# SUBJECT: CA-505-MJP: Lab Course on CA-502-MJ (Python Programmin And Data Structure)

# Assignment 1

**Q.1) Write a Python Program to Calculate the Average of Numbers in a List.**

**[10 Marks]**

```
numbers = [10, 20, 30, 40, 50]
average = sum(numbers) / len(numbers)
print("Numbers:", numbers)
print("Average:", average)
```

**Output:**

```
Numbers: [10, 20, 30, 40, 50]
Average: 30.0
```

**Q.2) Write a Python program to perform following operations on BST:**

**Insert** and **Display (Inorder Traversal)**
**[20 Marks]**

```
class Node:
  def _init_(self, data):
    self.data = data self.left =
    None self.right = None

class BST:
   def __init__(self):
      self.root = None

  def insert(self, data):
    if not self.root:
      self.root = Node(data)
    else:
      self._insert(self.root, data)

  def _insert(self, current, data): if
    data < current.data:
      if current.left is None:
        current.left = Node(data)
      else:
        self._insert(current.left, data) elif
    data > current.data:
      if current.right is None:
        current.right = Node(data)
      else:
```

```
            self._insert(current.right, data)

    def inorder(self, node): if
        node:
            self.inorder(node.left)
            print(node.data, end=" ")
            self.inorder(node.right)

# Main Program
tree = BST()
for val in [50, 30, 70, 20, 40, 60, 80]:
    tree.insert(val)

print("Inorder Traversal of BST:")
tree.inorder(tree.root)
```

**Output:**

```
Inorder Traversal of BST:
20 30 40 50 60 70 80
```

# Assignment 2

**Q.1) Write a Python program which accepts 6 integer values and prints "DUPLICATES" if any of the values entered are duplicates otherwise it prints "ALL UNIQUE".**

**[10 Marks]**

```
numbers = []
for i in range(6):
    n = int(input(f"Enter number {i+1}: "))
    numbers.append(n)

if len(numbers) != len(set(numbers)):
    print("DUPLICATES")
else:
    print("ALL UNIQUE")
```

**Output:**

```
Enter number 1: 32
Enter number 2: 10
Enter number 3: 45
Enter number 4: 90
Enter number 5: 45
Enter number 6: 6
DUPLICATES
```

**Q.2) Write a Python program to perform following operations on BST:**

**Create, Search, and Display (Inorder Traversal)**
**[20 Marks]**

```
class Node:
  def _init_(self, data):
    self.data = data
    self.left = self.right = None

class BST:
  def __init__(self):
    self.root = None

  def insert(self, data):
    self.root = self._insert(self.root, data)

  def _insert(self, root, data):
    if root is None:
      return Node(data)
    if data < root.data:
      root.left = self._insert(root.left, data) else:
      root.right = self._insert(root.right, data) return
    root

  def search(self, root, key):
    if root is None or root.data == key:
      return root
    if key < root.data:
      return self.search(root.left, key)
```

```
        return self.search(root.right, key)

    def inorder(self, root):
        if root:
            self.inorder(root.left)
            print(root.data, end=" ")
            self.inorder(root.right)

# Main
tree = BST()
for val in [50, 30, 70, 20, 40, 60, 80]:
    tree.insert(val)

print("Inorder Traversal:")
tree.inorder(tree.root)

key = 60
if tree.search(tree.root, key):
    print(f"\n{key} found in BST.")
else:
    print(f"\n{key} not found in BST.")
```

**Output:**

```
Inorder Traversal:
20 30 40 50 60 70 80
60 found in BST.
```

# Assignment 3

**Q.1) Write a Python program to add and remove operation on set.**

**[10 Marks]**

**Program:**

```
numbers = set()
numbers.add(10)
numbers.add(20)
numbers.add(30)
print("After adding elements:", numbers)

numbers.remove(20)
print("After removing 20:", numbers)

numbers.discard(50)
print("After discarding non-existent 50:", numbers)

numbers.update([40, 50])
print("After adding multiple elements:", numbers)
```

**Output:**

```
After adding elements: {10, 20, 30}
After removing 20: {10, 30}
After discarding non-existent 50: {10, 30}
After adding multiple elements: {40, 10, 50, 30}
```

**Q.2) Write a Python program to perform following operations on Binary Search Tree:**

**Create, Count Non-leaf Nodes, and Display (Inorder Traversal)**
**[20 Marks]**

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        self.root = self._insert(self.root, data)

    def _insert(self, root, data):
        if root is None:
            return Node(data)
        if data < root.data:
            root.left = self._insert(root.left, data) else:
            root.right = self._insert(root.right, data) return
        root

    def inorder(self, root): if
        root:
            self.inorder(root.left)
            print(root.data, end=" ")
            self.inorder(root.right)
```

