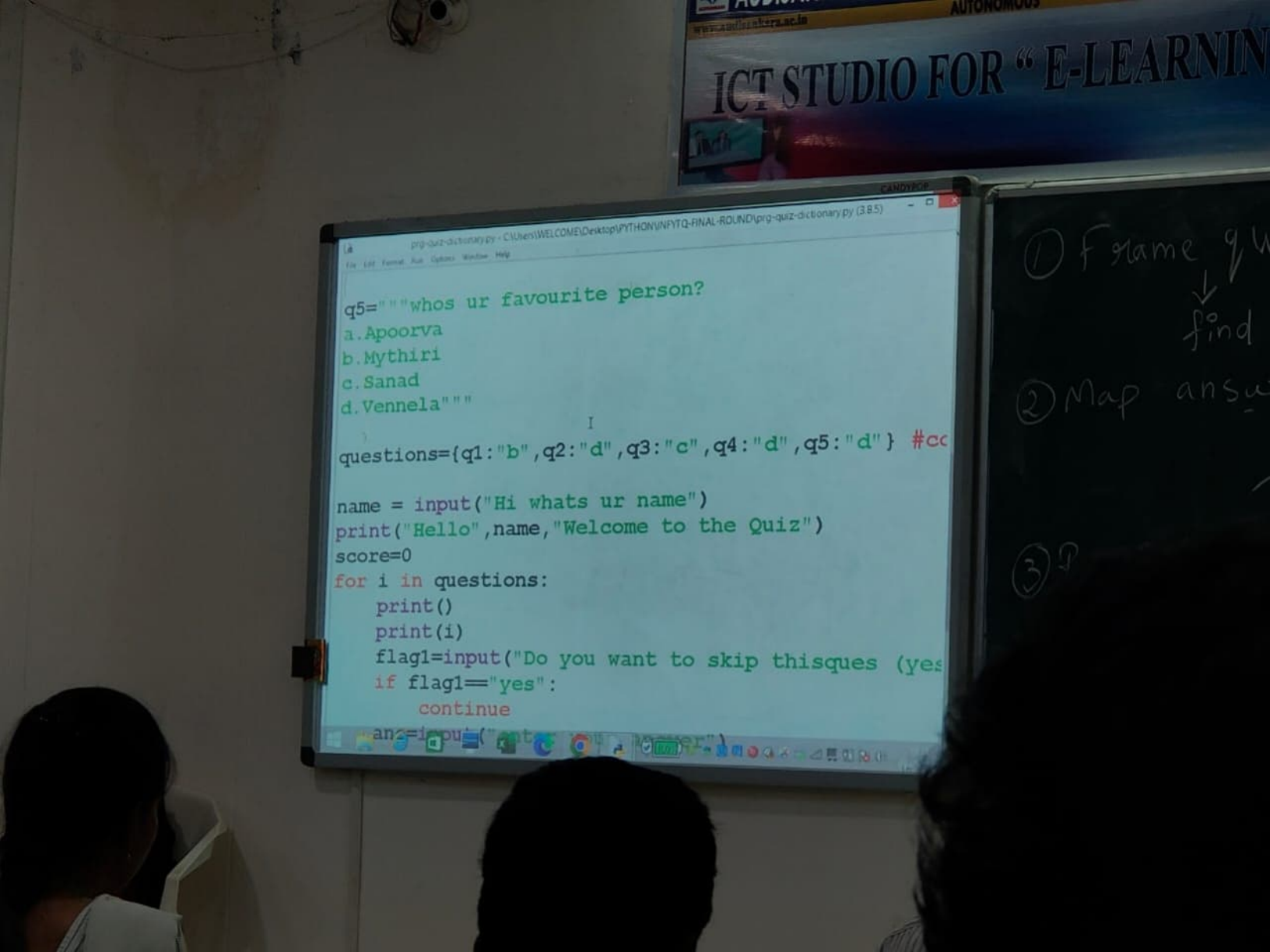
