**MSCF Financial Computing I**

**Mini 1, 2022**

**Homework 3**

***Due At 11:59 pm Sunday, Sept. 18***

***You will lose 10 points per hour after that time***

1. **SinglyLinkedList Class (50 points)**

In MSCF Programming Prep Day 1, Part 1, we described the Singly Linked List, and implemented part of a **SinglyLinkedList** class. Here we will make many modifications and extensions, to gain more practice with linked data structure programming, class implementation, recursion, and iterables (from FC I Week 3 Part 1), and the like.

The code file **SinglyLinkedList.py** contains the **SinglyLinkedList** class, with **\_\_str\_\_()** slightly modified. The code file **hw3\_1.py** contains test code for the **SinglyLinkedList** class.

1. Run **hw3\_1.py**. Part **3\_1.a** tests **SinglyLinkedList** object creation, display with **print()** (with implicit call of **str()**), and the initial **append()** and **insert()** functions.
2. Modify the **insert()** method to behave more like the **list** type’s **insert()** method. If **insert()** is called with just one argument, then that argument’s value should be inserted at the head of the **SinglyLinkedList**. But if **insert()** is called with two arguments, then the first argument should be treated as the *index* ahead of which the second argument value should be inserted. Uncomment the part **3\_1.b** test code, save, and test. (The test code for **3\_1.a** should still work.)
3. Modify the **\_\_init\_\_()** method so that if an (optional) iterable argument is passed in, the **SinglyLinkedList** object is initialized with the items from that iterable. Uncomment the part **3\_1.c** test code, save, and test. (The test code for **3\_1.a** and **3\_1.b** should still work.)
4. Define the **\_\_len\_\_()** method that returns the number of items in the **SinglyLinkedList**. Uncomment the part **3\_1.d** test code, save, and test. (Previous tests should still work.)
5. Define the **reverse()** method, that reverses the order of the items in the **SinglyLinkedList**. (You must do direct **SinglyLinkedList**/**\_SLLnode** operations to accomplish this. Reversing a singly linked list is a popular interview question.) Uncomment the part **3\_1.e** test code, save, and test. (Previous tests should still work.)
6. Define the **copy()** method, that makes and returns an independent copy of a **SinglyLinkedList** object. Uncomment the part **3\_1.f** test code, save, and test. (Previous tests should still work.)
7. Define the **\_\_eq\_\_()** method, that tests whether two **SinglyLinkedList** objects contain equal items in the same order. This function is called when **==** is used to compare objects. (By default, the inverse of **\_\_eq\_\_()** is used when **!=** is used to compare objects, or you can define a separate **\_\_ne\_\_()** method.) Recall that **is** tests whether two variables refer to the same object: you don’t need to define a function for this, since **is** directly compares objects’ ids. Uncomment the part **3\_1.g** test code, save, and test. (Previous tests should still work.)
8. Modify the **SinglyLinkedList** class so that a **SinglyLinkedList** object is *iterable*. Uncomment the part **3\_1.h** test code, save, and test. (Previous tests should still work.)
9. Modify the **SinglyLinkedList** class so that two **SinglyLinkedList** objects can be concatenated with the **+** operator. That is, if **s1** and **s2** are **SinglyLinkedList** objects, then **s1 + s2** should be a new **SinglyLinkedList** object consisting of all of the item values in **s1** followed by all of the item values in **s2**. You will need to define the **\_\_add\_\_()** method for this. Uncomment the part **3\_1.j** test code, save, and test. (Previous tests should still work.)
10. Define the **\_\_contains\_\_()** method, that tests whether its argument value equals one of the items in the **SinglyLinkedList**. Uncomment the part **3\_1.j** test code, save, and test. (Previous tests should still work.)
11. Define the **count()** and **remove()** methods to behave as they do for **list**. Uncomment the part **3\_1.k** test code, save, and test. (Previous tests should still work.)
12. **Binary Search Trees (50 points)**

In MSCF Programming Prep Day 1 Part 2, we introduced the Binary Search Tree data structure. In MSCF FC I Week 3 Part 2, we described deleting a node; left and right rotations; and Day’s algorithm for balancing a Binary Search Tree.

The code file **BinaryTree\_hw3.py** contains the **BinaryTree** class solution code from our previous Binary Search Tree homework. The code file **BT\_app\_hw3.py** contains test code for some existing **BinaryTree** class methods, and test code (commented out) for the new methods we will develop in Homework 3.

1. Run **BT\_app\_hw3.py** to confirm that existing code works as expected.
2. Define the **delete()** method that takes a value as an argument, and that deletes the node having that value from the **BinaryTree**. As we described in lecture, the left subtree of the deleted node should move up to take the place of the deleted node, and the right subtree of the deleted node should be connected to the tree at the right-most leaf of that left child. (But recall that there are special cases to handle: what if no left subtree? What if no right subtree?) If the value does not exist in the **BinaryTree**, nothing should happen. Uncomment the part **2.b** test code; save and test.
3. Define the **is\_balanced()** method that returns **True** if the **BinaryTree** object is balanced, and **False** otherwise. Uncomment the part **2.c**. test code; save and test.
4. Define the **rotate\_left()** method that takes a value as an argument, and performs a *left rotation* on the node with that value in the **BinaryTree** object, as described in the lecture. If the value does not exist in the **BinaryTree** object, nothing should happen. Uncomment the part **2.d** test code, save, and test.
5. Define the **rotate\_right()** method that takes a value as an argument, and performs a *right rotation* on the node with that value in the **BinaryTree** object, as described in the lecture. If the value does not exist in the **BinaryTree** object, nothing should happen. Uncomment the part **2.e** test code, save, and test.
6. Define the **Day\_balance()** method, that applies Day’s algorithm as described in the lecture to balance the **BinaryTree** object. Uncomment the part **2.f** test code, save, and test.

***REMEMBER*** to put all team members’ names (Andrew IDs) into your source code file.Put your **hw3\_1.py, SinglyLinkedList.py, BinaryTree\_hw3.py,** and **BT\_app\_hw3.py** files into a **Team***N***\_HW3.zip** archive, where *N* is your team number, and upload to Canvas.