1. **Doubly linked list**

class Node:

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

self.data=data

self.next=None

self.prev=None

class Dlinkedlist:

def \_\_init\_\_(self):

self.head=None

def listprint(self):

ptr=self.head

if (ptr is None):

print("Empty list")

while ptr is not None:

print(ptr.data,end="<-->")

ptr=ptr.next

print()

def search(self,value):

ptr=self.head

if (ptr is None):

print("Empty list")

while ((ptr is not None) and (ptr.data != key)):

ptr=ptr.next

if (ptr is None):

print("Key not found")

else:

print("Key found")

def insert(self,value):

new\_node=Node(value)

if (self.head==None):

list.head=new\_node

else:

self.head.prev=new\_node

new\_node.next=self.head

self.head=new\_node

def delete(self):

if (self.head==None):

print("Empty List")

else:

print("Node value deleted is",self.head.data)

self.head=self.head.next

self.head.prev=None

list=Dlinkedlist()

n=int(input("how many node you want : "))

for i in range(n):

value=int(input("Enter the node value : "))

list.insert(value)

list.listprint()

print("\nDemonstrating search,Insert and delete operation")

key=int(input("Enter the value to search : "))

list.search(key)

print("\nInserting Node value at the beginning")

value=int(input("Enter the value to insert : "))

list.insert(value)

print("\nList after inserting Node")

list.listprint()

print("\nDeleting Node")

list.delete()

print("\nList after deleting first node : ")

list.listprint()

print("------END------")

**Output:**

how many node you want : 4

Enter the node value : 23

Enter the node value : 54

Enter the node value : 12

Enter the node value : 76

76<-->12<-->54<-->23<-->

Demonstrating search,Insert and delete operation

Enter the value to search : 2

Key not found

Inserting Node value at the beginning

Enter the value to insert : 14

List after inserting Node

14<-->76<-->12<-->54<-->23<-->

Deleting Node

Node value deleted is 14

List after deleting first node :

76<-->12<-->54<-->23<-->

------END------

1. **Implementation of stack**

class Stack:

def \_\_init\_\_(self,size):

self.maxsize = size

self.top = -1

self.list = []

def isEmpty(self):

return self.top == -1

def isFull(self):

return self.top == self.maxsize - 1

def push(self,value):

if self.isFull():

print("Cannot add! Stack is full")

else:

self.top += 1

self.list.append(value)

print(value, "has been added sucessfully")

def pop(self):

if self.isEmpty():

print("Stack is empty! underflow")

else:

print("Popped item:", self.list.pop())

self.top -= 1

def display(self):

if self.top == -1:

print("Stack is empty!")

else:

print("Contents of the stack are:")

print(self.list)

#Main program

stk = Stack(3)

print("Enter 1 to push 2 to pop and 3 to display and 4 to stop")

choice = int(input("Input choice:"))

while (choice == 1 or choice == 2 or choice == 3):

if (choice == 1):

item = int(input("Enter item to push:"))

stk.push(item)

elif (choice == 2):

stk.pop()

elif (choice ==3):

stk.display()

else:

break

print("")

print("Enter 1 to push 2 to pop 3 to display 4 to stop")

choice = int(input("input choice"))

print("---End---")

**Output:**

Enter 1 to push 2 to pop and 3 to display and 4 to stop

Input choice:1

Enter item to push:32

32 has been added sucessfully

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice1

Enter item to push:54

54 has been added sucessfully

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice1

Enter item to push:57

57 has been added successfully

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice1

Enter item to push:32

Cannot add! Stack is full

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice2

Popped item: 57

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice2

Popped item: 54

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice2

Popped item: 32

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice2

Stack is empty! underflow

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice3

Stack is empty!