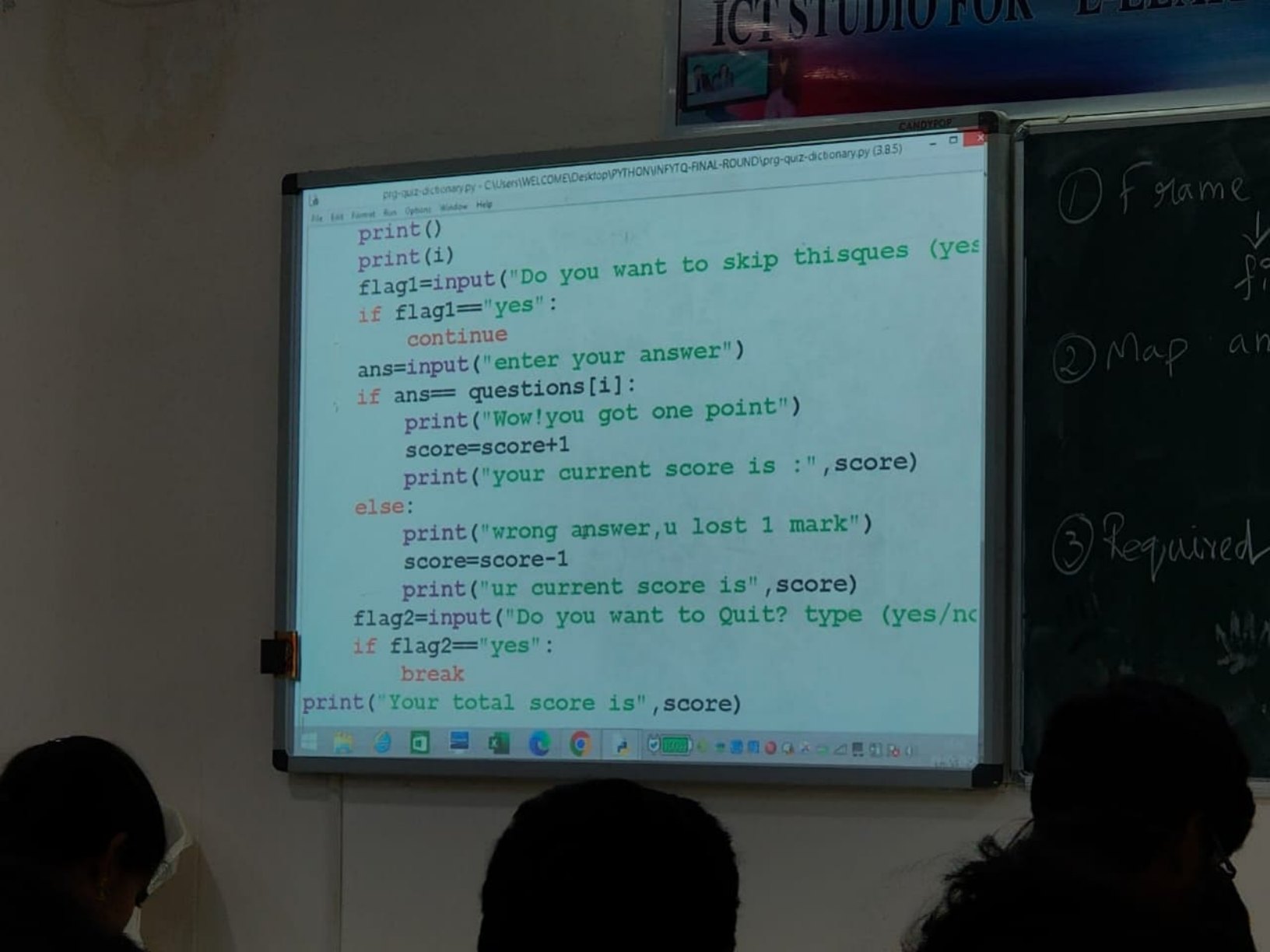
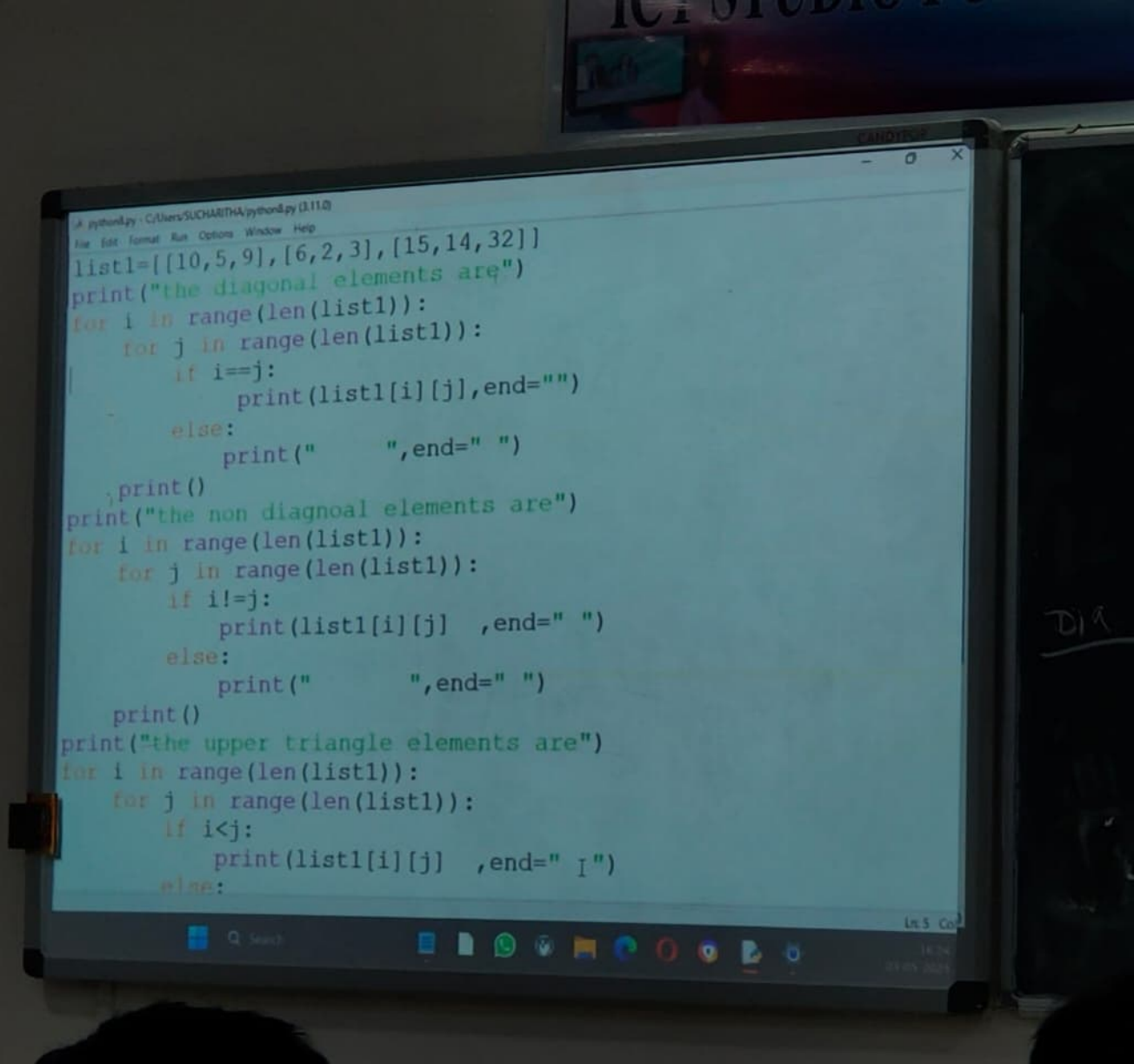
