DATA STRUCTURE

EX.NO:01

IMPLEMENTING ANY ONE PYTHON DATA STRUCTURE

AIM:

To write a program to implement any one python data structure with the following operations A) Create B) Add elements C) Access elements D) Remove elements.

ALGORITHM:

1. Start the program.
2. Create an empty list.
3. Add an element using append().
4. Access elements using index values.
5. Remove elements using remove() and pop().
6. Display the results.
7. Stop the program

PROGRAM:

```
numbers = []
print("Initial List:", numbers)
numbers.append(10)
numbers.append(20)
numbers.append(30)
numbers.append(40)
numbers.append(50)
numbers.append(60)
numbers.append(70)
numbers.insert(1, 15)
print("List after adding element:", numbers)
print("First element:", numbers[0])
print("Last element:", numbers[-1])
print("All elements:", numbers)
for num in numbers:
    print(num, end="")
    print()
numbers.remove(20)
remove_item = numbers.pop()
print("Removed item using pop():", remove_item)
print("List after removing elements:", numbers)
```

RESULT:

Thus, the above python program is executed and the output is obtained.

EX.NO:02

IMPLEMENTING A SINGLY LINKED LIST

AIM:

To write a python program to implement a singly linked list a) Create a singly linked list b) Add element to singly linked list c) Remove element from singly linked list.

ALGORITHM:

1. Start the program
2. Define a node with data and next reference
3. Create an empty linked list
4. Insert nodes at the beginning of the list
5. Remove a specified node from the list
6. Display the linked list
7. Stop the program

PROGRAM:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class SinglyLinkedList:
    def __init__(self):
        self.head = None
    def create(self, data):
        self.head = Node(data)
        print(f"List created with head node: {data}")
    def addstart(self, data):
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node
        print(f"Added node with value {data} at the start")
    def remove(self, data):
        current = self.head
        if not current:
            print("List is empty")
            return
        if current.data == data:
            self.head = current.next
            print(f"Removed element {data} from head")
            return
        prev = None
        while current and current.data != data:
            prev = current
            current = current.next
        if not current:
            print(f"Element {data} not found in the list")
            return
        prev.next = current.next
        print(f"Removed element {data}")
    def display(self):
        if not self.head:
            print("The list is empty")
            return
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
sll = SinglyLinkedList()
while True:
    print("\n1. Create list")
    print("2. Add to start of list")
    print("3. Remove from list")
    print("4. Display list")
    print("5. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter the value for the head node: "))
        sll.create(data)
    elif choice == 2:
        data = int(input("Enter the value to add to start: "))
        sll.addstart(data)
    elif choice == 3:
        data = int(input("Enter the value to remove: "))
        sll.remove(data)
    elif choice == 4:
        print("Current Linked List:")
        sll.display()
    elif choice == 5:
        print("Exiting...")
        break
    else:
        print("Invalid choice. Please try again.")
```

RESULT:

Thus, the above python program is executed and the output is obtained.

EX.NO:03

IMPLEMENTING STACK

AIM:

To write a python program to implement stack.

ALGORITHM:

1. Start the program
2. Create an empty stack using list

3. Push elements into the stack
4. Pop elements from the stack
5. Check if stack is empty
6. Display stack elements
7. Stop the program

PROGRAM:

```
class Stack:
    def __init__(self):
        self.stack = []
    def push(self, item):
        self.stack.append(item)
        print(str(item) + " pushed to Stack.")
    def pop(self):
        if not self.isempty():
            ritem = self.stack.pop()
            print(str(ritem) + " popped from Stack.")
            return ritem
        else:
            print("Stack is empty! Cannot pop an element.")
            return None
    def isempty(self):
        return len(self.stack) == 0
    def display(self):
        if not self.isempty():
            print("Stack elements from top to bottom:")
            for item in reversed(self.stack):
                print(item)
        else:
            print("Stack is empty!")
# Main Program
s = Stack()
while True:
    print("\nMenu:")
    print("1. Push element to Stack")
    print("2. Pop element from Stack")
    print("3. Display Stack")
    print("4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        item = int(input("Enter the value to push onto the Stack: "))
        s.push(item)
    elif choice == 2:
        s.pop()
    elif choice == 3:
        s.display()
    elif choice == 4:
        print("Exiting...")
        break
    else:
        print("Invalid choice! Please select a valid option.")
```

RESULT:

Thus, the above python program is executed and the output is obtained.

EX.NO:04

IMPLEMENTING QUEUE

AIM:

To Write a python program to implement queue.

ALGORITHM:

1. Start the program
2. Create an empty queue
3. Insert elements using enqueue
4. Remove elements using dequeue
5. Check if queue is empty
6. Display queue elements
7. Stop the program

PROGRAM:

