**Single linked list:**

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

class SingleLinkedList:

    def \_\_init\_\_(self):

        self.head = None

    def display(self):

        if self.head is None:

            print("Linked List is empty")

        else:

            temp = self.head

            while temp:

                print(temp.data, "->",end="")

                temp = temp.next

            print("Null")

    def insert\_at\_beg(self,data):

        new\_node=Node(data)

        new\_node.next=self.head

        self.head = new\_node

    def insert\_at\_mid(self,number,data):

        new\_node=Node(data)

        if self.head is None:

            print("Linked List is empty")

        elif self.head.data==number:

            new\_node.next=self.head.next

            self.head.next=new\_node

        else:

            temp = self.head

            while temp.next:

                if temp.next.data == number:

                    temp.next=new\_node.next

                    temp.next=new\_node

                    break

                temp = temp.next

    def insert\_at\_end(self, data):

        new\_node = Node(data)

        temp = self.head

        while temp.next:

            temp = temp.next

        temp.next = new\_node

    def delete(self, number):

        if self.head is None:

            print("Linked List is empty")

        elif self.head.data == number:

            self.head = self.head.next

        else:

            temp = self.head

            while temp.next:

                if temp.next.data == number:

                    temp.next = temp.next.next

                    break

                temp = temp.next

obj = SingleLinkedList()

while True:

    n = int(input("\n1.Create\n2.Insert at beginning\n3.insert at mid\n4.insert at end\n5.Display\n6.Delete\n7.Exit\nEnter a option:"))

    if n == 1:

        a = int(input("Enter the head value: "))

        n1 = Node(a)

        obj.head = n1

    elif n == 2:

        b = int(input("Enter a element: "))

        obj.insert\_at\_beg(b)

    elif n==3:

        a=int(input("Enter the element after which you want to enter:"))

        b=int(input("Enter a element:"))

        obj.insert\_at\_mid(a,b)

    elif n==4:

        b=int(input("Enter a element:"))

        obj.insert\_at\_end(b)

    elif n == 5:

        obj.display()

    elif n == 6:

        c = int(input("Enter the element you want to delete: "))

        obj.delete(c)

    else:

        break

**Floyd algorithm:**

Floyd algorithm is used to find out whether a linked list contain cycle or not.

**Floyd algorithm:**

* To find out the meeting point if there is cycle assign two pointers to head node.
* One pointer will jump one step, and another pointer will jumps two steps.
* If they are meeting at one node it declares that linked list contain cycle, that node is called as meeting point.

**Finding starting node of a cycle:**

* Assign two pointers one with head node and another with meeting point node.
* Now both of them will jump one step at a time.
* Where they are meeting is called as starting node of the cycle.

**Removing:**

* Find out the node which is connected with starting node of the cycle and next as none.

**Diagram:**

4

20

15

10

500

1

Extra…………

**Circular list:**

* Last node is connected with first node it will become circular the output will be infinite loop.

**Deleting from a linked list:**

1. Deleting the first node
2. Deleting the last node
3. Deleting the middle node

**Deleting the last node:**

* travel till last but before node and make next as null.

**Deleting the first node:**

* make the next node of head node as head node.

**Searching of an element inside the linked list:**

* traverse the whole list and after finding the element exit the loop.

**Code:**

temp = self.head

count = 0

flag=0

            while temp.next:

                if temp.next.data == number:

                    print(“element is found at”,count)

flag=1

                    break

count+=1

                temp = temp.next

if flag==0:

print(“not found”)

**Double linked list:**

class Node:

    def \_\_init\_\_(self,data):

        self.data=data

        self.next=None

        self.prev=None

class DoubleLinkedList:

    def \_\_init\_\_(self):

        self.head=None

    def insert\_at\_beg(self,data):

        n=Node(data)

        if self.head is None:

            self.head=n

        else:

            n.next=self.head

            self.head.prev=n

            self.head=n

    def insert\_at\_end(self,data):

        n=Node(data)

        if self.head is None:

            self.head=n

        else:

            temp=self.head

            while (temp.next!=None):

                temp=temp.next

            temp.next=n

            n.prev=temp

    def insert\_at\_mid(self,loc,data):