Enter 1 to push 2 to pop 3 to display 4 to stop

input choice4

---End---

1. **Recursive Fibonacci series**

def recur\_fibo(n):

if n<=1:

return n

else:

return(recur\_fibo(n-1)+recur\_fibo(n-2))

nterms =10

if nterms <=0:

print("please enter a positive number ")

else:

print("fibonacci sequence")

for i in range (nterms):

print(recur\_fibo(i))

**Output:**

fibonacci sequence

0

1

1

2

3

5

8

13

21

34

1. **Tower of Hanoi**

def towerofhanoi(n,source,destination,auxiliray):

if n==1:

print("Move disk 1 From source" ,source,"to destination",destination)

return

towerofhanoi(n-1,source,auxiliray,destination)

print("move disk" , n, "from source",source ,"to destination",destination)

towerofhanoi(n-1,source,auxiliray,destination)

n=3

towerofhanoi(n,'a','b','c')

**Output:**

Move disk 1 From source a to destination b

move disk 2 from source a to destination c

Move disk 1 From source a to destination b

move disk 3 from source a to destination b

Move disk 1 From source a to destination b

move disk 2 from source a to destination c

Move disk 1 From source a to destination b

1. **Implementation of queue**

class queue:

def \_\_init\_\_(self,size):

self.capacity = size

self.front = -1

self.q = []

self.rear = -1

def isEmpty(self):

return self.front == -1

def dequeue(self):

if self.isEmpty():

print("queue underflow")

else:

print("removing element ....",self.q[self.front])

if(self.rear ==self.front):

self.front=-1

self.rear=-1

self.q.clear()

else:

self.front=self.front+1

def insertqueue(self):

if(self.rear == self.capacity-1):

print("overflow")

return

else:

value=int(input("enter item to insert:"))

print("inserting element ....",value)

self.rear =self.rear+1

self.q.append(value)

if(self.front == -1):

self.front=0

def display(self):

if self.isEmpty():

print("queue empty")

return

print("the queue is",end ='|')

i=self.front

while i<=self.rear:

print(self.q[i],end='|')

i=i+1

#Main program

q = queue(3)

print("Enter 1 to insert 2 to dequeue and 3 to display and 4 to stop")

choice = int(input("Input choice:"))

while (choice == 1 or choice == 2 or choice == 3):

if (choice == 1):

q.insertqueue()

elif (choice == 2):

q.dequeue()

elif (choice ==3):

q.display()

else:

break

print("")

print("Enter 1 to insert 2 to dequeue 3 to display 4 to stop")

choice = int(input("input choice"))

print("---End---")

**Output:**

Enter 1 to insert 2 to dequeue and 3 to display and 4 to stop

Input choice:1

enter item to insert:32

inserting element .... 32

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice1

enter item to insert:43

inserting element .... 43

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice1

enter item to insert:87

inserting element .... 87

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice1

overflow

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice3

the queue is|32|43|87|

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice2

removing element .... 32

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice2

removing element .... 43

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice2

removing element .... 87

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice2

queue underflow

Enter 1 to insert 2 to dequeue 3 to display 4 to stop

input choice4

---End---

1. **Implementation of Priority Queue**

class PriorityQueue(object):

def \_\_init\_\_(self):

self.queue = []

def isEmpty(self):

return len(self.queue) == 0

def Insert(self,data):

self.queue.append(data)

def Delete(self):

max\_val = 0

for i in range(len(self.queue)):

if self.queue[i] > self.queue[max\_val]:

max\_val = i

item = self.queue[max\_val]

del(self.queue[max\_val])

return item

myQueue = PriorityQueue()

myQueue.Insert(12)

myQueue.Insert(1)

myQueue.Insert(14)

myQueue.Insert(7)

print("Queue created")

print(myQueue.queue)

print("Demonstrating deletion from priority Queue")

while not myQueue.isEmpty():

print(myQueue.Delete())

**Output:**

Queue created

[12, 1, 14, 7]

Demonstrating deletion from priority Queue

12

1

14

7

1. **Binary Search Tree**

class Node:

def \_\_init\_\_ (self,key):

self.left = None

self.right = None

self.val = key

def Insert(root,key):

if root is None:

return Node(key)

if root.val == key:

return root

if key > root.val:

root.right = Insert(root.right,key)

else:

root.left = Insert(root.left,key)

return root

def Search(root,key):

if root is None:

print("Key is not present in the tree\n")

return root

if root.val == key:

print(key,"is present in the tree\n")

return root

if root.val < key:

return Search(root.right,key)

else:

return Search(root.left,key)

def Inorder(root):

if root:

