NAMAL UNIVERSITY MIANWALI
DEPARTMENT OF ELECTRICAL ENGINEERING

DATA STRUCTURE AND ALGORITHM
LAB # 05
REPORT
Title : Lists in Python

Name	*Fahim-Ur-Rehman Shah*
Roll No	*NIM-BSEE-2021-24*
Intructor	*Ms. Naureen Shaukat*
Lab Engineer	*Mr .Ali Hasnain*
Date	*27-April-2023*
Marks	

In previous lab, we have studied the concept of arrays in python. We have studied types of arrays (referential array, compact array, dynamic array, etc.) how to make an array and perform different operations (traversing, updating, removing, etc.) on arrays.

Lab Objectives :

The objective of this lab is to introduce students to the concept of lists in python. In this lab students will enable the concepts of making a linked list, types of linked list, accessing the elements of linked list and operations which can be performed on linked list. Students will be provided with examples, followed by performing lab tasks.

Instructions

1. This is an individual lab. You will perform the tasks individually and submit a report.
2. Some of these tasks (marked as 'Example') are for practice purposes only while others (marked as 'Task') have to be answered in the report.
3. When asked to display an output in the task, either save it as jpeg or take a screenshot, in order to insert it in the report.
4. The report should be submitted on the given template, including:
    a. Code (copy and pasted, NOT a screenshot)
    b. Output figure (as instructed in 3)
    c. Explanation where required
5. The report should be properly formatted, with easy to read code and easy to see figures.
6. Plagiarism or any hint thereof will be dealt with strictly. Any incident where plagiarism is caught, both (or all) students involved will be given zero marks, regardless of who copied whom. Multiple such incidents will result in disciplinary action being taken.
7. Late submission of report is allowed within 03 days after lab with 20% deduction of marks every day.
8. You have to submit report in pdf format (Reg.X_DSA_LabReportX.pdf).

Task 1: Write a Python program to make a singly linked list using above class in example 1. Make a method (add_node) to insert a data of your choice after 1 st node of list. Also make a method (delete_node) to delete the data from tail of list. Also make a method (print_list) to print the list after every step.

Python Code:

```
# Define a Node class to represent each node in the linked list
class Node:
    def __init__(self, data):
        # Each node has a piece of data and a reference to the next node
        self.data = data
        self.next = None

# Define a LinkedList class to manage the linked list
class LinkedList:
    def __init__(self):
        # Initialize the head of the list to None
        self.head = None

    def append(self, data):
        # Create a new node with the given data
```

```
        new_node = Node(data)

        # If the list is empty, set the new node as the head of the list
        if self.head is None:
            self.head = new_node
            return

        # If the list is not empty, traverse to the end of the list and add the new
node
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node

    def add_node(self, data):
        # Create a new node with the given data
        new_node = Node(data)

        # If the list is empty, set the new node as the head of the list
        if self.head is None:
            self.head = new_node
            return

        # If the list is not empty, insert the new node after the first node
        first_node = self.head
        new_node.next = first_node.next
        first_node.next = new_node

    def delete_node(self):
        # If the list is empty, do nothing
        if self.head is None:
            return

        # If the list has only one node, set the head to None
        if self.head.next is None:
            self.head = None
            return

        # If the list has multiple nodes, traverse to the second-to-last node and set
its next to None
        second_last_node = self.head
        while second_last_node.next.next:
            second_last_node = second_last_node.next
        second_last_node.next = None

    def print_list(self):
        # Traverse the list starting from the head and print each node's data
        current_node = self.head
        while current_node:
            print(current_node.data, end=" ")
```

```
            current_node = current_node.next
        print()

# Create a new linked list and append some nodes
my_list = LinkedList()
my_list.append(11)
my_list.append(12)
my_list.append(13)

# Print the initial list
print("Initial list:")
my_list.print_list()

# Add a new node after the first node
my_list.add_node(14)
print("After adding a node:")
my_list.print_list()

# Delete the last node
my_list.delete_node()
print("After deleting a node:")
my_list.print_list()
```

Output Screen Shot:

```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 05> python -u "e:\Semeste
r 4\Data Structure and Algorithm\Lab\Lab 05\Task_01.py"
Initial list:
11 12 13
After adding a node:
11 14 12 13
After deleting a node:
11 14 12
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 05>
```

Explanation:

This Python code demonstrates how to use a singly linked list to manage a list of nodes. The code defines two classes: Node and LinkedList. Each Node represents a single node in the linked list and has two attributes: data, which stores the value of the node, and next, which references the next node in the list. The LinkedList class is responsible for managing the linked list and provides four methods: append, which adds a new node to the end of the list, add_node, which inserts a new node after the first node, delete_node, which removes the last node from the list, and print_list, which prints the data of all the nodes in the list. The code creates a new linked list, adds three nodes to it, inserts a node after the first node, deletes the last node, and prints the list at each stage to demonstrate the functionality of the linked list.

