Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

JUNAID GIRKAR
60004190057
TE COMPS A4

EXPERIMENT - 3

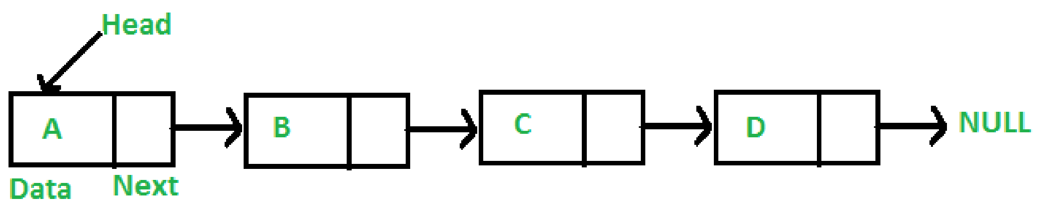
Aim: Menu driven program for data structure using built in function for linked list, stack and queues

Theory:
Linked List

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

- Link − Each link of a linked list can store a data called an element.
- Next − Each link of a linked list contains a link to the next link called Next.
- LinkedList − A Linked List contains the connection link to the first link called First.

Linked List Representation
Linked list can be visualised as a chain of nodes, where every node points to the next node.



As per the above illustration, following are the important points to be considered.

- Linked List contains a link element called first.
- Each link carries a data field(s) and a link field called next.
- Each link is linked with its next link using its next link.
- Last link carries a link as null to mark the end of the list.

Types of Linked List
Following are the various types of linked list.

- **Simple Linked List** − Item navigation is forward only.
- **Doubly Linked List** − Items can be navigated forward and backward.
- **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

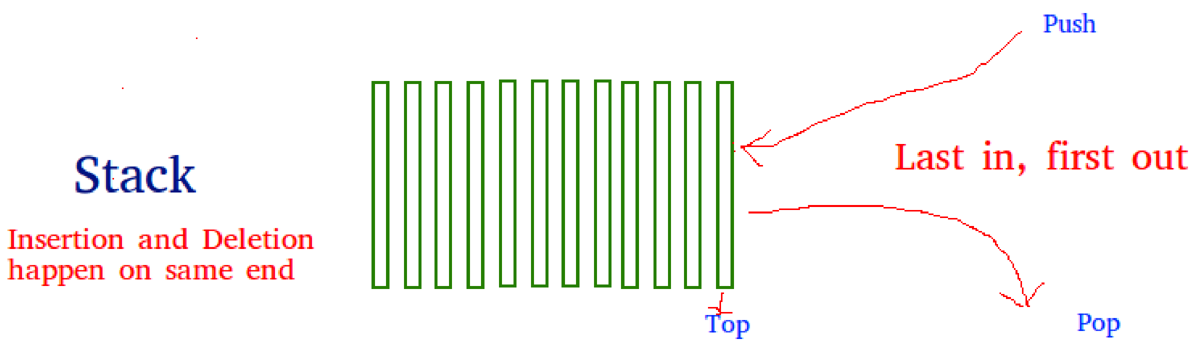
Basic Operations
Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Display** − Displays the complete list.
- **Search** − Searches an element using the given key.
- **Delete** − Deletes an element using the given key.

Code:

```python
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:

    def __init__(self):
        self.start = None

    def insert_beg(self, data):
        y = Node(data)
        if self.start is None:
            self.start = y
        else:
            y.next = self.start
            self.start = y
    def insert_after(self, data, val):
        y = Node(data)
        current = self.start
        while current.next is not None and current.data is not val:
            current = current.next
        if current.data is val:
            y.next = current.next
            current.next = y
        else:
            print("Value " + str(val) + " not found.")

    def delete(self, data):
        current = self.start
        if self.start.data is data:
            self.start = self.start.next
            del current
            return
        while current.next is not None and current.data is not data:
            preptr = current
            current = current.next
        if current.data is data:
            preptr.next = current.next
            del current
        else:
            print("Value " + str(data) + " not found.")

    def display(self):
        print("Linked List: ")
        current=self.start
        while current is not None:
            print(f"Data:{current.data}")
            current=current.next



def exec():
    x = LinkedList()
    option = 0
    while option != 5:
        print("\nEnter an option:\n1)Insert at Beg\n2)Insert after a 
particular value")
        print("3)Delete a particular value\n4)Display\n5)Exit\n")
        option = int(input())
        if option == 1:
            data = int(input("Enter the element: "))
            x.insert_beg(data)
        elif option == 2:
            val = int(input("Enter the value after which data is to be
inserted: "))
            data = int(input("Enter the element: "))
            x.insert_after(data, val)
        elif option == 3:
            data = int(input("Enter the value to be deleted: "))
            x.delete(data)
        elif option == 4:
            x.display()
        elif option != 5:
            print("Invalid entry. Retry")
```

STACK:
A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle. Stack has one end, whereas the Queue has two ends (front and rear). It contains only one pointer top pointer pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.