```
    def count_non_leaf(self, root):
        if root is None or (root.left is None and root.right is None): return 0
        return 1 + self.count_non_leaf(root.left) + self.count_non_leaf(root.right)

# Main
tree = BST()
for val in [50, 30, 70, 20, 40, 60, 80]:
    tree.insert(val)

print("Inorder Traversal:")
tree.inorder(tree.root)
print("\nNon-leaf Nodes:", tree.count_non_leaf(tree.root))
```

**Output:**

```
Inorder Traversal:
20 30 40 50 60 70 80
Non-leaf Nodes: 3
```

# Assignment 4

**Q.1) Write a Python program to find maximum and the minimum value in a set.**

**[10 Marks]**

**Program:**

```
numbers = {12, 45, 23, 78, 56, 10, 5}

print("Set elements:", numbers)
print("Maximum value:", max(numbers))
print("Minimum value:", min(numbers))
```

**Output:**

Set elements: {5, 10, 12, 45, 23, 56, 78}

Maximum value: 78

Minimum value: 5

**Q.2) Write a Python program to perform following operations on Binary Search Tree:**

**i. Create**
**ii. Count leaf nodes**
**iii. Traversal (Preorder / Inorder / Postorder) [20 Marks]**

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        self.root = self._insert(self.root, data)

    def _insert(self, root, data):
        if root is None:
            return Node(data)
        if data < root.data:
            root.left = self._insert(root.left, data) else:
            root.right = self._insert(root.right, data) return
        root

    def inorder(self, node): if
        node:
            self.inorder(node.left)
```

```
            print(node.data, end=" ")
            self.inorder(node.right)

    def preorder(self, node):
        if node:
            print(node.data, end=" ")
            self.preorder(node.left)
            self.preorder(node.right)

    def postorder(self, node):
        if node:
            self.postorder(node.left)
            self.postorder(node.right)
            print(node.data, end=" ")

    def count_leaf_nodes(self, node):
        if node is None:
            return 0
        if node.left is None and node.right is None:
            return 1
        return self.count_leaf_nodes(node.left) + self.count_leaf_nodes(node.right)

# Main
tree = BST()
for val in [50, 30, 70, 20, 40, 60, 80]:
    tree.insert(val)

print("Inorder Traversal:")
tree.inorder(tree.root)
```

```
print("\nPreorder Traversal:")
tree.preorder(tree.root)
print("\nPostorder Traversal:")
tree.postorder(tree.root)
print("\nNumber of Leaf Nodes:", tree.count_leaf_nodes(tree.root))
```

**Output:**

```
Inorder Traversal:
20 30 40 50 60 70 80
Preorder Traversal:
50 30 20 40 70 60 80
Postorder Traversal:
20 40 30 60 80 70 50
Number of Leaf Nodes: 4
```

# Assignment 5

**Q.1) Write a Python program to create an array of 'n' integers and display the array elements. Access individual elements through indexes.**

**[10 Marks]**

```
n = int(input("Enter number of elements: ")) arr
= array('i', [])

for i in range(n):
    val = int(input(f"Enter element {i+1}: "))
    arr.append(val)
print("\nArray elements:", arr.tolist())
print("Accessing elements through indexes:")
for i in range(len(arr)):
    print(f"Element at index {i}: {arr[i]}")
```

**Output:**

```
Enter number of elements: 5
Enter element 1: 10
Enter element 2: 20
Enter element 3: 30
Enter element 4: 40
Enter element 5: 50
Array elements: [10, 20, 30, 40, 50]
Accessing elements through indexes:
Element at index 0: 10
Element at index 1: 20
Element at index 2: 30
Element at index 3: 40
Element at index 4: 50
```

**Q.2) Write a Python program to perform following operations on BST:**

**i. Create**
**ii. Delete**
**iii. Traversal (Inorder / Preorder / Postorder) [20 Marks]**

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        self.root = self._insert(self.root, data)

    def _insert(self, root, data):
        if root is None:
            return Node(data)
        if data < root.data:
            root.left = self._insert(root.left, data) else:
            root.right = self._insert(root.right, data) return
        root

    def delete(self, root, key): if
        root is None:
```

```
            return root
        if key < root.data:
            root.left = self.delete(root.left, key) elif
        key > root.data:
            root.right = self.delete(root.right, key) else:
            if root.left is None:
                return root.right
            elif root.right is None:
                return root.left
            temp = self._min_value_node(root.right) root.data =
            temp.data
            root.right = self.delete(root.right, temp.data) return
        root

    def _min_value_node(self, node): while
        node.left:
            node = node.left
        return node

    def inorder(self, root):
        if root:
            self.inorder(root.left)
            print(root.data, end=" ")
            self.inorder(root.right)

