1. Python program to Use and demonstrate basic data structures.

```
print("List")
l1 = [1, 2,"ABC", 3, "xyz", 2.3]
print(l1)
print("Dictionary")
d1={"a":134,"b":266,"c":343}
print(d1)
print("Tuples")
t1=(10,20,30,40,50,40)
print (t1)
print("Sets")
s1={10,30,20,40,10,30,40,20,50,50}
print(s1)
```

2. Implement an ADT with all its operations.

```
class date:
    def __init__(self,a,b,c):
        self.d=a
        self.m=b
        self.y=c
    def day(self):
        print("Day = ", self.d)
    def month(self):
        print("Month = ", self.m)
    def year(self):
        print("year = ", self.y)
    def monthName(self):
        months = ["Unknown","January","Febuary","March","April","May","June","July",
         "August","September","October","November","December"]
        print("Month Name:",months[self.m])
    def isLeapYear(self):
        if (self.y % 400 == 0) and (self.y % 100 == 0):
            print("It is a Leap year")
        elif (self.y % 4 == 0) and (self.y % 100 != 0):
            print("It is a Leap year")
        else:
            print("It is not a Leap year")
d1 = date(3,8,2000)
d1.day()
d1.month()
d1.year()
```

```
d1.monthName()
d1.isLeapYear()
```

3. Implement an ADT and Compute space and time complexities.

```
import time
class stack:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items) - 1]
    def size(self):
        return len(self.items)
    def display(self):
        return (self.items)
s=stack()
start = time.time()
print(s.isEmpty())
print("push operations")
s.push(11)
s.push(12)
s.push(13)
print("size:",s.size())
print(s.display())
print("peek",s.peek())
print("pop operations")
print(s.pop())
print(s.pop())
print(s.display())
print("size:",s.size())
end = time.time()
print("Runtime of the program is", end - start)
```

4. Implement Linear Search and compute space and time complexities, plot graph using asymptomatic notations

```
import time
def linearsearch(a, key):
    n = len(a)
    for i in range(n):
        if a[i] == key:
            return i;
    return -1
a = [13,24,35,46,57,68,79]
start = time.time()
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
i = linearsearch(a,k)
if i == -1:
    print("Search UnSuccessful")
else:
    print("Search Successful key found at location:",i+1)
end = time.time()
print("Runtime of the program is", end-start)
```

5. Implement Bubble Sort and compute space and time complexities, plot graph using asymptomatic notations

```
def bubblesort(a):
    n = len(a)
    for i in range(n-2):
        for j in range(n-2-i):
            if a[j]>a[j+1]:
                temp = a[j]
                a[j] = a[j+1]
                a[j+1] = temp
x = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",x)
bubblesort(x)
print("After sorting:",x)
```

6. Implement Selection Sort and compute space and time complexities, plot graph using asymptomatic notations

```
def selectionsort(a):
    n = len(a)
```

```
    for i in range(n-2):
        min = i
        for j in range(i+1,n-1):
            if a[j]<a[min]:
                min=j
        temp = a[i]
        a[i] = a[min]
        a[min] = temp
x = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",x)
selectionsort(x)
print("After sorting:",x)
```

7. Implement Insertion Sort and compute space and time complexities, plot graph using asymptomatic notations

```
def insertionsort(a):
    n = len(a)
    for i in range(1,n-1):
        v=a[i]
        j = i-1
        while j>=0 and a[j]>v:
            a[j+1] = a[j]
            j=j-1
            a[j+1] = v
x = [34,46,43,27,57,41,45,21,70]
print("Before sorting:",x)
insertionsort(x)
print("After sorting:",x)
```

8. Implement Binary Search and compute space and time complexities, plot graph using asymptomatic notations

```
import time
def binarysearch(a, key):
    low = 0
    high = len(a) - 1
    while low <= high:
        mid = (high + low) // 2
        if a[mid] == key:
            return mid
        elif key < a[mid]:
            high = mid - 1
```

```
        else :
            low = mid + 1
    return -1
start = time.time()
a = [13,24,35,46,57,68,79]
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
r = binarysearch(a,k)
if r == -1:
    print("Search UnSuccessful")
else:
    print("Search Successful key found at location:",r+1)
end = time.time()
print("Runtime of the program is:", end-start)
```

