Objectives

* Creating a Linked List (to support future ADT)

Due Date

This assignment is due on Sunday, February 11, 2024, by 6:00 pm.

**Remarks:**

* When you are asked to hand in code, you cut-and-paste your code as text and paste into this Word document immediately following the prompt for your code.
* Include with your code several test case examples and your results.
* This lab is cooperative - talk with your partner as you go through, and make sure you are progressing together.

Value

This assignment is worth 10 points.

Activities

1. Below is a Node Class definition for the linked list. Add this to a file LinkedList.py.

class Node:

def \_\_init\_\_(self, item, \_next = None):

self.item = item

self.\_next = \_next

def \_\_repr\_\_(self):

return f"Node({self.item}, {self.\_next} )"

def \_\_iter\_\_(self):

yield self.item

if self.\_next is not None:

yield from self.\_next

1. Below is a shell for the LinkedList Class (note that it continues on the next page). Add this to LinkedList.py.

class LinkedList:

def \_\_init\_\_(self, items = None):

self.\_head = None

self.\_tail = None

self.\_len = 0

if items is not None:

for item in items:

self.addlast(item)

def addfirst(self, item):

# You need to develop this method

def addlast(self, item):

# You need to develop this method

def removefirst(self):

# You need to develop this method

def removelast(self):

# You need to develop this method

def \_\_len\_\_(self):

return self.\_len

def \_\_iter\_\_(self):

if self.\_head is not None:

yield from self.\_head

1. Develop the following methods above to your LinkedList that has O(1) except for remove\_last(). That will be O(*n*) :
   1. add\_first(value)
   2. add\_last(value)
   3. remove\_first()
   4. remove\_last()
2. Develop and override the \_\_str\_\_() method so that you can display the linked list as a list of nodes (see override to a node class).
3. Provide a listing of LinkedList.py below this bullet.

class Node:  
 *""""Class to define a node in a linked list"""* def \_\_init\_\_(self, item, \_next=None):  
 *""""Constructor of the Node, builds the item (data) and the link to the next node \_next"""* self.item = item  
 self.\_next = \_next  
  
 def \_\_repr\_\_(self):  
 *"""Returns the Node data and what it is pointing to"""* return f"Node({self.item}, {self.\_next} )"  
  
 def \_\_iter\_\_(self):  
 *""""Allows for the iteration over Nodes"""* yield self.item  
 if self.\_next is not None:  
 yield from self.\_next  
  
  
class LinkedList:  
 *"""Class defining the Linked List ADT and her methods"""* def \_\_init\_\_(self, items=None):  
 *"""Initialise the LinkedList with a head, tail and length."""* self.\_head = None  
 self.\_tail = None  
 self.\_length = 0  
  
 if items is not None:  
 for item in items:  
 self.addlast(item)  
  
 def addfirst(self, item):  
 *""""Adds a new node at the beginning of the linked list."""* self.\_head = Node(item, self.\_head)  
 if self.\_tail is None:  
 self.\_tail = self.\_head  
 self.\_length += 1  
  
 def addlast(self, item):  
 *"""Adds a new node at the end of the linked list. Enqueue"""*  
 if self.\_head is None:  
 self.addfirst(item)  
 else:  
 self.\_tail.\_next = Node(item)  
 self.\_tail = self.\_tail.\_next  
 self.\_length += 1  
  
 def removefirst(self):  
 *"""Removes the first node from the linked list."""*  
 if self.\_head is None:  
 return None # or raise an exception  
 item = self.\_head.item  
 self.\_head = self.\_head.\_next  
 if self.\_head is None:  
 self.\_tail = None  
 self.\_length -= 1  
 return item  
  
 def removelast(self):  
 *"""Removes the last node from the linked list"""* if self.\_head is None or self.\_head.\_next is None:  
 return self.removefirst()  
  
 currentnode = self.\_head  
 while currentnode.\_next.\_next is not None:  
 currentnode = currentnode.\_next  
 item = self.\_tail.item  
 self.\_tail = currentnode  
 self.\_tail.\_next = None  
 self.\_length -= 1  
 return item  
  
 def \_\_str\_\_(self):  
 *"""Formats the str magic method to return human-readable representation of linked list"""* string = 'Your linked list contains: '  
 currentnode = self.\_head  
 while currentnode is not None:  
 string += str(currentnode.item)  
 currentnode = currentnode.\_next  
 if currentnode is not None:  
 string += " ~and~ "  
 return string  
  
 def \_\_len\_\_(self):  
 *"""Returns length of the linked list"""* return self.\_length  
  
 def \_\_iter\_\_(self):  
 *"""Modifies the iter magic method to allow for iteration on linked list"""* if self.\_head is not None:  
 yield from self.\_head  
  
 def \_\_repr\_\_(self):  
 *"""Returns a more basic representation of the linked list"""* items = []  
 for item in self:  
 items.append(str(item))  
 return f"LinkedList({items})"