# Main
tree = BST()
for val in [50, 30, 70, 20, 40, 60, 80]:
```

```
    tree.insert(val)

print("Inorder before deletion:")
tree.inorder(tree.root)

tree.root = tree.delete(tree.root, 70)

print("\nInorder after deletion:")
tree.inorder(tree.root)
```

**Output:**

```
Inorder before deletion:
20 30 40 50 60 70 80
Inorder after deletion:
20 30 40 50 60 80
```

# Assignment 6

**Q.1) Write a Python program to get the number of occurrences of specified elements in an array.**

**[10 Marks]**

```
from array import array

arr = array('i', [1, 2, 3, 2, 4, 2, 5])
x = 2

print("Array:", arr.tolist())
print(f"Occurrences of {x}: {arr.count(x)}")
```

**Output:**

Array: [1, 2, 3, 2, 4, 2, 5]

Occurrences of 2: 3

**Q.2) Write a Python program to perform following operations on Binary Search Tree:**

**i. Create**
**ii. Count total nodes**
**iii. Traversal**
**(Inorder) [20 Marks]**

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        self.root = self._insert(self.root, data)

    def _insert(self, root, data):
        if root is None:
            return Node(data)
        if data < root.data:
            root.left = self._insert(root.left, data) else:
            root.right = self._insert(root.right, data) return
        root

    def count_nodes(self, node): if
        node is None:
            return 0
```

```
        return 1 + self.count_nodes(node.left) + self.count_nodes(node.right)

    def inorder(self, node):
        if node:
            self.inorder(node.left)
            print(node.data, end=" ")
            self.inorder(node.right)

# Main
tree = BST()
for val in [45, 25, 70, 20, 40, 60, 80]:
    tree.insert(val)

print("Inorder Traversal:")
tree.inorder(tree.root)
print("\nTotal Nodes:", tree.count_nodes(tree.root))
```

**Output:**

```
Inorder Traversal:
20 25 40 45 60 70 80
Total Nodes: 7
```

# Assignment 7

**Q.1) Write a Python program to reverse the order of the items in the array.**

**[10 Marks]**

```
from array import array

arr = array('i', [10, 20, 30, 40, 50])
print("Original Array:", arr.tolist())

arr.reverse()
print("Reversed Array:", arr.tolist())
```

**Output:**

Original Array: [10, 20, 30, 40, 50]

Reversed Array: [50, 40, 30, 20, 10]

**Q.2) Write a Python program to perform following operations on BST:**

**Create and Display (Inorder Traversal)**
**[20 Marks]**

```
class Node:
  def _init_(self, data):
    self.data = data
    self.left = self.right = None

class BST:
  def __init__(self):
    self.root = None

  def insert(self, data):
    self.root = self._insert(self.root, data)

  def _insert(self, root, data):
    if root is None:
      return Node(data)
    if data < root.data:
      root.left = self._insert(root.left, data) else:
      root.right = self._insert(root.right, data) return
    root

  def inorder(self, node): if
    node:
      self.inorder(node.left)
      print(node.data, end=" ")
      self.inorder(node.right)
```

```
# Main
tree = BST()
for val in [40, 20, 60, 10, 30, 50, 70]:
    tree.insert(val)

print("Inorder Traversal of BST:")
tree.inorder(tree.root)
```

**Output:**

```
Inorder Traversal of BST:
10 20 30 40 50 60 70
```

# Assignment 8

**Q.1) Write a Python program to find sum of all the elements in a list.**

**[10 Marks]**

**Program:**

```
numbers = [10, 20, 30, 40, 50]
print("List:", numbers)
print("Sum of elements:", sum(numbers))
```

**Output:**

```
List: [10, 20, 30, 40, 50]
Sum of elements: 150
```

**Q.2) Write a Python program to perform following operations on Binary Search Tree:**

**i. Insert**
**ii. Delete**
**iii. Display (Preorder / Inorder / Postorder) [20 Marks]**

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

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        self.root = self._insert(self.root, data)

    def _insert(self, root, data):
        if root is None:
            return Node(data)
        if data < root.data:
            root.left = self._insert(root.left, data) else:
            root.right = self._insert(root.right, data) return
        root

    def delete(self, root, key): if
        root is None:
            return root
```