9. Implement Binary Search using Recursion and compute space and time complexities, plot graph using asymptomatic notations

```
def binarysearch(a, low, high, key):
    if low <= high:
        mid = (high + low) // 2
        if a[mid] == key:
            print("Search Successful key found at location:",mid+1)
            return
        elif key < a[mid]:
            binarysearch(a, low, mid-1, k)
        else :
            binarysearch(a, mid + 1, high, k)
    else:
        print("Search UnSuccessful")
a = [13,24,35,46,57,68,79]
print("the array elements are:",a)
k = int(input("enter the key element to search:"))
binarysearch(a, 0, len(a)-1, k)
```

10. Implement Fibonacci sequence with dynamic programming.

```
def fib(n):
    if n<=1:
        return n
    f = [0, 1]
    for i in range(2, n+1):
        f.append(f[i-1] + f[i-2])
    print("The Fibonacci sequence is:",f)
```

```
    return f[n]
n=int(input("Enter the term:"))
print("The Fibonacci value is:",fib(n))
```

11. Implement singly linked list (Traversing the Nodes, searching for a Node, Prepending Nodes, and Removing Nodes)

```
class Node:
    def __init__(self, data = None):
        self.data = data
        self.next = None
class SinglyLinkedList:
    def __init__(self):
        self.first = None
    def insertFirst(self, data):
        temp = Node(data)
        temp.next=self.first
        self.first=temp
    def removeFirst(self):
        if(self.first== None):
            print("list is empty")
        else:
            cur=self.first
            self.first=self.first.next
            print("the deleted item is",cur.data)
    def display(self):
        if(self.first== None):
            print("list is empty")
            return
        cur = self.first
        while(cur):
          print(cur.data, end = " ")
          cur = cur.next
    def search(self,item):
        if(self.first== None):
            print("list is empty")
            return
        cur = self.first
        while cur != None:
            if cur.data == item:
                print("Item is Present in the Linked list")
                return
            else:
                cur = cur.next
        print("Item is not present in the Linked list")
#Singly Linked List
sll = SinglyLinkedList()
while(True):
```

```
    ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
    if(ch == 1):
        item = input("Enter the element to insert:")
        sll.insertFirst(item)
        sll.display()
    elif(ch == 2):
        sll.removeFirst()
        sll.display()
    elif(ch == 3):
        item = input("Enter the element to search:")
        sll.search(item)
    elif(ch == 4):
        sll.display()
    else:
        break
```

12. Implement singly linked list using Iterators.

```
class Node:
    def __init__(self, data = None):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.first = None
    def insert(self, data):
        temp = Node(data)
        if(self.first):
            cur = self.first
            while(cur.next):
                cur = cur.next
            cur.next = temp
        else:
            self.first = temp
    def __iter__(self):
        cur = self.first
        while cur:
            yield cur.data
            cur = cur.next
# Linked List Iterators
ll = LinkedList()
ll.insert(9)
ll.insert(98)
ll.insert("welcome")
ll.insert("govt polytechnic koppal")
ll.insert(456.35)
ll.insert(545)
ll.insert(5)
for x in ll:
    print(x)
```

13. Implementation of Doubly linked list (DLL)(Traversing the Nodes, searching for a Node, Appending Nodes, Deleting Nodes):

```
class Node:
    def __init__(self, data = None):
        self.data = data
        self.next = None
        self.prev = None
class DoublyLinkedList:
    def __init__(self):
        self.first = None
    def insertAtEnd(self, data):
        temp = Node(data)
        if(self.first == None):
             self.first=temp
        else:
            cur = self.first
            while(cur.next != None):
                cur = cur.next
            cur.next = temp
            temp.prev = cur
    def deleteFirst(self):
        if(self.first== None):
            print("list is empty")
        elif(self.first.next == None):
            print("the deleted item is",self.first.data)
            self.first = None
        else:
            cur=self.first
            self.first=self.first.next
            self.first.prev = None
            print("the deleted item is",cur.data)
    def display(self):
        if(self.first== None):
            print("list is empty")
            return
        cur = self.first
        while(cur):
          print(cur.data, end = " ")
          cur = cur.next
    def search(self,item):
        if(self.first== None):
            print("list is empty")
```

```
            return
        cur = self.first
        while cur != None:
            if cur.data == item:
                print("Item is present in the Linked list")
                return
            else:
                cur = cur.next
        print("Item is not present in the Linked list")
#Doubly Linked List
dll = DoublyLinkedList()
while(True):
    ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
    if(ch == 1):
        item = input("Enter the element to insert:")
        dll.insertAtEnd(item)
        dll.display()
    elif(ch == 2):
        dll. deleteFirst()
        dll.display()
    elif(ch == 3):
        item = input("Enter the element to search:")
        dll.search(item)
    elif(ch == 4):
        dll.display()
    else:
        break
```

