Note: 2 out of 4 lab assignments will be graded and counted towards 20% of the course. Python is the ONLY accepted programming language for this course.

WARNING: disciplinary actions (zero mark for the lab, or immediate failure of the course, or academic warning) will be taken for any plagiarism.

Due time: Thursday, 15th Feb, 23:59PM through NTULearn -> MH1402 -> Labs -> LAB1 Submission. You may submit multiple times, only the last version counts.

Introduction to class in Python

In this and all upcoming labs, we are to use "class" to present the data structures. "Class" allows users to define new types of objects, bundling data and functionality together. In python, the definition of class starts with a key word class followed by the name of this class ("Node" here as an example). The data of the class are referred by self.dataName inside the class or Instance.dataName from instances of the class, and they can be assigned to default values by the __init__ function, which will be called automatically when instancing a class. For example, the statement n = Node('x') creates an object/instance n of class Node, and assigns the value of n.data to 'x'. n contains two other variables n.next and n.prev. Other functions like getNext, setNext defines how the variables data, next, prev of the class can be retrieved and modified. Note, when calling class functions, self is omitted from the list of provided parameters, e.g., n.getData() (rather than n.getData(self)) will return the value of n.data, and n.setData(newdata) (rather than n.setData(self, newdata)) will assign the value of n.data to newdata.

An example of the "Node" class used in the linked list is given below

NodeClass.py

```
class Node:
  def __init__(self,initdata):
    self.data = initdata
    self.next = None
    self.prev = None

  def getData(self):
    return self.data

  def getNext(self):
    return self.next

  def getPrev(self):
    return self.prev

  def setData(self,newdata):
    self.data = newdata

  def setNext(self,newnext):
    self.next = newnext

  def setPrev(self,newPrev):
    self.prev = newPrev
```

Task:

1. Implement the Index-Based List using array within a class named “IBList” and save the implementation in a file named “IBListClass.py”. The IBList class should support the following functions:

get(r):	Return the element of S with index r; an error condition occurs if $r < 0$ or $r > n-1$.
set(r, e):	Replace with e the element at index r and return it; an error condition occurs if $r < 0$ or $r > n-1$.
add(r, e):	Insert a new element e into S to have index r; an error condition occurs if $r < 0$ or $r > n$.
remove(r):	Remove from S the element at index r; an error condition occurs if $r < 0$ or $r > n-1$.

One can use the available del and insert functions of List. To test the correctness of your implementation, you can instantiate the class and follow the calls and check the content of the List after each call, following the example in Lecture Notes:

Method	Return Value	List Contents
add(0, A)	–	(A)
add(0, B)	–	(B, A)
get(1)	A	(B, A)
set(2, C)	“error”	(B, A)
add(2, C)	–	(B, A, C)
add(4, D)	“error”	(B, A, C)
remove(1)	A	(B, C)
add(1, D)	–	(B, D, C)
add(1, E)	–	(B, E, D, C)
get(4)	“error”	(B, E, D, C)
add(4, F)	–	(B, E, D, C, F)
set(2, G)	D	(B, E, G, C, F)
get(2)	G	(B, E, G, C, F)

Sample procedure is as follow:

```
list = IBList()          # create an instance named ‘list’ of class IBList
list.add(0, ‘A’)         # call the add function
list.data                # check the output of the data, if the array is named data in the class
```

2. Implement the doubly Linked List in a class named “LinkedList”, with each node being instance of the “Node” class. Store your implementation in a file named “LinkedListClass.py”. The LinkedList class should support the following:

first( ):	Returns the position of the first element of L (or null if empty).
last( ):	Returns the position of the last element of L (or null if empty).
before(p):	Returns the position of L immediately before position p (or null if p is the first position).
after(p):	Returns the position of L immediately after position p (or null if p is the last position).
isEmpty( ):	Returns true if list L does not contain any elements.
size( ):	Returns the number of elements in list L.
insertBefore(p, e):	Insert a new element e into S before position p in S.
insertAfter(p, e):	Insert a new element e into S after position p in S.
remove(p):	Remove from S the element at position p.

To test the correctness of your implementation, by creating 5 Node instances, a, b, c, d, e and link a,b,c in sequence, and test:

- Whether the first() function returns the Node a
- Whether the last() function returns the Node c
- What is the node before b
- What is the node after b
- Is it empty
- insertBefore(b, d)
- insertAfter(b, e)
- remove(b)
- print all nodes
- check size

Hint: inside the LinkedList class, you need a variable to keep track of the first node, another variable to keep track of the last node, and a counter to keep track of the total number of nodes. To view all nodes in the linked list, one can also implement an printAllNodes() function to iterate the entire linked list and print the data of each node one by one.

3. Comment each line of your code.