        n=Node(data)

        if self.head is None:

            self.head=n

        else:

            temp=self.head

            while temp.data!=loc:

                temp=temp.next

            n.next=temp.next

            temp.next.prev=n

            n.prev=temp

            temp.next=n

    def delete(self,loc):

        if self.head is None:

            print("List Empty")

        else:

            temp=self.head

            while temp.next.data!=loc:

                temp=temp.next

            if temp.next.next is None:

                temp.next=None

            else:

                temp.next.next.prev=temp.next

                temp.next=temp.next.next

    def display(self):

        n=int(input("1.Forward\n2.Backward\nEnter the choice:"))

        if n==1:

            p=self.head

            while p:

                print(p.data,"->",end=" ")

                p=p.next

        if n==2:

            p=self.head

            while p.next is not None:

                p=p.next

            while p:

                print(p.data,"->",end=" ")

                p=p.prev

obj = DoubleLinkedList()

while True:

    n = int(input("\n1.Create\n2.Insert at beginning\n3.insert at mid\n4.insert at end\n5.Display\n6.Delete\n7.Exit\nEnter a option:"))

    if n == 1:

        a = int(input("Enter the head value: "))

        n1 = Node(a)

        obj.head = n1

    elif n == 2:

        b = int(input("Enter a element: "))

        obj.insert\_at\_beg(b)

    elif n==3:

        a=int(input("Enter the element after which you want to enter:"))

        b=int(input("Enter a element:"))

        obj.insert\_at\_mid(a,b)

    elif n==4:

        b=int(input("Enter a element:"))

        obj.insert\_at\_end(b)

    elif n == 5:

        obj.display()

    elif n == 6:

        c = int(input("Enter the element you want to delete from the linked list: "))

        obj.delete(c)

    else:

        break

**Stack:**

* It uses lifo concept which is last-in-first-out.
* We can implement stack using 3 methods they are

1. Arrays
2. Linked lists
3. Modules

* Insertion and deletion happen at one end of the stack which is top.

**Code:**

stack=[]

def push():

    n=int(input("\nenter an element:"))

    stack.append(n)

    display()

def pop():

    if len(stack)==0:

        print("Stack is empty")

    else:

        x=stack.pop()

        print("removed element:",x)

        display()

def display():

    print(stack)

def top():

    if len(stack)==0:

        print("stack is empty")

    else:

        x=stack[-1]

        print("the top element is",x)

while True:

    n=int(input("1.push\n2.pop\n3.display\n4.exit\n5.top\nenter you choice:"))

    if n==1:

        push()

    if n==2:

        pop()

    if n==3:

        display()

    if n==4:

        top()

    if n==5:

        break

**Stack implementation using linked list:**

**Method-1:**

* Inserting node at last and deleting the last node becomes stack.

**Method-2:**

* While using stack with linked list we can opt this method also we can inset at the beginning and deleting the head node.

**Note:**

* In this two methods, method-2 is efficient because in method-1 we need to traverse to last node for inserting or deleting the element.

**Parenthesis problem:**

**Code:**

st=input("enter the string:")

li=[i for i in st]

li2=[]

if len(li)%2!=0:

    print(False)

elif li[0]==')' or li[0]=='}' or li[0]==']':

    print(False)

else:

    for i in range(len(li)):

        if li[i]=='(' or li[i]=='{' or li[i]=='[':

            li2.append(li[i])

        if li[i]==')' and li2[len(li2)-1]=='(':

            li2.pop()

            continue

        if li[i]==']' and li2[len(li2)-1]=='[':

            li2.pop()

            continue

        if li[i]=='}' and li2[len(li2)-1]=='{':

            li2.pop()

            continue

        if len(li2)==0:

            break

    if len(li2)==0:

        print(True)

    else:

        print(False)

**Queue:**

* It can be implemented by using three methods.

1. Lists
2. Linked lists
3. Modules

* It uses fifo method.
* Insertion and delete happens at two different ends, rear and front.

**Applications of stack:**

* One of most important application of stack is expression conversion.