14. Implementation of Circular linked list (CLL)(Traversing the Nodes, searching for a Node, Appending Nodes, and Deleting Nodes):

```
class Node:
    def __init__(self, data = None):
        self.data = data
        self.next = None
class CircularLinkedList:
    def __init__(self):
        self.first = None
    def insertAtEnd(self, data):
        temp = Node(data)
        if(self.first == None):
            self.first = temp
            self.first.next = temp
```

```
        else:
            cur = self.first
            while(cur.next != self.first):
                cur = cur.next
            cur.next = temp
            temp.next = self.first
    def deleteAtEnd(self):
        if(self.first== None):
            print("list is empty")
        elif(self.first.next == self.first):
            print("the deleted item is",self.first.data)
            self.first = None
        else:
            cur=self.first
            while(cur.next != self.first):
                pr = cur
                cur = cur.next
            pr.next = self.first
            print("the deleted item is",cur.data)
    def display(self):
        if(self.first== None):
            print("list is empty")
            return
        cur = self.first
        while(True):
          print(cur.data, end = " ")
          cur = cur.next
          if(cur == self.first):
              break
    def search(self,item):
        if(self.first== None):
            print("list is empty")
            return
        cur = self.first
        while cur.next != self.first:
            if cur.data == item:
                print("Item is present in the linked list")
                return
            else:
                cur = cur.next
        print("Item is not present in the linked list")
#Circular Linked List
cll = CircularLinkedList()
```

```
while(True):
    ch = int(input("\nEnter your choice 1-insert 2-delete 3-search 4-display 5-exit :"))
    if(ch == 1):
        item = input("Enter the element to insert:")
        cll.insertAtEnd(item)
        cll.display()
    elif(ch == 2):
        cll.deleteAtEnd()
        cll.display()
    elif(ch == 3):
        item = input("Enter the element to search:")
        cll.search(item)
    elif(ch == 4):
        cll.display()
    else:
        break
```

15. Implement Stack Data Structure.

```
class Stack:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return len(self.items) == 0
    def push(self,item):
       self.items.append(item)
    def pop(self):
        if self.isEmpty():
            print("Stack is Empty")
        else:
            item = self.items[-1]
            del(self.items[-1])
            print("The popped element is:",item)
    def display(self):
        if self.isEmpty():
            print("Stack is Empty")
        else:
            for i in reversed(self.items):
                print(i)
    def peek(self):
        if self.isEmpty():
            print("Stack is Empty")
        else:
```

```
        print("Top item is ", self.items[-1])
s = Stack()
while(True):
    print("1:push 2:pop 3:display 4:peek 5:exit")
    choice = int(input("Enter your choice:"))
    if choice == 1:
        item = input("Enter the item to push:")
        s.push(item)
    elif choice == 2:
        s.pop()
    elif choice == 3:
        s.display()
    elif choice == 4:
        s.peek()
    else:
        break
```

16. Implement bracket matching using stack.

```
class Stack:
    def __init__(self):
        self.items = []
    def push(self,item):
       self.items.append(item)
    def pop(self):
        if len(self.items) is 0:
            print("Stack is Empty")
        else:
            item = self.items[-1]
            del(self.items[-1])
            return item
def check_brackets(expr):
    s = Stack()
    for token in expr:
        if token in "{[(":
            s.push(token)
        elif token in "}])":
            if  len(s.items) == 0:
                return False
            else:
                left = s.pop()
                if (token == "}" and left != "{") or \
                   (token == "]" and left != "[") or \
```

```
            (token == ")" and left != "(") :
             return False
    if  len(s.items) == 0:
            return True
expr =input("Enter the Expertion:")
result = check_brackets(expr)
if result:
    print("The Given Expression is Valid")
else:
    print("The Given Expression is Invalid")
```