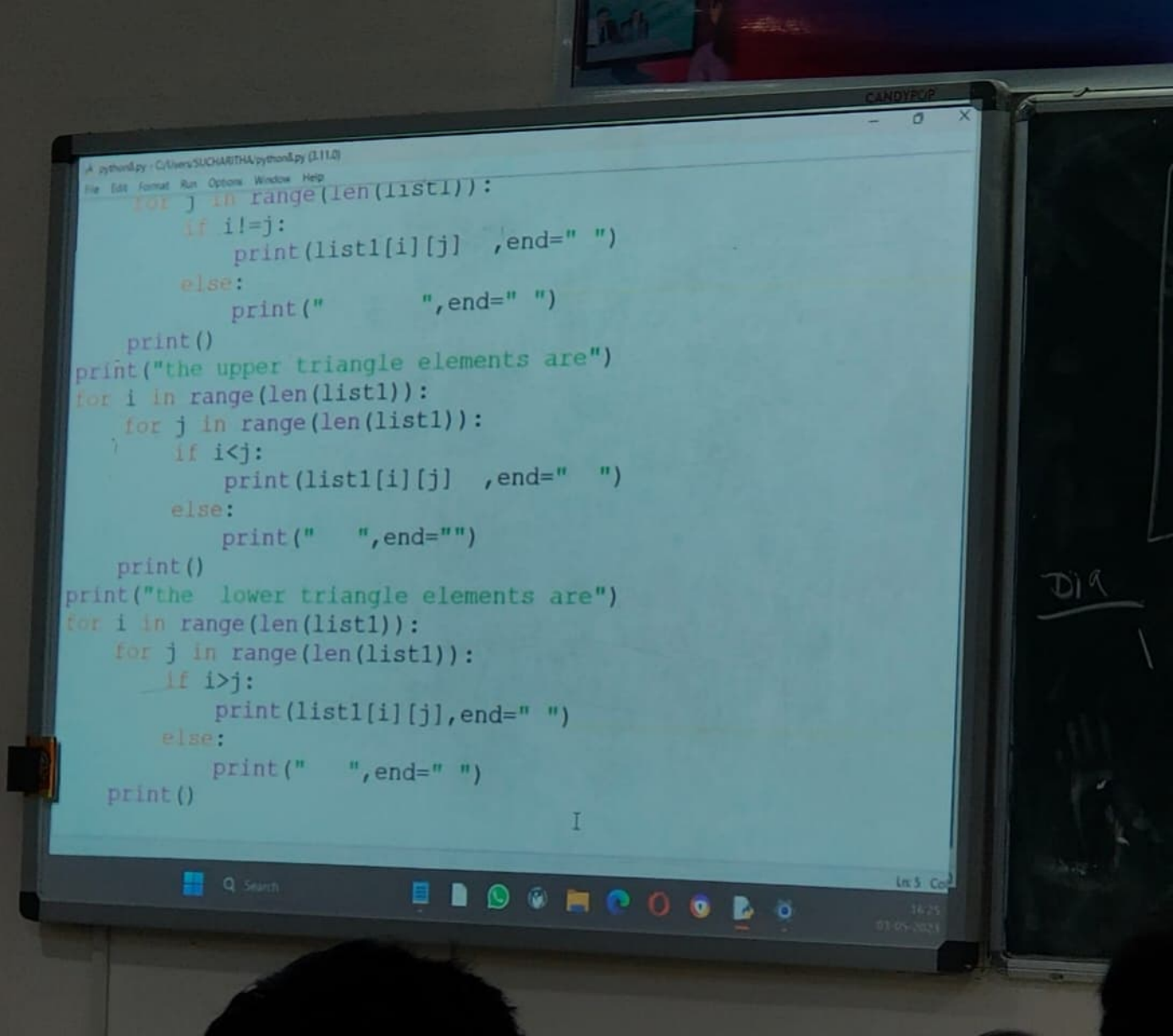