Inorder(root.left)

print(root.val,end = ",")

Inorder(root.right)

r = None

n = int(input("How many nodes you want in the tree?"))

print("")

val = int(input("input data for the root node: "))

print("")

r = Node(val)

while n > 1:

val = int(input("input value to insert: "))

r = Insert(r,val)

n = n-1

print("\n Tree contents")

Inorder(r)

ch = int(input("Enter 1 to search"))

while ch == 1:

key = int(input("Input value to search"))

Search(r,key)

ch = int(input("Enter 1 to search"))

print("-----END-----")

**Output:**

How many nodes you want in the tree?3

input data for tbe root node: 32

input value to insert: 54

input value to insert: 21

Tree contents

21,32,54,Enter 1 to search1

Input value to search32

32 is present in the tree

Enter 1 to search1

Input value to search45

Key is not present in the tree

Enter 1 to search4

-----END-----

1. **BFS**

class Node:

def \_\_init\_\_(self,key):

self.data=key

self.left=None

self.right=None

def printLevelorder(root):

h=height(root)

for i in range(1,h+1):

printCurrentLevel(root,i)

def printCurrentLevel(root,level):

if root is None:

return

if level==1:

print(root.data,end=",")

elif level>1:

printCurrentLevel(root.left,level-1)

printCurrentLevel(root.right,level-1)

def height(Node):

if Node is None:

return 0

else:

lheight=height(Node.left)

rheight=height(Node.right)

if lheight>rheight:

return lheight+1

else:

return rheight+1

Root=Node(1)

Root.left=Node(2)

Root.right=Node(3)

Root.left.left=Node(4)

Root.left.right=Node(5)

Root.right.left=Node(6)

Root.right.right=Node(7)

print("Level order traversal of Binary Tree is : ")

printLevelorder(Root)

1. **DFS**

class Node:

def \_\_init\_\_(self, key):

self.left = None

self.right = None

self.val = key

# Function to insert given key as a node

def insert(root, key):

if root is None:

return Node(key)

if root.val == key:

return root

if key > root.val:

root.right = insert(root.right, key)

else:

root.left = insert(root.left, key)

return root

# Recursive function to traverse BST in inorder

def inorder(root):

if root:

inorder(root.left)

print(root.val, end = ',')

inorder(root.right)

# Recursive function to traverse BST in postorder

def postorder(root):

if root:

inorder(root.left)

inorder(root.right)

print(root.val, end = ',')

# Recursive function to traverse BST in preorder

def preorder(root):

if root:

print(root.val, end = ',')

inorder(root.left)

inorder(root.right)

#Driver program

r = None

n = int(input("how many nodes you want in the tree? :"))

val = int(input("input data for root node :"))

r = Node(val)

while n > 1:

val = int(input("input value to insert :"))

r = insert(r, val)

n=n-1

# Print inoder traversal of the BST

print("Tree created :")

print("\n Postorder traversal")

postorder(r)

print("\n Preorder traversal")

preorder(r)

print("\n Inorder traversal")

inorder(r)

**Output:**

how many nodes you want in the tree? :5

input data for root node :6

input value to insert :3

input value to insert :8

input value to insert :5

input value to insert :2

Tree created :

Postorder traversal

2,3,5,8,6,

Preorder traversal

6,2,3,5,8,

Inorder traversal

2,3,5,6,8,

1. **Hash Table**

hash\_table = [[] for i in range(10)]

def hashing\_func(key):

return key % len(hash\_table)

def insert(hash\_table, key, value):

hash\_key = hashing\_func(key)

bucket = hash\_table[hash\_key]

bucket.append((key, value))

insert(hash\_table, 10, 'Nepal')

insert(hash\_table, 25, 'USA')

insert(hash\_table, 20, 'India')

insert(hash\_table, 43, 'Bhutan')

print (hash\_table)

**Output:**

[[(10, 'Nepal'), (20, 'India')], [], [], [(43, 'Bhutan')], [], [(25, 'USA')], [], [], [], []]