17. Program to demonstrate recursive operations (factorial/ Fibonacci)

a) Factorial

```
def fact(n):
    if n == 1:
        return 1
    else:
        return (n * fact(n-1))
n=int(input("Enter the number:"))
print("The factorial of a number is:",fact(n))
```

b) Fibonacci

```
def fib(n):
    if n<=1:
        return n
    return fib(n-1) + fib(n-2)
n=int(input("Enter the range:"))
print("The fibonacci value is:",fib(n))
```

18. Implement solution for Towers of Hanoi.

```
def towerofhanoi(n, source, destination, auxiliary):
    if n==1:
        print ("Move disk 1 from source",source,"to destination",destination)
        return
    towerofhanoi(n-1, source, auxiliary, destination)
    print ("Move disk",n,"from source",source,"to destination",destination)
    towerofhanoi(n-1, auxiliary, destination, source)
n = 4
towerofhanoi(n,'A','B','C')
```

19. Implement Queue Data Structure.

```
class Queue:
    def __init__(self):
        self.items = []
    def enqueue(self,item):
        self.items.append(item)
    def dequeue(self):
        if self.isEmpty():
            print("Queue is Empty cannot delete")
        else:
            item=self.items.pop(0)
            print("Deleted Item is:",item)
    def display(self):
        if self.isEmpty():
            print("Queue is Empty")
        else:
            print(self.items)
    def length(self):
        return len(self.items)
    def isEmpty(self):
        return len(self.items) == 0
q = Queue()
while True:
    print("1:Enqueue 2:Dequeue 3:Display 4:Length 5:Exit")
    choice = int(input("Enter your choice:"))
    if choice==1:
        item=input("Enter the element:")
        q.enqueue(item)
    elif choice==2:
        q.dequeue()
    elif choice==3:
        q.display()
    elif choice==4:
        n = q.length()
        print("Length of the queue is ",n)
    elif choice==5:
        break
```

20. Implement Priority Queue Data Structure.

```
class PriorityQueueEntry:
    def __init__(self,value,priority):
```

```
        self.v = value
        self.p = priority
class PriorityQueue:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return len(self.items) == 0
    def length(self):
        return len(self.items)
    def enqueue(self,value,priority):
        item  = PriorityQueueEntry(value,priority)
        self.items.append(item)
    def dequeue(self):
        if self.isEmpty():
            print("Queue is empty cannot delete")
        else:
            highest = self.items[0].p
            index = 0
            for i in range(0,self.length()):
                if highest > self.items[i].p:
                    highest = self.items[i].p
                    index = i
            del_item = self.items.pop(index)
            print("Deleted item is ",del_item.v)
    def display(self):
         if self.isEmpty():
             print("Queue is empty")
         else:
             for x in range(0,self.length()):
               print(self.items[x].v,":",self.items[x].p)
pq = PriorityQueue()
while(True):
    print("1:Enqueue 2:Dequeue 3:Display 4:Length 5:Exit")
    choice = int(input("Enter your choice:"))
    if choice == 1:
        value =  input("Enter the item to insert:")
        priority = int(input("Enter the priority:"))
        pq.enqueue(value,priority)
    if choice == 2:
        pq.dequeue()
    if choice == 3:
        pq.display()
    if choice == 4:
```

```
        n = pq.length()
        print("Length of queue is:",n)
    if choice == 5:
        break
```

21. Implement Binary Search Tree and its operations using list.

```
class Node:
    def __init__(self,value):
        self.data = value
        self.left = None
        self.right =None
class BinarySearchTree:
    def __init__(self):
        self.root=None
    def insert(self,value):
        newNode=Node(value)
        if self.root is None:
            self.root = newNode
        else:
            curNode = self.root
            while curNode is not None:
                if value < curNode.data:
                    if curNode.left is None:
                        curNode.left=newNode
                        break
                    else:
                        curNode = curNode.left
                else:
                    if curNode.right is None:
                        curNode.right=newNode
                        break
                    else:
                        curNode=curNode.right
    def preorder(self, rt):
        print(rt.data, end=" ")
        if rt.left is not None:
            self.preorder(rt.left)
        if rt.right is not None:
            self.preorder(rt.right)
    def postorder(self, rt):
        if rt.left is not None:
            self.postorder(rt.left)
```

```
        if rt.right is not None:
            self.postorder(rt.right)
        print(rt.data, end=" ")
    def inorder(self, rt):
        if rt.left is not None:
            self.inorder(rt.left)
        print(rt.data, end=" ")
        if rt.right is not None:
            self.inorder(rt.right)
bst = BinarySearchTree()
ls = [25,10,35,20,65,45,24]
for i in ls:
    bst.insert(i)
print("\nPre-order traversal is:")
bst.preorder(bst.root)
print("\nPost-order traversal is:")
bst.postorder(bst.root)
print("\nIn-order traversal is:")
bst.inorder(bst.root)
```