ICT STUDIO FOR " E-LEARNIN
prg-quiz-dictionary.py - C:\Users\WELCOME\Desktop\PYTHON\NFYTQ-FINAL-ROUND\prg-quiz-dictionary.py (3.8.5)
q5="""whos ur favourite person?
a.Apoorva
b.Mythiri
c.Sanad
d.Vennela"""
questions={q1:"b",q2:"d",q3:"c",q4:"d",q5:"d"} #cc
name = input("Hi whats ur name")
print("Hello",name,"Welcome to the Quiz")
score=0
for i in questions:
print()
print(i)
flag1=input("Do you want to skip thisques (yes
if flag1=="yes":
continue
find
Map


prg-quiz-dictionary.py - C:\Users\WELCOME\Desktop\PYTHON\NFYTQ-FINAL-ROUND\prg-quiz-dictionary.py (3.8.5)
File Edit Format Run Options Window Help
print()
print(i)
flag1=input("Do you want to skip thisques (yes
if flag1=="yes":
continue
ans=input("enter your answer")
if ans== questions[i]:
print("Wow!you got one point")
score=score+1
print("your current score is :",score)
else:
print("wrong answer,u lost 1 mark")
score=score-1
print("ur current score is",score)
flag2=input("Do you want to Quit? type (yes/nc
if flag2=="yes":
break
print("Your total score is",score)
1 Frame
2 Map
3 Required



```
list1=[[10,5,9],[6,2,3],[15,14,32]]
print("the diagonal elements are")
for i in range(len(list1)):
    for j in range(len(list1)):
        if i==j:
            print(list1[i][j],end="")
        else:
            print("      ",end=" ")
    print()
print("the non diagnoal elements are")
for i in range(len(list1)):
    for j in range(len(list1)):
        if i!=j:
            print(list1[i][j]  ,end=" ")
        else:
            print("        ",end=" ")
    print()
print("the upper triangle elements are")
for i in range(len(list1)):
    for j in range(len(list1)):
        if i<j:
            print(list1[i][j]  ,end=" I")
        else:
```



```
    for j in range(len(list1)):
        if i!=j:
            print(list1[i][j]  ,end=" ")
        else:
            print("       ",end=" ")
    print()
print("the upper triangle elements are")
for i in range(len(list1)):
    for j in range(len(list1)):
        if i<j:
            print(list1[i][j]  ,end="  ")
        else:
            print("   ",end="")
    print()
print("the  lower triangle elements are")
for i in range(len(list1)):
    for j in range(len(list1)):
        if i>j:
            print(list1[i][j],end=" ")
        else:
            print("   ",end=" ")
    print()
```

```
#we use 2 pointers  here
#second node  is temp/ first node prev
#temp will point  last
#prev node u r making it as quit
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class LL:
    def __init__(self):
        self.head=None
    def display(self):
        if self.head==None:
            print("empty")
        else:
            temp=self.head
            while (temp):
                print(temp.data)
                temp=temp.next
```

```
#we use 2 pointers  here
#second node  is temp/ first node prev
#temp will point  last
#prev node u r making it as quit
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
class LL:
    def __init__(self):
        self.head=None
    def display(self):
        if self.head==None:
            print("empty")
        else:
            temp=self.head
            while (temp):
                print(temp.data)
                temp=temp.next
```

```
                temp=temp.next
    def delete(self):
        temp=self.head.next
        prev=self.head
        while temp.next is not None:
            temp=temp.next
            prev=prev.next
        prev.next=None #last but before node's nex



obj=LL()
n1=Node(1)
obj.head=n1
n2=Node(2)
obj.head.next=n2
n3=Node(3)
n2.next=n3
obj.display()
```

```
        temp=self.head.next
        prev=self.head
        while temp.next is not None:
            temp=temp.next
            prev=prev.next
        prev.next=None #last but before node's nex


obj=LL()
n1=Node(1)
obj.head=n1
n2=Node(2)
obj.head.next=n2
n3=Node(3)
n2.next=n3
obj.display()
obj.delete()
obj.display()
```

```
#2 pointers
#temp will be 30
#prev  will  be 20

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

class singlelinkedlist:
    def __init__(self):
        self.head=None

    def delete_position(self,pos):
        temp=self.head.next
        prev=self.head
        #2 iterations
        for i in range(1,pos-1):
            temp=temp.next
```

```
        for i in range(1,pos-1):
            temp=temp.next
            prev=prev.next
        prev.next=temp.next#20  1 point 40
        temp.next=None #30s next will be null

    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


obj=singlelinkedlist()
n=Node(10)
obj.head=n
n1=Node(20)
```

```
n=Node(10)
obj.head=n
n1=Node(20)
n.next=n1
n2=Node(30)
n1.next=n2
n3=Node(40)
n2.next=n3
n4=Node(50)
n3.next=n4
print("before")
obj.display()


obj.delete_position(3)

print("after deleting pos-3")
obj.display()
```

```
#create node
class Node:
    def __init__(self,data):
        self.data=data
        self.previous=None
        self.next=None

class dll:
    def __init__(self):
        self.head=None

    def display(self):
        if self.head is None:
            print("Empty")
        else:
            temp=self.head
            while temp:
                print(temp.data,"<-->",end="  ")
                temp=temp.next
```

```
class Node:

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

class CreateList:
  def __init__(self):
    self.head = Node(None)
    self.tail = Node(None)
    self.head.next = self.tail #making single node
    self.tail.next = self.head

  #Adding node at end of LL
  def add(self,data):
    newNode = Node(data)
    #Is Empty?
    if self.head.data is None:
      self.head = newNod
```