```
    if key < root.data:
        root.left = self.delete(root.left, key) elif
    key > root.data:
        root.right = self.delete(root.right, key) else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left
        temp = self._min_value_node(root.right) root.data =
        temp.data
        root.right = self.delete(root.right, temp.data) return
    root

def _min_value_node(self, node): while
    node.left:
        node = node.left
    return node

def inorder(self, root): if
    root:
        self.inorder(root.left)
        print(root.data, end=" ")
        self.inorder(root.right)

def preorder(self, root): if
    root:
        print(root.data, end=" ")
        self.preorder(root.left)
```

```
            self.preorder(root.right)
    def postorder(self, root):
        if root:
            self.postorder(root.left)
            self.postorder(root.right)
            print(root.data, end=" ")

# Main
tree = BST()
for val in [50, 30, 70, 20, 40, 60, 80]:
    tree.insert(val)
print("Inorder Traversal:")
tree.inorder(tree.root)
tree.root = tree.delete(tree.root, 70)
print("\nAfter deleting 70:")
tree.inorder(tree.root)
print("\nPreorder Traversal:")
tree.preorder(tree.root)
print("\nPostorder Traversal:")
tree.postorder(tree.root)
```

**Output:**

```
Inorder Traversal:
20 30 40 50 60 70 80
After deleting 70:
20 30 40 50 60 80
Preorder Traversal:
50 30 20 40 80 60
Postorder Traversal:
20 40 30 60 80 50
```

# Assignment 9

**Q.1) Write a Python function to calculate the factorial of a number. The function accepts the number as an argument.**

**[10 Marks]**

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

num = int(input("Enter a number: "))
print(f"Factorial of {num} is {factorial(num)}")
```

**Output:**

Enter a number: 5

Factorial of 5 is 120

**Q.2) Write a Python program to search an element using Linear Search.**

**[20 Marks]**

```
def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i
    return -1

arr = [10, 20, 30, 40, 50]
key = int(input("Enter element to search: "))

result = linear_search(arr, key) if
result != -1:
    print(f"Element {key} found at index {result}") else:
    print("Element not found")
```

**Output:**

Enter element to search: 30

Element 30 found at index 2

# Assignment 10

**Q.1) Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x : x*x).**

**[10 Marks]**

```
n = int(input("Enter a number: ")) d
= {}


for x in range(1, n + 1):
   d[x] = x * x


print("Generated Dictionary:", d)
```

**Output:**

Enter a number: 5

Generated Dictionary: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

**Q.2) Write a Python program to search an element using Binary Search.**

**[20 Marks]**

```
def binary_search(arr, key):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1
    return -1

arr = [10, 20, 30, 40, 50]
key = int(input("Enter element to search: "))

result = binary_search(arr, key) if
result != -1:
    print(f"Element {key} found at index {result}") else:
    print("Element not found")
```

**Output:**

Enter element to search: 40

Element 40 found at index 3

---

# Assignment 11

**Q.1) Write a program to generate Fibonacci numbers using function.**

**[10 Marks]**

```
def fibonacci(n):
    a, b = 0, 1
    print("Fibonacci Series:")
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b

num = int(input("Enter number of terms: "))
fibonacci(num)
```

**Output:**

Enter number of terms: 7

Fibonacci Series:

0 1 1 2 3 5 8

**Q.2) Write a Python program to sort given numbers using Bubble Sort algorithm. [20 Marks]**

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n - 1):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

arr = [64, 25, 12, 22, 11]
print("Original array:", arr)

bubble_sort(arr)
print("Sorted array:", arr)
```

**Output:**

Original array: [64, 25, 12, 22, 11]

Sorted array: [11, 12, 22, 25, 64]

# Assignment 12

**Q.1) Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x).**

**[10 Marks]**

```
n = int(input("Enter a number: "))
d = {x: x * x for x in range(1, n + 1)}

print("Generated Dictionary:", d)
```

**Output:**

Enter a number: 5

Generated Dictionary: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

**Q.2) Write a Python program to implement sorting using Merge Sort algorithm. [20 Marks]**

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0
        while i < len(left_half) and j < len(right_half): if
            left_half[i] < right_half[j]:
                arr[k] = left_half[i] i
                += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
```

```
        j += 1
        k += 1

arr = [38, 27, 43, 3, 9, 82, 10]
print("Original Array:", arr)

merge_sort(arr)
print("Sorted Array:", arr)
```

**Output:**

Original Array: [38, 27, 43, 3, 9, 82, 10]

Sorted Array: [3, 9, 10, 27, 38, 43, 82]