1. Infix
2. Postfix
3. Prefix

**Infix to postfix:**

**Ex: A+B-C\*D+E**

1. **A+B- CD\*+E**
2. **AB+-CD\*+E**
3. **AB+CD\*-+E**
4. **AB+CD\*-E+**

**Ex: X/Y-(A+B)-(C\*D)+Z+(E-F)**

**Ans: XY/AB+-CD\*-Z+EF-+**

**Steps to Implement infix to postfix using stack:**

* Check the priority of the operators .

()->/->\*->-+

* No operators of same priority together in the stack.
* Lower priority operator cannot be place before highest priority operator in the stack.
* If one operator is in between open and close braces in the stack pop it.

**Types of queues:**

1. Priority queue
2. Circular queue

**Priority queue:**

* Jobs will be in a queue however priorities will be assigned.
* As per the priority jobs will be allotted.

**Ex:**

* **T1->3**
* **T2->2**
* **T3->4**
* **T4->1**
* when we enter this jobs into the list they will enter according to their priorities.

T3

T1

T4

T2

**Circular queue:**

10

20

30

40

50

* In normal queue concept even if there is space after deleting items, we would not be able to utilise space.
* In order to use the spaces, we are coming up with this concept circular queue.

1. Initialising circular queue.

Front=rear=1

1. Insertion happens to rear, delete happens at front.
2. Inserting 1st element

Front=rear=0

1. Insertion from 2nd element

Rear=rear+1 and insert

1. Deletion: after deleting front element

Front=front+1

**Applications of SLL:**

* Implementation of stack/queue.
* Redo or undo operations.

**Applications of DLL:**

* Used to represent polynomial
* Used to manipulate polynomial expressions.

1. X+Y+3=0
2. 4X+2Y+9=0

* Implementation of graphs.
* Arithmetic operations with very big nos.
* To implement hierarcial data we use tree ds, in a tree, every node can have ‘n’ no of children.

**Height of a node:**

* Look down & count the levels

**Depth of a node:**

* Look up& count the levels.
* If a parent is having 2 children we call them as left child & right child.
* Tree can be divided into 2 parts:

1. Left sub tree
2. Right sub tree

**Note:** whenever, we performing any operation any operation in tree first complete left sub tree.

**Leaf node:**

A node which doesn’t have children.

**Types of tress:**

1. Binary tree
2. Binary search tree
3. AVL tree
4. Trie tree
5. Heap tree

**Binary tree:**

* Any node can have max of 2 nodes.

**Complete binary tree:**

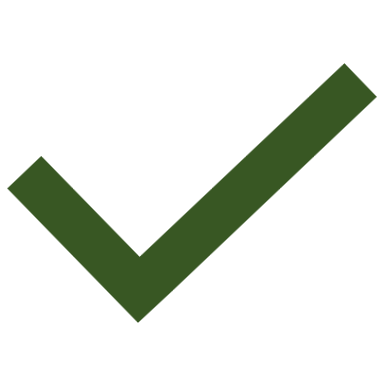
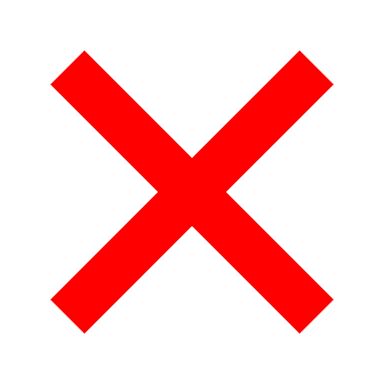
* All levels should be full except for last level. Leaf node should be at almost left arranged.

**Strict binary tree:**

* All nodes should have 2 children or no children.

**Perfect binary tree:**

* A tree in which all interior nodes have 2 children & all leaves have the same depth of same level.



**Tree traversal methods:**

1. Level order traversal
2. Depth order traversal

**Level order traversal:**

* Hou would traverse level by level left to right.

**Depth first search:**

* **In order**
* **Pre order**
* **Post order**

**In order(left, Root, Right): 4 2 5 1 3**

**Preorder (root, left ,right):1 2 4 5 3**

**Post order (left ,right, root):4 5 2 3 1**

**Binary search tree:**

* Every node is lesser than its parent, if it is lying on left side.
* Every node is greater than its parent, if it is lying on right side.