File Edit Format Run Options Window Help

```
class CreateList:
    def __init__(self):
        self.head = Node(None)
        self.tail = Node(None)
        self.head.next = self.tail #making single node
        self.tail.next = self.head

    #Adding node at end of LL
    def add(self,data):
        newNode = Node(data)
        #Is Empty?
        if self.head.data is None:
            self.head = newNode
            self.tail = newNode
            newNode.next = self.head
        else:
            self.tail.next = newNode
            self.tail = newNode
            #It is CLL so tail will point to head.
```

```
    if self.head is None:
      print("List is empty")
      return
    else:
        print("Nodes of the circular linked list:
        print(current.data,"-->")
        while(current.next != self.head):
            current = current.next
        print(current.data,"-->") #it shows E conr

class CircularLinkedList:
  cl = CreateList()
  cl.add("s")
  cl.add("M")
  cl.add("I")
  cl.add("L")
  cl.add("E")
  cl.display()
```

File Edit Format Run Options Window Help

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

class Queue:
    def __init__(self):
        self.head = None
        self.last = None

    def enqueue(self, data):
        if self.last is None:
            self.head = Node(data)
            self.last = self.head
        else:
            self.last.next = Node(data)
            self.last = self.last.next

    def dequeue(self):
```

File Edit Format Run Options Window Help

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

class Queue:
    def __init__(self):
        self.head = None
        self.last = None

    def enqueue(self, data):
        if self.last is None:
            self.head = Node(data)
            self.last = self.head
        else:
            self.last.next = Node(data)
            self.last = self.last.next

    def dequeue(self):
```

```
class PriorityQueue(object):
#like int and float object is one
#primitive data type
    def __init__(self):
        self.queue = []

    def __str__(self):      #represent class objects
        return ' '.join([str(i) for i in I
                         self.queue])

    # for checking if the queue is empty
    def isEmpty(self):
        return len(self.queue) == 0

    # for inserting an element in the queue
    def insert(self, data):
        self.queue.append(data)

    # for popping an element based on Priority
```

File Edit Format Run Options Window Help

```
    # for popping an element based on Priority
    def delete(self):
        try:
            max = 0
            for i in range(len(self.queue)):
                print("i",i)
                if self.queue[i] > self.queue[max]
                    max = i
                    print("max",max)
            item = self.queue[max]
            del self.queue[max]
            return item
        except IndexError:
            print()
            exit()


if __name__ == '__main__':
    myQueue = PriorityQueue()
```

File Edit Format Run Options Window Help

```
            item = self.queue[max]
            del self.queue[max]
            return item
        except IndexError:
            print()
            exit()

if __name__ == '__main__':
    myQueue = PriorityQueue()
    myQueue.insert(12)
    myQueue.insert(1)
    myQueue.insert(14)#high priority
    myQueue.insert(7)
    print(myQueue)
    while not myQueue.isEmpty():
        print(myQueue.delete())
```

```
#insert:rear=rear+1--we add new person at end of c
#delete:front =front+1 --FIFO

class CircularQueue() :
    def __init__(self, size): #
        #initializing the class
        self.size = size
        #can use self.queue=[None]*size
        self.queue = [None for i in range(size)]
        self.front = self.rear = -1

    def enqueue(self, data):
        # condition if queue is full
        if ((self.rear + 1) % self.size ==
            self.front):# size 6 index from 0
            print(" Queue is Full\n")
        # condition for empty queue
        elif (self.front == -1):
            self.front = 0
```

File Edit Format Run Options Window Help

```
        # condition for empty queue
        elif (self.front == -1):
            self.front = 0
            self.rear = 0
            self.queue[self.rear] = data
            #always add element at tail place
        else:
            # next position of rear
            self.rear = (self.rear + 1)
            %self.size
            self.queue[self.rear] = data
    def dequeue(self):
        if (self.front == -1):
            # condition for empty queue
            print ("Queue is Empty\n")
        # condition for only one element
        elif (self.front == self.rear):
            temp=self.queue[self.front]
            self.front = -1
```

```
            self.front = -1
            self.rear = -1
            return temp
        else:
            temp = self.queue[self.front]
            self.front = (self.front + 1)
            %self.size
            return temp

    def display(self):
        # condition for empty queue
        if(self.front == -1):
            print ("Queue is Empty")

        elif (self.rear >= self.front):
            print("Elements in the circular queue
                                                end
            for i in range(self.front,self.rear +
                print(self.queue[i], end = " ")
```

```
File  Edit  Format  Run  Options  Window  Help
                                                    end
            for i in range(self.front,
                           self.rear + 1):
                print(self.queue[i], end = " ")
            print ()

        else:
            print ("Elements in Circular Queue are
                                                end = "
            for i in range(self.front, self.size):
                print(self.queue[i], end = " ")
            for i in range(0, self.rear + 1):
                print(self.queue[i], end = " ")
            print ()

        if ((self.rear + 1) % self.size ==self.fro
            print("Queue is Full")
ob = CircularQueue(5)
ob.enqueue(14)
```