Standard Stack Operations
The following are some common operations implemented on the stack:

- **push**(): When we insert an element in a stack then the operation is known as a push. If the stack is full then the overflow condition occurs.

- **pop**(): When we delete an element from the stack, the operation is known as a pop. If the stack is empty means that no element exists in the stack, this state is known as an underflow state.
- **isEmpty**(): It determines whether the stack is empty or not.
- **isFull**(): It determines whether the stack is full or not.'
- **peek**(): It returns the element at the given position.
- **count**(): It returns the total number of elements available in a stack.
- **change**(): It changes the element at the given position.
- **display**(): It prints all the elements available in the stack.

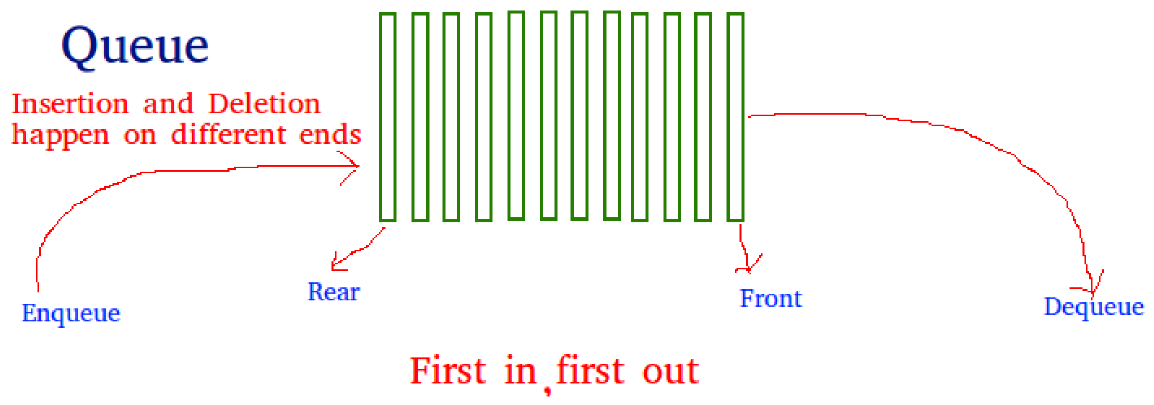
Code:

```
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None


class Stack:

    def __init__(self):
        self.top = None

    def push(self, data):
        y = Node(data)
        if self.top is None:
            self.top = y
        else:
            y.next = self.top
            self.top = y

    def pop(self):
        if self.top is None:
            print("Stack Underflow")
            return -1
        else:
            val = self.top.data
            cur = self.top
            self.top = self.top.next
            del cur
            return val

    def peek(self):
        if self.top is None:
            print("Empty.")
            return
        else:
            print("Top of Stack: " + str(self.top.data))
            return

    def display(self):
        print("Stack: ")
        current = self.top
        if self.top is not None:
            print(self.top.data, end=' ')
        else:
            print("Empty.")
        while current.next is not None:
            print("<-", end=' ')
            print(current.next.data, end=' ')
            current = current.next



def exec():
    x = Stack()
    option = 0
    while option is not 5:
        print("\nEnter an option:\n1)Push\n2)Pop")
        print("3)Peek\n4)Display\n5)Exit\n")
        option = int(input())
        if option is 1:
            data = int(input("Enter the element: "))
            x.push(data)
        elif option is 2:
            y = x.pop()
            if y is not -1:
                print("Popped Element: " + str(y))
        elif option is 3:
            x.peek()
        elif option is 4:
            x.display()
        elif option is not 5:
            print("Invalid entry. Retry")
```

QUEUE:
Queue is the data structure that is similar to the queue in the real world. A queue is a data structure in which whatever comes first will go out first, and it follows the FIFO (First-In-First-Out) policy. Queue can also be defined as the list or collection in which the insertion is done from one end known as the rear end or the tail of the queue, whereas the deletion is done from another end known as the front end or the head of the queue.

The real-world example of a queue is the ticket queue outside a cinema hall, where the person who enters first in the queue gets the ticket first, and the last person enters in the queue gets the ticket at last. Similar approach is followed in the queue in data structure.

The representation of the queue is shown in the below image -



Types of Queue
There are four different types of queue that are listed as follows -

- Simple Queue or Linear Queue
- Circular Queue
- Priority Queue
- Double Ended Queue (or Deque)

Operations performed on queue
The fundamental operations that can be performed on queue are listed as follows -

- **Enqueue**: The Enqueue operation is used to insert the element at the rear end of the queue. It returns void.
- **Dequeue**: It performs the deletion from the front-end of the queue. It also returns the element which has been removed from the front-end. It returns an integer value.
- **Peek**: This is the third operation that returns the element, which is pointed by the front pointer in the queue but does not delete it.