Task 2: Write a Python program to make a circular linked list using above class in example 2. Make a method (manipulate_node) to manipulate a data of your choice at central node of list. Also make a method (delete_current_node) to delete the data from current node of list. Also make a method (print_list) to print the list after every step.

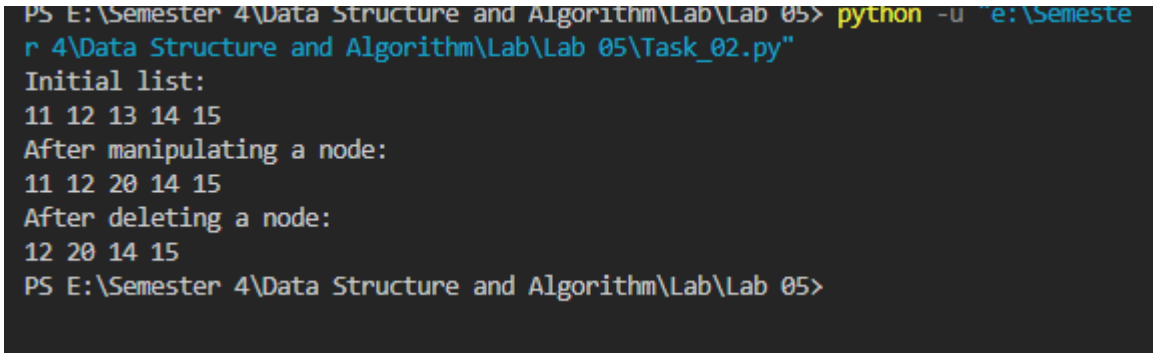
Python Code:

```
# Define a Node class to represent each node in the linked list
class Node:
    def __init__(self, data):
        # Each node has a piece of data and a reference to the next node
        self.data = data
        self.next = None

# Define a LinkedList class to manage the linked list
class LinkedList:
    def __init__(self):
        # Initialize the head of the list to None
        self.head = None

    def append(self, data):
        # Create a new node with the given data
        new_node = Node(data)

        # If the list is empty, set the new node as the head of the list and make it
circular
        if self.head is None:
            self.head = new_node
            new_node.next = new_node
            return

        # If the list is not empty, traverse to the end of the list and add the new
node
        last_node = self.head
        while last_node.next != self.head:
            last_node = last_node.next
        last_node.next = new_node
        new_node.next = self.head

    def manipulate_node(self, new_data):
        # If the list is empty, do nothing
        if self.head is None:
            return

        # Traverse the list to find the central node (if there are an even number of
nodes, use the second node from the middle as the central node)
        slow_ptr = self.head
        fast_ptr = self.head
        while fast_ptr.next != self.head and fast_ptr.next.next != self.head:
            slow_ptr = slow_ptr.next
            fast_ptr = fast_ptr.next.next
```

```
        # Manipulate the data of the central node
        slow_ptr.data = new_data

    def delete_current_node(self):
        # If the list is empty, do nothing
        if self.head is None:
            return

        # If the list has only one node, set the head to None
        if self.head.next == self.head:
            self.head = None
            return

        # Traverse the list to find the current node (the node before the head)
        current_node = self.head
        while current_node.next != self.head:
            current_node = current_node.next

        # Delete the current node by setting its next to the head and updating the
head
        current_node.next = self.head.next
        self.head = current_node.next

    def print_list(self):
        # Traverse the list starting from the head and print each node's data
        if self.head is None:
            print("List is empty")
            return
        current_node = self.head
        while True:
            print(current_node.data, end=" ")
            current_node = current_node.next
            if current_node == self.head:
                break
        print()

# Create a new circular linked list and append some nodes
my_list = LinkedList()
my_list.append(11)
my_list.append(12)
my_list.append(13)
my_list.append(14)
my_list.append(15)

# Print the initial list
print("Initial list:")
my_list.print_list()

# Manipulate the central node
```

```
my_list.manipulate_node(20)
print("After manipulating a node:")
my_list.print_list()

# Delete the current node
my_list.delete_current_node()
print("After deleting a node:")
my_list.print_list()
```

Output Screen Shot:



Explanation

This is a Python code that implements a circular linked list data structure with the ability to manipulate and delete nodes. The code defines two classes: Node and LinkedList. The Node class represents each node in the linked list and has two attributes: data to store the value of the node and next to reference the next node in the list. The LinkedList class manages the linked list and has four methods: append to add a new node to the end of the list, manipulate_node to modify the data of the central node, delete_current_node to remove the current node from the list, and print_list to print the data of all the nodes in the list. The code creates a new circular linked list, appends five nodes to it, manipulates the central node, deletes the current node, and prints the list at each stage.

Task 3: Write a Python program to make a doubly linked list using above class in example 3. Make a method (next_of_node) to add a new node containing data to the front of the list and a method (prev_of_node) to add a new node containing data to the back of the list. Also make a method (remove_next) to remove and return the data from the first node in the list and a method (remove_prev) to remove and return the data from the last node in the list. Also make a method (print_list) to print the list after every step.