File Edit Format Run Options Window Help

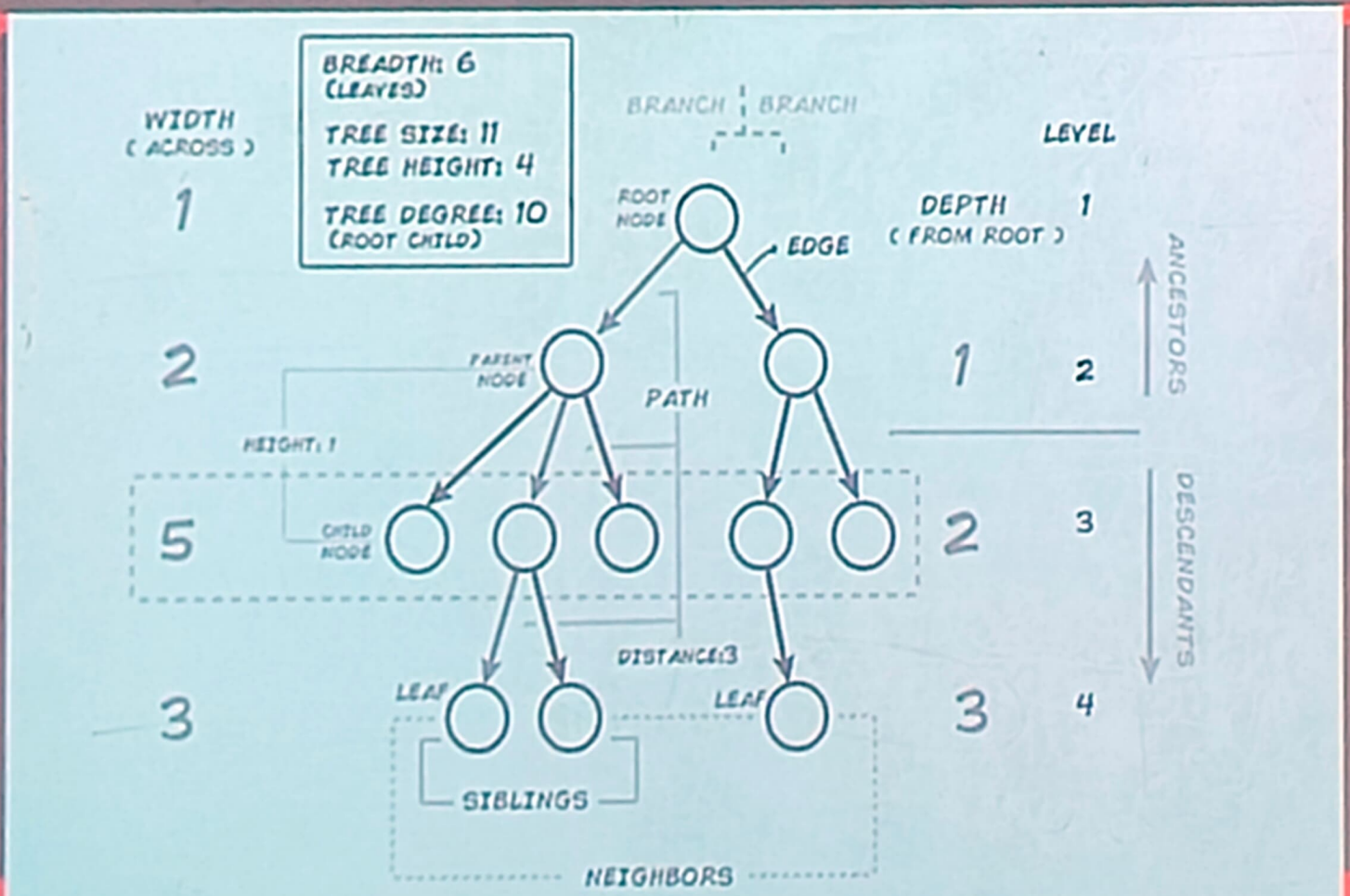
```
        if ((self.rear + 1) %
            self.size ==self.front):
            print("Queue is Full")
ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
#it wnt be inserted bcs queue full
ob.enqueue(100)
```