22. Implementations of BFS.

```
class Queue:
    def __init__(self):
        self.items=[]
    def enqueue(self,value):
        self.items.append(value)
    def dequeue(self):
        if len(self.items) != 0:
            return self.items.pop(0)
    def isEmpty(self):
        return len(self.items) == 0
class Node:
    def __init__(self,value):
        self.data = value
        self.left = None
        self.right = None
class BinarySearchTree:
    def __init__(self):
        self.root = None
    def insert(self,value):
        newNode=Node(value)
```

```
            if self.root is None:
                self.root = newNode
            else:
                curNode = self.root
                while curNode is not None:
                    if value < curNode.data:
                        if curNode.left is None:
                            curNode.left=newNode
                            break
                        else:
                            curNode = curNode.left
                    else:
                        if curNode.right is None:
                            curNode.right=newNode
                            break
                        else:
                            curNode=curNode.right
    def BFS(root):
        q = Queue()
        q.enqueue(root)
        while q.isEmpty() != True:
            node=q.dequeue()
            print(node.data,end=" ")
            if node.left is not None:
                q.enqueue(node.left)
            if node.right is not None:
                q.enqueue(node.right)
    bst = BinarySearchTree()
    ls = [25,10,35,20,5,30,40]
    for i in ls:
        bst.insert(i)
    print("Breadth First Search Traversal:")
    BFS(bst.root)
```

23. Implementations of DFS.

```
class Stack:
    def __init__(self):
        self.items=[]
    def push(self,value):
        self.items.append(value)
    def pop(self):
        if len(self.items) != 0:
```

```
        return self.items.pop()
    def isEmpty(self):
        return len(self.items) == 0
class Node:
    def __init__(self,value):
        self.data = value
        self.left = None
        self.right = None
class BinarySearchTree:
    def __init__(self):
        self.root = None
    def insert(self,value):
        newNode=Node(value)
        if self.root is None:
            self.root = newNode
        else:
            curNode = self.root
            while curNode is not None:
                if value < curNode.data:
                    if curNode.left is None:
                        curNode.left=newNode
                        break
                    else:
                        curNode = curNode.left
                else:
                    if curNode.right is None:
                        curNode.right=newNode
                        break
                    else:
                        curNode=curNode.right
def DFS(root):
    s = Stack()
    s.push(root)
    while s.isEmpty() != True:
        node=s.pop()
        print(node.data,end=" ")
        if node.right is not None:
            s.push(node.right)
        if node.left is not None:
            s.push(node.left)
bst = BinarySearchTree()
ls = [25,10,35,20,5,30,40]
for i in ls:
```

```
    bst.insert(i)
print("Depth First Search Traversal:")
DFS(bst.root)
```

24. Implement Hash functions.

```
class Hash:
    def __init__(self):
        self.buckets=[[],[],[],[],[]]
    def insert(self,key):
        bindex = key % 5
        self.buckets[bindex].append(key)
        print(key,"inserted in Bucket No.",bindex+1)
    def search(self,key):
        bindex = key % 5
        if key in self.buckets[bindex]:
            print(key,"present in bucket No.",bindex+1)
        else:
            print(key,"is not present in any of the buckets")
    def display(self):
        for i in range(0,5):
            print("\nBucket No.",i+1,end=":")
            for j in self.buckets[i]:
                print(j,end="->")
hsh = Hash()
while True:
    print("\nHash operations 1.Insert 2.Search 3.Display 4.Quit")
    ch=int(input("Enter your choice:"))
    if ch == 1:
        key=int(input("Enter key to be inserted:"))
        hsh.insert(key)
    elif ch == 2:
        key=int(input("\nEnter key to be searched:"))
        hsh.search(key)
    elif ch == 3:
        hsh.display()
    else:
        break
```