Python Code:

```
# Define a class for the doubly linked list node
class Node_of_doubly_linked_list:
    def __init__(self, data):
        self.data = data
        self.next = None
```

```
        self.prev = None

# Define a class for the doubly linked list
class a_Doubly_Linked_List:
    def __init__(self):
        self.head = None
        self.tail = None

    # Method to add a new node with data to the front of the list
    def next_of_node(self, data):
        new_node = Node_of_doubly_linked_list(data)
        if self.head is None:
            self.head = self.tail = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node
        self.print_list()

    # Method to add a new node with data to the back of the list
    def prev_of_node(self, data):
        new_node = Node_of_doubly_linked_list(data)
        if self.tail is None:
            self.head = self.tail = new_node
        else:
            new_node.prev = self.tail
            self.tail.next = new_node
            self.tail = new_node
        self.print_list()
    # Method to remove and return data from the first node in the list
    def remove_next(self):
        if self.head is None:
            return None
        data = self.head.data
        self.head = self.head.next
        if self.head is not None:
            self.head.prev = None
        else:
            self.tail = None
        self.print_list()
        return data
    # Method to remove and return data from the last node in the list
    def remove_prev(self):
        if self.tail is None:
            return None
        data = self.tail.data
        self.tail = self.tail.prev
        if self.tail is not None:
            self.tail.next = None
        else:
```

```
            self.head = None
        self.print_list()
        return data

    # Method to print the list
    def print_list(self):
        current_node = self.head
        print("List contents: ", end="")
        while current_node is not None:
            print(current_node.data, end=" ")
            current_node = current_node.next
        print()
# Example usage
my_list = a_Doubly_Linked_List()

my_list.prev_of_node(11)
my_list.next_of_node(10)
my_list.prev_of_node(12)
my_list.next_of_node(9)

my_list.remove_next()
my_list.remove_prev()

my_list.prev_of_node(5)
my_list.next_of_node(6)
```

Output Screen Shot:

```
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 05> python -u "e:\Semeste
r 4\Data Structure and Algorithm\Lab\Lab 05\Task_03.py"
List contents: 11
List contents: 10 11
List contents: 10 11 12
List contents: 9 10 11 12
List contents: 10 11 12
List contents: 10 11
List contents: 10 11 5
List contents: 6 10 11 5
PS E:\Semester 4\Data Structure and Algorithm\Lab\Lab 05>
```

Explanation

The provided code includes the implementation of a doubly linked list in Python using two classes: **Node_of_doubly_linked_list** and a_**Doubly_Linked_List**. The first class defines the structure of a node in the linked list, including the data, the reference to the next node and the reference to the previous node. The second class defines the doubly linked list itself, including the head and tail nodes, as well as several methods to add, remove, and print the list. Example usage is provided at the end of the code, which demonstrates how to add nodes to the front and back of the list, remove nodes from the front and back of the list, and print the contents of the list.

Lab Evaluation Rubrics							
Domain	**CLOs/ Rubric**	**Performance Indicator**	**Unsatisfactory 0-5**	**Marginal 5-10**	**Satisfactory 11-15**	**Exemplary 16-20**	**Allocated Marks**
Psychomotor	**CLO:1 R2**	Implementation with Results **(P)**	Does not try to solve problems. Many mistakes in code and difficult to comprehend for the instructor. There is not result of the problem.	Does not suggests or refine solutions but is willing to try out solutions suggested by others. Few mistakes in code, but done along with comments, and easy to comprehend for the instructor. Few mistake in result.	Refines solutions suggested byothers. Complete and error-free code is done. No comments in the code, but easy to comprehend for the instructor. Results are correctly produced.	Actively looks for and suggests solution to problems. Complete and error free code is done, easy to comprehend forthe instructor. Results are correctly produced. Student incorporated comments in the code.	
Affective	**CLO:3 R3**	Lab Report **(A)**	Code of the problem is not given. Outputs are not provided. Explanation of the solution is not stated.	Code of the problem is not given. Output is not complete. Explanation of the solution is not satisfactory.	Code of the problem is not given. Output is completely given. Explanation of the solution is not satisfactory.	Code of the problem is not given. Output is completely given. Explanation of the solution is satisfactory.	
	CLO:1 R5	Discipline and Behavior **(A)**	Got and wandered around.Chased others, ran, or played around. More than two incidents of talking non-lab related stuff in laband/or any talk with other groups, voice level exceeding the appropriate level, use of cell phones and involvement in any non lab activity.	Got out of seat and wander around for some time. No more than two incidents of talking non-lab related stuff inlab. Voice level exceeding theappropriate level, use of cell phones and involvement in any non-lab related activity.	Stayed in seat and got up for a specific lab related reason, but took more time than required to do the job. No more than one incidents of talking non-lab related stuff in lab. Voice level exceedingthe appropriate level, use of cell phones and involvementin any non-lab related activity.	Stayed in seat and got up for a specific lab related reason. Tookcare of lab related business and sat down right away. Voice level kept appropriate. Not used cell phones or involved in any non- lab related activity.	