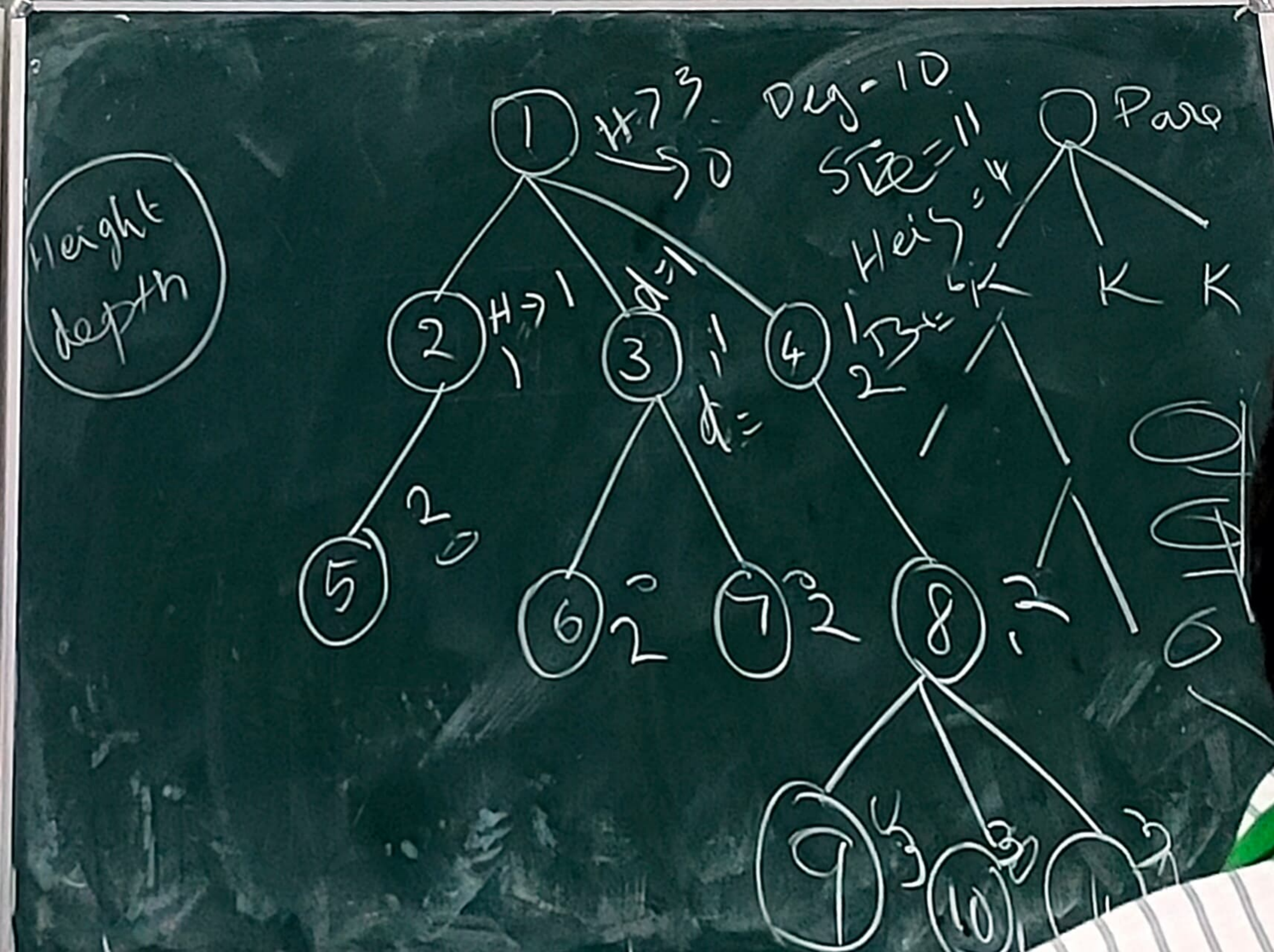
TREE

Applications of trees:-

Following are some of the main applications of trees.

- Used to manipulate hierarchical data or information.
- Through trees, Searching is very easy.
- Router Algorithms.
- Used to manipulate sorted lists of data.
- With this, you can form multi stage decision making.


BREADTH: 6
(LEAVES)
TREE SIZE: 11
TREE HEIGHT: 4
TREE DEGREE: 10
(ROOT CHILD)
WIDTH
(ACROSS)
BRANCH
BRANCH
LEVEL
1
ROOT
NODE
EDGE
DEPTH
(FROM ROOT)
1
ANCESTORS
2
PARENT
NODE
PATH
1
2
HEIGHT: 1
5
CHILD
NODE
2
3
DESCENDANTS
DISTANCE:3
LEAF
LEAF
3
3
4
SIBLINGS
NEIGHBORS


Height
depth
H→3
D
Deg-10
SIZE=11
Pare
K
K
K
1
2
3
4
5
6
7
8
9
10
H→1

```
def printPreorder(root):
    if root:
        # First print the data of node
        print(root.val,end=" "),
        # Then recur on left child
        printPreorder(root.left)
        # Finally recur on right child
        printPreorder(root.right)

root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
print ("PRE-ORDER:")
printPreorder(root)
print()
print ("\nIN-ORDER:")
printInorder(root)
print()
```

```
#BST-INSERT
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)
    else:
        if root.val == key:
            return root
        elif root.val < key:
            root.right = insert(root.right, key)
        else:
            root.left = insert(root.left, key)
    return root
#Inorder-traversal
def inorder(root):
```

5-BST-search-logicpartalone.py - C:\Users\WELCOME\Desktop\PYTHON\TREE\5-BST-search-logicpartalone.py (3.8.5)

File Edit Format Run Options Window Help

```
def search(root,key):

    # Base Cases: root is null or key is present a
    if root is None or root.val == key:
        return root

    # Key is greater than root's key
    if root.val < key:
        return search(root.right,key)

    # Key is smaller than root's key
    return search(root.left,key)
```

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
def inorder(root):
    if root is not None:
        inorder(root.left)
        print (root.key)
        inorder(root.right)
def insert(node, key):
    if node is None:
        return Node(key)
    if key < node.key:
        node.left = insert(node.left, key)
    else:
        node.right = insert(node.right, key)
    return node
# Given a non-empty binary
```

```
    return node
# Given a non-empty binary
# search tree, return the node
# with minum key value
# found in that tree. Note that the
# entire tree does not need to be searched
def minValueNode(node):    #right sub tree
    current = node
    # loop down to find the leftmost leaf
    while(current.left is not None):
        current = current.left
    return current
# Given a binary search tree and a key, this func
# delete the key and returns the new root
def deleteNode(root, key):
    # Base Case
    if root is None:
        return root
    # key<root, it lies in left subtree
```

File Edit Format Run Options Window Help

```
        return root
    # key<root,it lies in  left subtree
    if key < root.key:
        root.left = deleteNode(root.left, key)
    elif(key > root.key):
        root.right = deleteNode(root.right, key)
    # If key is same as root's key, then this is t
    # to be deleted
    else:
        # Node with only one child or no child
        if root.left is None:
            temp = root.right
            root = None
            return temp
        elif root.right is None:
            temp = root.left
            root = None
            return temp
        # Node with two children:
```