1. Create a test file TestLinkedList.py using unittest to test your class definitions. Be sure to check the end cases. Provide a listing of TestLinkedList.py below this bullet.

import unittest  
from LinkedList import LinkedList  
  
  
class TestLinkedList(unittest.TestCase):  
  
 def test\_addfirst(self):  
 *"""Test for adding a node to the beginning of a Linked List"""* ll = LinkedList()  
 ll.addfirst(1)  
 self.assertEqual(repr(ll),"LinkedList(['1'])")  
  
 def test\_addlast(self):  
 *"""Tests for adding a node to the end of a Linked List"""* ll = LinkedList()  
 ll.addlast(5)  
 self.assertEqual(repr(ll), "LinkedList(['5'])")  
  
 def test\_removefirst(self):  
 *"""Tests for removing the first node of a Linked List"""* ll = LinkedList()  
 ll.addfirst(1)  
 ll.addfirst(2)  
 removed\_item = ll.removefirst()  
 self.assertEqual(removed\_item, 2)  
 self.assertEqual(repr(ll), "LinkedList(['1'])")  
  
 # Test removing from an empty list  
 ll.removefirst()  
 self.assertEqual(repr(ll), "LinkedList([])")  
 self.assertIsNone(ll.removefirst())  
  
 def test\_removelast(self):  
 *"""Tests removing the last node of a Linked List"""* ll = LinkedList()  
 ll.addfirst(1)  
 ll.addfirst(2)  
 removed\_item = ll.removelast()  
 self.assertEqual(removed\_item, 1)  
 self.assertEqual(repr(ll), "LinkedList(['2'])")  
  
 # Test removing from an empty list  
 ll.removelast()  
 self.assertEqual(repr(ll), "LinkedList([])")  
 self.assertIsNone(ll.removelast())  
  
 def test\_length(self):  
 *"""Tests for the length of the Linked List"""* ll = LinkedList()  
 self.assertEqual(len(ll), 0)  
 ll.addfirst(1)  
 self.assertEqual(len(ll), 1)  
 ll.addlast(2)  
 self.assertEqual(len(ll), 2)  
 ll.removefirst()  
 self.assertEqual(len(ll), 1)  
 ll.removelast()  
 self.assertEqual(len(ll), 0)  
  
 def test\_str\_and\_repr\_consistency(self):  
 *"""Test to show consistent behavior of repr and str for the Linked List"""* ll = LinkedList([1, 2, 3])  
 expected\_repr = "LinkedList(['1', '2', '3'])"  
 self.assertEqual(repr(ll), expected\_repr)  
 expected\_str = 'Your linked list contains: 1 ~and~ 2 ~and~ 3'  
 self.assertEqual(str(ll), expected\_str)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

Supplemental Information

In python, default arguments for functions are only initialized once on the first call. This means we should not use a mutable argument as a default, since it can change over time:

**class** Foo:

**def** \_\_init\_\_(self, L**=**[]): *# Empty list is a BAD default*

self.L **=** L

...

x **=** Foo() *# x.L is the default empty list*

y **=** Foo() *# y.L is the SAME default list*

x.L.append(3)

print(x.L)

print(y.L)

**>>>** [3]

**>>>** [3]

If we want to make a custom collection with an optional collection of arguments, we should use an immutable like None for our default list, and create an empty list on the fly *inside*of the constructor method:

**class** Bar:

**def** \_\_init\_\_(self, L**=**None):

**if** L **is** None:

self.L **=** [] *# new empty list created for every object*

**else**:

self.L **=** L *# whatever the user passed in*

...

x **=** Bar()

y **=** Bar()

x.L.append(3)

**print**(x.L)

print(y.L)

**>>>** [3]

**>>>** []