# Assignment 13

**Q.1) Write a Python script to sort (ascending and descending) a dictionary by value. [10 Marks]**

```
d = {'apple': 50, 'banana': 20, 'mango': 30, 'orange': 10}

asc = dict(sorted(d.items(), key=lambda x: x[1]))
desc = dict(sorted(d.items(), key=lambda x: x[1], reverse=True))

print("Original Dictionary:", d)
print("Ascending Order:", asc)
print("Descending Order:", desc)
```

**Output:**

```
Original Dictionary: {'apple': 50, 'banana': 20, 'mango': 30, 'orange': 10}
Ascending Order: {'orange': 10, 'banana': 20, 'mango': 30, 'apple': 50}
Descending Order: {'apple': 50, 'mango': 30, 'banana': 20, 'orange': 10}
```

**Q.2) Write a Python program to implement sorting using Quick Sort algorithm.**

**[20 Marks]**

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot] right =
    [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

arr = [10, 7, 8, 9, 1, 5]
print("Original Array:", arr)

sorted_arr = quick_sort(arr)
print("Sorted Array:", sorted_arr)
```

**Output:**

Original Array: [10, 7, 8, 9, 1, 5]

Sorted Array: [1, 5, 7, 8, 9, 10]

# Assignment 14

**Q.1) Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x : x*x).**

**[10 Marks]**

**Program:**

```
# Program to generate dictionary of numbers and their squares


n = int(input("Enter a number: "))

d = {x: x * x for x in range(1, n + 1)}


print("Generated Dictionary:", d)
```

**Output:**

```
Enter a number: 5

Generated Dictionary: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

**Q.2) Write a Python program to implement sorting using Insertion Sort algorithm. [20 Marks]**

```
def insertion_sort(arr): for
   i in range(1, len(arr)):
      key = arr[i]
      j = i - 1
      while j >= 0 and key < arr[j]: arr[j
         + 1] = arr[j]
         j -= 1
       arr[j + 1] = key


  arr = [12, 11, 13, 5, 6]
print("Original Array:", arr)

insertion_sort(arr)
print("Sorted Array:", arr)
```

**Output:**

Original Array: [12, 11, 13, 5, 6]

Sorted Array: [5, 6, 11, 12, 13]

# Assignment 15

**Q.1) Write a Python program to combine two dictionaries adding values for common keys.**

**Sample Input:**

d1 = {'a':100, 'b':200, 'c':300}

d2 = {'a':300, 'b':200, 'd':400}

**Expected Output:**

{'a':400, 'b':400, 'c':300, 'd':400}

**[10 Marks]**

```
from collections import Counter

d1 = {'a': 100, 'b': 200, 'c': 300}
d2 = {'a': 300, 'b': 200, 'd': 400}

combined = Counter(d1) + Counter(d2)
print("Combined Dictionary:", combined)
```

**Output:**

Combined Dictionary: Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})

**Q.2) Write a Python class named Rectangle constructed by length and breadth with two methods which compute area and perimeter of the rectangle.**

**[20 Marks]**

```
class Rectangle:
   def _init_(self, length, breadth):
      self.length = length self.breadth =
      breadth

   def area(self):
      return self.length * self.breadth

   def perimeter(self):
      return 2 * (self.length + self.breadth)

# Main
l = float(input("Enter length of rectangle: "))
b = float(input("Enter breadth of rectangle: "))

rect = Rectangle(l, b)
print("Area of Rectangle:", rect.area())
print("Perimeter of Rectangle:", rect.perimeter())
```

**Output:**

Enter length of rectangle: 5

Enter breadth of rectangle: 3

Area of Rectangle: 15.0

Perimeter of Rectangle: 16.0

# Assignment 16

**Q.1) Write a Python program to create a list of tuples with the first element as the number and the second element as the square of the number. Also display the original list in reverse.**

**[10 Marks]**

```
n = int(input("Enter number of elements: ")) lst
= [(x, x ** 2) for x in range(1, n + 1)]


print("List of tuples:", lst)
print("Reversed List:", lst[::-1])
```

**Output:**

Enter number of elements: 5

List of tuples: [(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

Reversed List: [(5, 25), (4, 16), (3, 9), (2, 4), (1, 1)]

**Q.2) Write a Python code for static implementation of Stack.**

**[20 Marks]**

```
class Stack:
    def __init__(self, size):
        self.size = size
        self.stack = [None] * size
        self.top = -1

    def push(self, item):
        if self.top == self.size - 1:
            print("Stack Overflow!")
        else:
            self.top += 1
            self.stack[self.top] = item
            print(f"Pushed {item} into stack.")

    def pop(self):
        if self.top == -1:
            print("Stack Underflow!")
        else:
            item = self.stack[self.top]
            self.top -= 1
            print(f"Popped {item} from stack.")

    def display(self):
        if self.top == -1:
            print("Stack is empty.")
        else:
```

```
        print("Stack elements:", self.stack[:self.top + 1])


# Main
s = Stack(5)
s.push(10)
s.push(20)
s.push(30)
s.display()
s.pop()
s.display()
```