- **Queue overflow (isfull):** It shows the overflow condition when the queue is completely full.
- **Queue underflow (isempty):** It shows the underflow condition when the Queue is empty, i.e., no elements are in the Queue.

Code:

```
class Node:

    def __init__(self, data):
        self.data = data
        self.next = None


class Queue:

    def __init__(self):
        self.front = None
        self.rear = None

    def enqueue(self, data):
        y = Node(data)
        if self.rear is None:
            self.front = y
            self.rear = y
        else:
            self.rear.next = y
            self.rear = y

    def dequeue(self):
        if self.front is None:
            print("Queue Underflow")
            return -1
        elif self.front is self.rear:
            val = self.front.data
            cur = self.front
            self.front = None
            self.rear = None
        else:
            val = self.front.data
            cur = self.front
            self.front = self.front.next
        del cur
        return val
    def display(self):
        print("Queue: ")
        current = self.front
        if self.front is not None:
            print(self.front.data, end=' ')
        else:
            print("Empty")
            return
        while current.next is not None:
            print("->", end=' ')
            print(current.next.data, end=' ')
            current = current.next



def exec():
    x = Queue()
    option = 0
    while option is not 4:
        print("\nEnter an option:\n1)Enqueue\n2)Dequeue")
        print("3)Display\n4)Exit\n")
        option = int(input())
        if option is 1:
            data = int(input("Enter the element: "))
            x.enqueue(data)
        elif option is 2:
            y = x.dequeue()
            if y is not -1:
                print("Dequeued Element: " + str(y))
        elif option is 3:
            x.display()
        elif option is not 4:
            print("Invalid entry. Retry")
```

OUTPUT:

LinkedList

```
Enter 1 for Linked List
Enter 2 for stack
Enter 3 for Queue
Enter Input: 1
Enter an option:
1)Insert at Beg
2)Insert after a particular value
3)Delete a particular value
4)Display
5)Exit

4
Linked List:

Enter an option:
1)Insert at Beg
2)Insert after a particular value
3)Delete a particular value
4)Display
5)Exit

1
Enter the element: 23

Enter an option:
1)Insert at Beg
2)Insert after a particular value
3)Delete a particular value
4)Display
5)Exit

2
Enter the value after which data is to be inserted: 23
Enter the element: 65

Enter an option:
1)Insert at Beg
2)Insert after a particular value
3)Delete a particular value
4)Display
5)Exit

4
Linked List:
Data:23
Data:65
```

```
Enter an option:
1)Insert at Beg
2)Insert after a particular value
3)Delete a particular value
4)Display
5)Exit

3
Enter the value to be deleted: 23

Enter an option:
1)Insert at Beg
2)Insert after a particular value
3)Delete a particular value
4)Display
5)Exit

4
Linked List:
Data:65

Enter an option:
1)Insert at Beg
2)Insert after a particular value
3)Delete a particular value
4)Display
5)Exit

5
```

Stack:

```
Enter 1 for Linked List
Enter 2 for stack
Enter 3 for Queue
Enter Input: 2

Enter an option:
1)Push
2)Pop
3)Peek
4)Display
5)Exit
```

```
1
Enter the element: 23

Enter an option:
1)Push
2)Pop
3)Peek
4)Display
5)Exit

1
Enter the element: 45

Enter an option:
1)Push
2)Pop
3)Peek
4)Display
5)Exit

4
Stack:
45 <- 23
Enter an option:
1)Push
2)Pop
3)Peek
4)Display
5)Exit

2
Popped Element: 45

Enter an option:
1)Push
2)Pop
3)Peek
4)Display
5)Exit

3
Top of Stack: 23
```

```
Enter an option:
1)Push
2)Pop
3)Peek
4)Display
5)Exit

5
```

Queue:

```
Enter 1 for Linked List
Enter 2 for stack
Enter 3 for Queue
Enter Input: 3

Enter an option:
1)Enqueue
2)Dequeue
3)Display
4)Exit

1
Enter the element: 23

Enter an option:
1)Enqueue
2)Dequeue
3)Display
4)Exit

1
Enter the element: 46

Enter an option:
1)Enqueue
2)Dequeue
3)Display
4)Exit

1
Enter the element: 78

Enter an option:
```

```
1)Enqueue
2)Dequeue
3)Display
4)Exit

3
Queue:
23 -> 46 -> 78
Enter an option:
1)Enqueue
2)Dequeue
3)Display
4)Exit

2
Dequeued Element: 23

Enter an option:
1)Enqueue
2)Dequeue
3)Display
4)Exit

3
Queue:
46 -> 78
Enter an option:
1)Enqueue
2)Dequeue
3)Display
4)Exit

4
```

Conclusion:
We learnt about the different advanced data structures - Linked list, Stack and Queue. Then we studied every one of them in detail and implemented those structures in a python program.