sets, which we'll notate `SetA | SetB`, returns a new Set with all of the members that were either in Set A or Set B or both. The intersection of two Sets, which we'll notate `SetA & SetB`, returns a new Set with the members that were in both SetA and SetB. The difference of two Sets, or `SetA - SetB`, returns a new Set with all of the members that were in SetA but not in SetB.

Your implementation should be done in a new file you create called `sets.py`. **We recommend you implement the below items incrementally and switch between this part (Part 1) and the next (Part 2) to test them as you go along.**

Functions to implement in your Set class:

- Constructor (you should not require any constructor arguments)
- Contains (checks if an element is in the set)
- Add (adds an element to the set)
- Remove (removes an element from the set—should handle case where the element is not in the set)
- Size (returns number of elements in the set)
- Union (see above description—overload the `|` operator)

- Intersection (see above description—overload the `&` operator)
- Subtract (see above description—overload the `-` operator)

Remember that the set cannot have repeated elements! Add any attributes or helper methods that you think are necessary for implementing the class and it is good style to write short docstrings for your methods that explain what they do, what the arguments should be, and what is returned. Note that it may be helpful to use some of these functions listed above to help you implement some other of the functions listed!

Hint: How do you store all the elements in your set? To do this you may want to have some type of basic Python data structure inside your Set class that is initialized in the constructor and updated with calls to your functions. What basic python data structure have we learned about that can have different types of elements in it?

Part 2: Testing

Once you've implemented all of the methods for the Set class, you should write a test script `settest.py` that makes sure your implemented methods work properly. Remember to import your Set class here!

There are many ways to test out your functions and you can choose to do this however you want. Some possible ideas are to to add and remove a bunch of values and then check to see if `MySet.size()` returns the proper size. Even better, you can test your union, intersection, and subtraction methods, by doing something like checking the intersection and difference of a Set of odd numbers and a Set of prime numbers (which you can generate yourself by just having a few manual `.add()` calls).

Part 3: If you have time: Iterators

You are now going to implement some more complex functionality in your set so that you can iterate through all the elements.

An `iterable` is an object that can return its members one at a time. A simple way of thinking of this is that an object that you can for loop over its elements with the syntax `for element in object` is an iterable. Examples of iterable objects in python that come to mind are lists, strings, numpy arrays, file object, etc.

An `iterator` is an object that can move through and access a stream of data. You have dealt with iterators extensively in this class although you may not know it as the for loop syntax above invokes these objects to access the individual elements in the loop. One can explicitly create an iterator with the syntax: `iterator = iter(iterable object)` and then move through elements with `iterator.next()`

We want you to implement this functionality for your Set class. To make the object iterable you will need to overload the `iter` method and then also overload the `next` operator. Test to make sure it works!