**Output:**

```
Pushed 10 into stack.
Pushed 20 into stack.
Pushed 30 into stack.
Stack elements: [10, 20, 30]
Popped 30 from stack.
Stack elements: [10, 20]
```

# Assignment 17

**Q.1) Write a Python Program to Calculate the Average of Numbers in a Given List. [10 Marks]**

```
numbers = [10, 20, 30, 40, 50]
print("List of numbers:", numbers)

average = sum(numbers) / len(numbers)
print("Average of numbers:", average)
```

**Output:**

```
List of numbers: [10, 20, 30, 40, 50]
Average of numbers: 30.0
```

**Q.2) Write a Python code for static implementation of Queue.**

**[20 Marks]**

**Program:**

```
# Static Implementation of Queue


class Queue:
  def __init__(self, size):
    self.size = size
    self.queue = [None] * size
    self.front = self.rear = -1


  def enqueue(self, item):
    if self.rear == self.size - 1:
      print("Queue Overflow!")
    else:
      if self.front == -1:
        self.front = 0
      self.rear += 1
      self.queue[self.rear] = item
      print(f"Enqueued: {item}")


  def dequeue(self):
    if self.front == -1 or self.front > self.rear:
      print("Queue Underflow!")
    else:
      print(f"Dequeued: {self.queue[self.front]}") self.front
      += 1


  def display(self):
```

```
        if self.front == -1 or self.front > self.rear:
            print("Queue is empty.")
        else:
            print("Queue elements:", self.queue[self.front:self.rear + 1])


# Main
q = Queue(5)
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
q.display()
q.dequeue()
q.display()
```

**Output:**

```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: [10, 20, 30]
Dequeued: 10
Queue elements: [20, 30]
```

# Assignment 18

**Q.1) Write a Python code to copy elements 44 and 55 from the following tuple into a new tuple tuple1 = (11, 22, 33, 44, 55, 66); also display the same tuple in reverse order.**

**[10 Marks]**

```
tuple1 = (11, 22, 33, 44, 55, 66)

new_tuple = (tuple1[3], tuple1[4])


print("Original Tuple:", tuple1)

print("New Tuple with 44 and 55:", new_tuple)

print("Reversed Tuple:", tuple1[::-1])
```

**Output:**

```
Original Tuple: (11, 22, 33, 44, 55, 66)

New Tuple with 44 and 55: (44, 55)

Reversed Tuple: (66, 55, 44, 33, 22, 11)
```

**Q.2) Write a Python code for simple implementation of Priority Queue.**

**[20 Marks]**

```
import heapq

class PriorityQueue:
    def __init__(self):
        self.queue = []

    def insert(self, priority, item):
        heapq.heappush(self.queue, (priority, item))
        print(f"Inserted item '{item}' with priority {priority}")

    def remove(self):
        if not self.queue:
            print("Priority Queue is empty!")
        else:
            item = heapq.heappop(self.queue)
            print(f"Removed item '{item[1]}' with priority {item[0]}")

    def display(self):
        print("Current Priority Queue:", self.queue)

# Main
pq = PriorityQueue()
pq.insert(3, "Task A")
pq.insert(1, "Task B")
pq.insert(2, "Task C")
pq.display()
```

```
pq.remove()

pq.display()
```

**Output:**

```
Inserted item 'Task A' with priority 3

Inserted item 'Task B' with priority 1

Inserted item 'Task C' with priority 2

Current Priority Queue: [(1, 'Task B'), (3, 'Task A'), (2, 'Task C')]

Removed item 'Task B' with priority 1

Current Priority Queue: [(2, 'Task C'), (3, 'Task A')]
```

# Assignment 19

**Q.1) Write a Python program to get the 5th element from front and 5th element from last of a tuple.**

**[10 Marks]**

```
tuple1 = (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)

print("Tuple:", tuple1)


fifth_front = tuple1[4] fifth_last

= tuple1[-5]


print("5th element from front:", fifth_front) print("5th

element from last:", fifth_last)
```

**Output:**

```
Tuple: (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)

5th element from front: 50 5th

element from last: 60
```

