Lab 5 – CSCI 112

Due Date: Wednesday, Sep 30 at 5:00pm EST

Information

- This lab is intended to be completed **individually.**
- The files must be submitted with the exact file name provided in this file. If the file names do not match you will receive **zero** points for that file.
- Before you submit, make sure that your code runs. Any code which does not run without errors will receive **zero** points.
- Do not share your work with anyone other than Professor Khan or the TAs. You may discuss algorithms, approaches, ideas, but **NOT** exact code.
- If you submit work after a second past the due date **WILL** be locked out from submission.

Review

Sets

Sets are a container similar to bags, except that it only keeps **one** copy of each unique thing added (or treats multiple copies of a thing as one). Sets share multiple behaviors with bags, and therefore their implementations should be quite similar.

Inheritance

When creating objects with similar behavior, inheritance can assure that you will not need to have identical code in multiple places. The code for **__len__**, for example, is the same regardless of implementation in our collections.

Abstract Classes

Sometimes when code is shared by multiple classes, but there is no concrete super class in which to this behavior, an **abstract** class can be created as an abstract idea class from which all other classes inherit. An abstract collection, for example, could contain the code that is the same for **all** collections, but not necessarily a class to be instantiated itself.

Assignment

Unzip the Lab_5 folder and put your names in each file. Open the file **bagInterface.py** and review the interface for the bag collections. Note that this code can be loaded into the shell, but the methods won't do anything. Copy your bag code from last week's lab into the new folder for modification in today's lab.

Open the file **testBag.py** and examine the code for testing a bag type. Note that the test function creates some bag objects and exercises the methods on them. Note that this function can be called with any bag type as an argument.

The other files contain code provided for you from the book. Familiarize yourself with the methods provided and how they work.

Task 1 – Abstract Classes [7 points]

You will be modifying the bag classes to inherit from **AbstractBag**, which will inherit from **AbstractCollection**. Create a new file named **abstractCollection.py**. Inside this file, create an abstract class **AbstractCollection**. Move the code for **isEmpty**, **__len__**, **__add__**, and **count** methods to the **AbstractCollection** class. These methods then should be deleted from your bag classes. Make sure you change any direct references to class types to be the type of self instead:

```
def __add__:
   result = type(self)(self) # instead of ArrayBag(self)
   for item in other:
      result.add(item)
   return result
```

Create a new file named **abstractBag.py**. In this file define an **AbstractBag** class. This class should inherit from **AbstractCollection**. Examine your various bag classes for commonality in implementation in methods. Move the appropriate methods to the **AbstractBag** class and remove them from your bag classes.

Make each of your bag classes a subclass of **AbstractBag**. Modify the **__init__** methods to make use of **super().__init__** and set up the unique instance variables necessary for the implementation. (Hint: what's the difference between **LinkedBag** and **ArrayBag**).

Modify the **ArraySortedBag** and **LinkedSortedBag** classes to inherit from **ArrayBag** and **LinkedBag** respectively. Modify the **__init__** method and delete the redundant methods. At this point, you should be able to test your new bag implementations with **testBag.py** like last week.

Task 2 – Sets [8 points]

Open **setInterface.py** and study the method headers. Ignore **__or__**, **__and__**, and **__sub__** for now. Think about where some implementations could go in your class hierarchy. Draw a class diagram to help illustrate this. Include all of the classes you've created so far in the diagram. Which implementations already in the hierarchy behave correctly for sets? Take a photo of the diagram that you draw and upload it along with the other files in the zip archive during submission.

Create the files **arraySet.py**, **linkedSet.py**, and **arraySortedSet.py** and define the appropriate classes. You may skip creating a set for the **LinkedSortedSet.py**. Don't include **__or__**, **__and__**, or **__sub__** for now. Using inheritance from the appropriate super class, which methods do you need to override? (Hint: it's not that many.) Make sure to account for mutations. Test each class using test in **testSet.py**.

The union, intersection, and difference operators are set-specific. The following table lists the operations and the corresponding Python methods.

Operation	Method	What it Does
`s1 | s2`	`__or__(self, other)`	`Returns a set containing all the items in s1 and s2`
`s1 & s2`	`__and__(self, other)`	`Returns a set containing only the items in s1 that are also in s2`
`s1 - s2`	`__sub__(self, other)`	`Returns a set containing only the items in s1 that are not in s2`

Does the behavior of these three methods change depending on the implementation? Could you use a for loop or iterator? These are abstract set operations.

Create a new file named **abstractSet.py** and define the **AbstractSet** class with these methods. Have your set implementation classes inherit from two parents to gain the benefit from two locations. You may achieve this by providing two class names to your class definition: `class ClassZ(ClassA, ClassB):`

Test each set using the **testSetSpecific** function in **testSet.py**.

Run the file **concordance.py** and provide the file **test.txt** as the input file name. The program displays the frequencies of each word in the input file. Note how the code uses your bag and set classes to simplify the code.

What To Turn In

Create a zip file named **Lab5_<your W&L ID>.zip**. Inside this zip archive should submit all the files from both tasks. Make sure your files have the correct names (including case), and extensions.