```
class Queue:
    def __init__(self):
        self.queue = []
    def enqueue(self, item):
        self.queue.append(item)
        print(f"{item} enqueued to queue.")
    def dequeue(self):
        if not self.isempty():
            ritem = self.queue.pop(0)   # remove from front
            print(f"{ritem} dequeued from queue.")
            return ritem
        else:
            print("Queue is empty! Cannot dequeue an element.")
            return None
    def isempty(self):
        return len(self.queue) == 0
    def display(self):
        if not self.isempty():
            print("Queue elements from front to rear:")
            for item in self.queue:
                print(item, end=" ")
            print()
        else:
            print("Queue is empty!")
q = Queue()
while True:
    print("\nMenu:")
    print("1. Enqueue element to queue")
    print("2. Dequeue element from queue")
    print("3. Display queue")
    print("4. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        item = int(input("Enter the value to enqueue into the queue: "))
        q.enqueue(item)
    elif choice == 2:
        q.dequeue()
    elif choice == 3:
        q.display()
    elif choice == 4:
        print("Exiting...")
        break
    else:
        print("Invalid choice! Please select a valid option.")
```

RESULT:

Thus, the above python program is executed and the output is obtained.

EX.NO:05

PRE-ORDER TRAVERSAL OF A BINARY TREE

AIM:

To write the python program for pre-order traversal of a binary tree.

ALGORITHM:

1. Start the program
2. Create a tree node class
3. Insert elements into the binary tree
4. Traverse the root node
5. Traverse the left subtree
6. Traverse the right subtree
7. Stop the program

PROGRAM:

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
def insert_tree(root, value):
    if root is None:
        return TreeNode(value)
    if value < root.value:
        root.left = insert_tree(root.left, value)
    else:
        root.right = insert_tree(root.right, value)
    return root
def preorder(root):
    if root:
        print(root.value, end=" ")
        preorder(root.left)
        preorder(root.right)
# Main program
root = None
n = int(input("Enter the number of nodes in the binary tree: "))
for i in range(n):
    value = int(input("Enter the value for the node: "))
    root = insert_tree(root, value)
print("\nPre-order Traversal of the Binary Tree:")
preorder(root)
```

RESULT:

Thus, the above python program is executed and the output is obtained.

EX.NO:06

BUBBLE SORT

AIM:

To write a python program to implement bubble sort.

ALGORITHM:

1. Start the program
2. Compare adjacent elements
3. Swap elements if needed
4. Repeat comparison for all elements
5. Reduce comparison range
6. Continue until sorted
7. Stop the program

PROGRAM:

```
def bubble_sort(a):
    n = len(a)
    for i in range(n):
        for j in range(0, n - i - 1):
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
n = int(input("Enter the number of elements in the list: "))
a = []
for i in range(n):
    element = int(input(f"Enter element {i + 1}: "))
    a.append(element)
print("Original list:", a)
bubble_sort(a)
print("Sorted list:", a)
```

RESULT:

Thus, the above python program is executed and the output is obtained.

EX.NO:07

LINEAR SEARCH

AIM:

To write a python program to implement linear search.

ALGORITHM:

1. Start the program
2. Read the list elements
3. Read the search element
4. Compare search element with list items
5. Continue till element is found
6. Display the position

7. Stop the program

PROGRAM:

```
def linear_search(a, x):
    for i in range(len(a)):
        if a[i] == x:
            return i
    return -1
n = int(input("Enter the number of elements in the list: "))
ll = []
for i in range(n):
    element = int(input(f"Enter element {i + 1}: "))
    ll.append(element)
value = int(input("Enter the value to search for: "))
pos = linear_search(ll, value)
if pos != -1:
    print(f"Value found at position {pos}.")
else:
    print("Value not found.")
```

RESULT:

Thus, the above python program is executed and the output is obtained.

EX.NO:08

BINARY SEARCH

AIM:

To write a python program to implement binary search.

ALGORITHM:

1. Start the program
2. Sort the list
3. Find the middle element
4. Compare search element with middle
5. Search left or right half
6. Repeat until found
7. Stop the program

PROGRAM:

```
def binary_search(a, x):
    left = 0
    right = len(a) - 1
    while left <= right:
        mid = (left + right) // 2
        midvalue = a[mid]
        if midvalue == x:
            return mid
        elif midvalue < x:
            left = mid + 1
        else:
            right = mid - 1
    return -1
n = int(input("Enter the number of elements in the sorted list: "))
sortlist = []
for i in range(n):
    element = int(input(f"Enter element {i + 1}: "))
    sortlist.append(element)
x = int(input("Enter the target value to search for: "))
pos = binary_search(sortlist, x)
if pos != -1:
    print(f"Value found at index {pos}.")
else:
    print("Value not found.")
```

RESULT:

Thus, the above python program is executed and the output is obtained.