```
            return temp
        # Node with two children:
        # Get the inorder successor
        # (smallest in the right subtree)
        temp = minValueNode(root.right)
        # Copy the inorder successor's
        # content to this node
        root.key = temp.key
        # Delete the inorder successor
        root.right = deleteNode(root.right,
                                temp.key)
    return root
""" Let us create following BST
              50
           /     \
          30      70
         /  \    /  \
       20   40  60   80 """
```

```
root = None
root = insert(root, 50)
root = insert(root, 30)
root = insert(root, 20)
root = insert(root, 40)
root = insert(root, 70)
root = insert(root, 60)
root = insert(root, 80)
print ("Inorder traversal of the given tree")
inorder(root)
print ("\nDelete 20")
root = deleteNode(root, 20)
print ("Inorder traversal of the modified tree")
inorder(root)
print ("\nDelete 30")
root = deleteNode(root, 30)
print ("Inorder traversal of the modified tree")
inorder(root)
print ("\nDelete 50")
```

```
root = insert(root, 40)
root = insert(root, 70)
root = insert(root, 60)
root = insert(root, 80)
print ("Inorder traversal of the given tree")
inorder(root)
print ("\nDelete 20")
root = deleteNode(root, 20)
print ("Inorder traversal of the modified tree")
inorder(root)
print ("\nDelete 30")
root = deleteNode(root, 30)
print ("Inorder traversal of the modified tree")
inorder(root)
print ("\nDelete 50")
root = deleteNode(root, 50)
print ("Inorder traversal of the modified tree")
inorder(root)
```

File Edit Format Run Options Window Help

```
graph = {
   '5' : ['3','7'],
   '3' : ['2', '4'],
   '7' : ['8'],
   '2' : [],
   '4' : ['8'],
   '8' : []
}
#BFS - we use Queue
visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:                # Creating loop to visit e
    m = queue.pop(0)
```

graph-2-bfs.py - C:\Users\WELCOME\Desktop\PYTHON\graph-2-bfs.py (3.8.5)

File Edit Format Run Options Window Help

```
visited = [] # List for visited nodes.
queue = []     #Initialize a queue

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:                # Creating loop to visit e
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

bfs(visited, graph, '5')     # function call
```

graph-3-dfs.py - C:\Users\WELCOME\Desktop\PYTHON\graph-3-dfs.py (3.8.5)

File Edit Format Run Options Window Help

```
# Using dictionary to act as an adjacency list
#stack
#dfs-produces a spanning tree-its without loop
#push vertex to stack, push one of its adjacent
#keep doing, once u see adjacets of vertex all vis
#back track - means pop top element
#go to next top of element repeat
#till stack becomes empty

graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = set() # Set to keep track of visited noc
```

graph-3-dfs.py - C:\Users\WELCOME\Desktop\PYTHON\graph-3-dfs.py (3.8.5)

File Edit Format Run Options Window Help

```
    '5' : ['3','7'],
    '3' : ['2', '4'],
    '7' : ['8'],
    '2' : [],
    '4' : ['8'],
    '8' : []
}

visited = set() # Set to keep track of visited noc

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

dfs(visited, graph, '5')
```

```
'''globals(global scope):
its a function, returns global dictionary
of symbol table
symbol table : its a data structure
complete info about the program'''
pos=-1
def search(list,n):
    l=0
    u=len(list)-1
    while l<=u:
        mid=(l+u)//2
        if(list[mid])==n:
            globals()['pos']=mid
            return True
        else:
            if list[mid]<n:
                l=mid+1
            else:
```

```
'''globals(global scope):
its a function, returns global dictionary
of symbol table
symbol table : its a data structure
complete info about the program'''
pos=-1
def search(list,n):
    l=0
    u=len(list)-1
    while l<=u:
        mid=(l+u)//2
        if(list[mid])==n:
            globals()['pos']=mid
            return True
        else:
            if list[mid]<n:
                l=mid+1
            else:
                u=mid-1
```

```
            globals()['pos']=mid
            return True
        else:
            if list[mid]<n:
                l=mid+1
            else:
                u=mid-1
    return False

list=[5,10,15,100]
n=167
if search(list,n):
    print("Found",pos+1)
else:
    print("Not found")
```

File Edit Format Run Options Window Help

```
70   40    50    10
PASS-1:
     70 and 40 will be compared
      40<70 -- so swap
40    70    50   10
     compare 70 and 50
        50<70   -- swap
40    50    70    10
       compare 70 and 10
          10 < 70 ---   swap
40   50   10    70   .....    LAST ELEMENT IS FIXED

PASS - 2
       40   AND 50
          50< 40 .. NO...NO need to swap
40   50   10   70
        50 and 10
        10<50 swap
```

bubble-sort-concept.py - C:\Users\WELCOME\Desktop\PYTHON\SORTING\bubble-sort-concept.py (3.8.5)

File Edit Format Run Options Window Help

```
PASS - 2
      40  AND 50
          50< 40 .. NO...NO need to swap
40   50   10   70
       50 and 10
       10<50  .....  swap
40   10   50   70  .....   2nd largest number got fi

PASS-3
      40   AND  10
      SWAP  BCS 10<40
10   40   50    70
       40  AND 50
           50<40 .. NO ....NO NEED TO SWAP
10   40    50    70  ----  DONE
```