**Q.2) Write a Python code for simple implementation of Priority Queue.**

**[20 Marks]**

**Program:**

```
# Priority Queue implementation using heapq module

import heapq

class PriorityQueue:
    def __init__(self):
        self.queue = []

    def insert(self, priority, item):
        heapq.heappush(self.queue, (priority, item))
        print(f"Inserted: {item} with priority {priority}")

    def delete(self):
        if not self.queue:
            print("Queue is empty!")
        else:
            item = heapq.heappop(self.queue)
            print(f"Deleted: {item[1]} (priority {item[0]})")

    def display(self):
        print("Current Queue:", self.queue)

# Main
pq = PriorityQueue()
pq.insert(2, "Clean Room") pq.insert(1,
"Do Homework")
```

```
pq.insert(3, "Watch TV")
pq.display()
pq.delete()
pq.display()
```

**Output:**

```
Inserted: Clean Room with priority 2
Inserted: Do Homework with priority 1
Inserted: Watch TV with priority 3
Current Queue: [(1, 'Do Homework'), (2, 'Clean Room'), (3, 'Watch TV')] Deleted: Do
Homework (priority 1)
Current Queue: [(2, 'Clean Room'), (3, 'Watch TV')]
```

# Assignment 20

**Q.1) Write a program to display the following pattern:**

**[10 Marks]**

1

2 3

4 5 6

7 8 9 10

```
num = 1
for i in range(1, 5):
    for j in range(i):
        print(num, end=" ")
        num += 1
    print()
```

**Output:**

1

2 3

4 5 6

7 8 9 10

**Q.2) Write a Python code for static stack implementation.**

**[20 Marks]**

```
class Stack:
    def __init__(self, size):
        self.size = size
        self.stack = [None] * size
        self.top = -1

    def push(self, item):
        if self.top == self.size - 1:
            print("Stack Overflow!")
        else:
            self.top += 1
            self.stack[self.top] = item
            print(f"Pushed {item} onto stack.")

    def pop(self):
        if self.top == -1:
            print("Stack Underflow!")
        else:
            item = self.stack[self.top]
            self.top -= 1
            print(f"Popped {item} from stack.")

    def display(self):
        if self.top == -1:
            print("Stack is empty.")
        else:
```

```
        print("Stack elements:", self.stack[:self.top + 1])


# Main
s = Stack(5)
s.push(10)
s.push(20)
s.push(30)
s.display()
s.pop()
s.display()
```

**Output:**

```
Pushed 10 onto stack.
Pushed 20 onto stack.
Pushed 30 onto stack.
Stack elements: [10, 20, 30]
Popped 30 from stack.
Stack elements: [10, 20]
```

# Assignment 21

**Q.1) Write a Python program which accepts 6 integer values and prints "DUPLICATES" if any of the values entered are duplicates otherwise it prints "ALL UNIQUE".**

**[10 Marks]**

```
numbers = []
for i in range(6):
    n = int(input(f"Enter number {i+1}: "))
    numbers.append(n)

if len(numbers) != len(set(numbers)):
    print("DUPLICATES")
else:
    print("ALL UNIQUE")
```

**Output:**

```
Enter number 1: 32
Enter number 2: 10
Enter number 3: 45
Enter number 4: 90
Enter number 5: 45
Enter number 6: 6
DUPLICATES
```

**Q.2) Show the static implementation of Queue using Python.**

**[20 Marks]**

```
class Queue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = self.rear = -1

    def enqueue(self, item):
        if self.rear == self.size - 1:
            print("Queue Overflow!")
        else:
            if self.front == -1:
                self.front = 0
            self.rear += 1
            self.queue[self.rear] = item
            print(f"Enqueued: {item}")

    def dequeue(self):
        if self.front == -1 or self.front > self.rear:
            print("Queue Underflow!")
        else:
            print(f"Dequeued: {self.queue[self.front]}") self.front
            += 1

    def display(self):
        if self.front == -1 or self.front > self.rear:
            print("Queue is empty.")
```

```
        else:
            print("Queue elements:", self.queue[self.front:self.rear + 1])


# Main
q = Queue(5)
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
q.display()
q.dequeue()
q.display()
```

**Output:**

```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: [10, 20, 30]
Dequeued: 10
Queue elements: [20, 30]
```

# Assignment 22

**Q.1) Write a Python program to find repeated items in a tuple.**

**[10 Marks]**

```
tuple1 = (10, 20, 30, 20, 40, 10, 50, 30)
print("Tuple:", tuple1)

repeated = []
for item in tuple1:
    if tuple1.count(item) > 1 and item not in repeated:
        repeated.append(item)

print("Repeated items:", repeated)
```

**Output:**

Tuple: (10, 20, 30, 20, 40, 10, 50, 30)

Repeated items: [10, 20, 30]

**Q.2) Show the static implementation of Stack using Python.**

**[20 Marks]**

```
class Stack:
    def __init__(self, size):
        self.size = size
        self.stack = [None] * size
        self.top = -1

    def push(self, item):
        if self.top == self.size - 1:
            print("Stack Overflow!")
        else:
            self.top += 1
            self.stack[self.top] = item
            print(f"Pushed {item} onto stack.")

    def pop(self):
        if self.top == -1:
            print("Stack Underflow!")
        else:
            item = self.stack[self.top]
            self.top -= 1
            print(f"Popped {item} from stack.")

    def display(self):
        if self.top == -1:
            print("Stack is empty.")
        else:
```

```
        print("Stack elements:", self.stack[:self.top + 1])


# Main
s = Stack(5)
s.push(10)
s.push(20)
s.push(30)
s.display()
s.pop()
s.display()
```

**Output:**

```
Pushed 10 onto stack.
Pushed 20 onto stack.
Pushed 30 onto stack.
Stack elements: [10, 20, 30]
Popped 30 from stack.
Stack elements: [10, 20]
```

# Assignment 23

**Q.1) Write a Python Program to Calculate the Average of Numbers in a Given List.**

**[10 Marks]**

```
numbers = [10, 20, 30, 40, 50]
print("List:", numbers)

average = sum(numbers) / len(numbers)
print("Average of the list is:", average)
```

**Output:**

List: [10, 20, 30, 40, 50]

Average of the list is: 30.0

**Q.2) Write a Python code for static queue implementation.**

**[20 Marks]**

```
class Queue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = self.rear = -1

    def enqueue(self, item):
        if self.rear == self.size - 1:
            print("Queue Overflow!")
        else:
            if self.front == -1:
                self.front = 0
            self.rear += 1
            self.queue[self.rear] = item
            print(f"Enqueued: {item}")

    def dequeue(self):
        if self.front == -1 or self.front > self.rear:
            print("Queue Underflow!")
        else:
            print(f"Dequeued: {self.queue[self.front]}") self.front
            += 1

    def display(self):
        if self.front == -1 or self.front > self.rear:
            print("Queue is empty.")
```

```
        else:
            print("Queue elements:", self.queue[self.front:self.rear + 1])


# Main
q = Queue(5)
q.enqueue(10)
q.enqueue(20)
q.enqueue(30)
q.display()
q.dequeue()
q.display()
```

**Output:**

```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: [10, 20, 30]
Dequeued: 10
Queue elements: [20, 30]
```

# Assignment 24

**Q.1) Write a Python Program to Calculate the Sum of Numbers in a List.**

**[10 Marks]**

```
numbers = [5, 10, 15, 20, 25]
print("List:", numbers)

total = sum(numbers)
print("Sum of numbers:", total)
```

**Output:**

```
List: [5, 10, 15, 20, 25]
Sum of numbers: 75
```

**Q.2) Write a Program to Search an Element using Binary Search.**

**[20 Marks]**

```
def binary_search(arr, key):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1
    return -1
# Main
arr = [10, 20, 30, 40, 50, 60]
key = int(input("Enter element to search: "))
print("Sorted Array:", arr)
result = binary_search(arr, key)

if result != -1:
    print(f"Element found at index {result}") else:
    print("Element not found in array")
```

**Output:**

Enter element to search: 40

Sorted Array: [10, 20, 30, 40, 50, 60]

Element found at index 3

# Assignment 25

**Q.1) Write a Python program to get the number of occurrences of a specified element in an array.**

**[10 Marks]**

```
arr = array('i', [1, 2, 3, 2, 4, 2, 5])

print("Array:", arr)


num = int(input("Enter element to count: "))

count = arr.count(num)

print(f"Number {num} occurs {count} times in the array.")
```

**OUTPUT:**

```
Array: array('i', [1, 2, 3, 2, 4, 2, 5])

Enter element to count: 2

Number 2 occurs 3 times in the array.
```

**Q.2) Write a Python Program to Sort Given Numbers using Bubble Sort Algorithm.**

**[20 Marks]**

**Program:**

```
# Bubble Sort Algorithm


def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
         if arr[j] > arr[j + 1]:
            arr[j], arr[j + 1] = arr[j + 1], arr[j]


    arr = [64, 34, 25, 12, 22, 11, 90]
print("Original Array:", arr)


bubble_sort(arr)
print("Sorted Array:", arr)
```

**Output:**

Original Array: [64, 34, 25, 12, 22, 11, 90]

Sorted Array: [11, 12, 22, 25, 34, 64, 90]