PRACTICAL No.1(A) [LISTS]

1.Write a Python program to create a list and perform the following methods.

1. insert()
2. remove()
3. append()
4. len()
5. pop()
6. clear()

```
# Create an empty list
my_list = []
# a) insert() - Insert 'apple' at index 0
my_list.insert(0, 'apple')
print("After insert:", my_list)

# b) remove() - Remove 'apple' from the list
my_list.remove('apple')
print("After remove:", my_list)

# c) append() - Append 'banana' to the list
my_list.append('banana')
print("After append:", my_list)

# d) len() - Get the length of the list
print("Length of the list:", len(my_list))

# e) pop() - Pop the last item in the list
my_list.pop()
print("After pop:", my_list)

# f) clear() - Clear the list
my_list.clear()
print("After clear:", my_list)
```

OUTPUT:

```
After insert:
['apple']
After remove:
[]
After append:
['banana']
Length of the list: 1
After pop:
[]
After clear:
[]
```

2.Write a Python program to create a list and extract the values into variables by unpacking.

```
# Create a list with three values
my_list = [10, 20, 30]

# Unpack the values into variables
a, b, c = my_list
print("Unpacked values:", a, b, c)
```

OUTPUT:

```
Unpacked values: 10 20 30
```

3.Write a Python program to create a list of string items and sort the list alphabetically.

```
# Create a list of strings
string_list = ["banana", "apple", "cherry", "date"]

# Sort the list alphabetically
string_list.sort()
print("Alphabetically sorted list:", string_list)
```

OUTPUT:

```
Alphabetically sorted list: ['apple', 'banana', 'cherry', 'date']
```

4.Write a Python program to sort the above list descending.

```
# Sort the list in descending order
string_list.sort(reverse=True)
print("Descending sorted list:", string_list)
```

OUTPUT:

```
Descending sorted list: ['date', 'cherry', 'banana', 'apple']
```

5.Write a Python program to create a list of numerical items and sort the list numerically.

```
# Create a list of numbers
num_list = [5, 3, 8, 1, 2]

# Sort the list numerically
num_list.sort()
print("Numerically sorted list:", num_list)
```

OUTPUT:

```
Numerically sorted list: [1, 2, 3, 5, 8]
```

6.Write a Python program to sort the above list descending.

```
# Sort the list in descending order
num_list.sort(reverse=True)
print("Numerically descending sorted list:", num_list)
```

OUTPUT:

```
Numerically descending sorted list: [8, 5, 3, 2, 1]
```

PRACTICAL No.1(B) [TUPLES]

1.Write Python program to perform following operations on Tuples:

a. Create Tuple
b. Access Tuple
c. Update Tuple
d. Delete Tuple
e. Remove item Tuple

```
# a) Create Tuple
my_tuple = (1, 2, 3, 4, 5)
print("Created Tuple:", my_tuple)

# b) Access Tuple
print("Accessing an element (index 2):", my_tuple[2])

# c) Update Tuple (Tuples are immutable, so we can't directly update them; however, we can create a new tuple)
# Let's add a new item by creating a new tuple
updated_tuple = my_tuple + (6,)
print("Updated Tuple:", updated_tuple)

# d) Delete Tuple
del updated_tuple  # This will delete the entire tuple
# print(updated_tuple)  # Uncommenting this line would raise an error because updated_tuple is deleted

# e) Remove Item from Tuple (Tuples are immutable, so we can't remove an item directly; convert to list temporarily)
temp_list = list(my_tuple)
temp_list.remove(3)  # Removing the item '3'
my_tuple = tuple(temp_list)
print("Tuple after removing an item:", my_tuple)
```

OUTPUT:

```
Created Tuple: (1, 2, 3, 4, 5)
Accessing an element (index 2): 3
Updated Tuple: (1, 2, 3, 4, 5, 6)
Tuple after removing an item: (1, 2, 4, 5)
```

2.Write a Python program to find the repeated items of a tuple.

```
def find_repeated_items(tup):
    repeated_items = set([item for item in tup if tup.count(item) > 1])
    return repeated_items

# Test the function
test_tuple = (1, 2, 3, 4, 2, 5, 3)
print("Repeated items:", find_repeated_items(test_tuple))
```

OUTPUT:

```
Repeated items: {2, 3}
```

3.Write a Python program to iterate through the items and print the values.

```
# Tuple of items
my_tuple = (10, 20, 30, 40, 50)

# Iterate and print each value
for item in my_tuple:
    print(item)
```

OUTPUT:

```
10
20
30
40
50
```

PRACTICAL No.1(C) [DICTIONARY]

1.Write Python program to perform following operations on Dictionaries:

a. Create Dictionary
b. Access Dictionary elements
c. Update Dictionary
d. Delete Set
e. Looping through Dictionary

```
# a) Create Dictionary
my_dict = {'name': 'Sohaib', 'age': 20, 'city': 'Karachi'}
print("Created Dictionary:", my_dict)

# b) Access Dictionary elements
print("Access 'name' element:", my_dict['name'])

# c) Update Dictionary
my_dict['age'] = 20
print("Updated Dictionary:", my_dict)

# d) Delete Set (Assuming it means deleting an element from the dictionary)
del my_dict['city']
print("Dictionary after deletion:", my_dict)

# e) Looping through Dictionary
for key, value in my_dict.items():
    print(f"{key}: {value}")
```

OUTPUT:

```
Created Dictionary: {'name': 'Sohaib', 'age': 20, 'city': 'Karachi'}
Access 'name' element: Sohaib
Updated Dictionary: {'name': 'Sohaib', 'age': 20, 'city': 'Karachi'}
Dictionary after deletion: {'name': 'Sohaib', 'age': 20}
name: Sohaib
age: 20
```

2.Write Python program to get the value of the 2nd key of dictionary.

```
my_dict = {'a': 100, 'b': 200, 'c': 300}
second_key = list(my_dict.keys())[1]
print("Value of 2nd key:", my_dict[second_key])
```

OUTPUT:

```
Value of 2nd key: 200
```

3.Write a Python script to sort (ascending and descending) a dictionary by value.

```
# Dictionary to sort
my_dict = {'apple': 50, 'banana': 20, 'cherry': 30}
# Sort in ascending order
sorted_dict_asc = dict(sorted(my_dict.items(), key=lambda item: item[1]))
print("Ascending order:", sorted_dict_asc)

# Sort in descending order
sorted_dict_desc = dict(sorted(my_dict.items(), key=lambda item: item[1], reverse=True))
print("Descending order:", sorted_dict_desc)
```

OUTPUT:

```
Ascending order: {'banana': 20, 'cherry': 30, 'apple': 50}
Descending order: {'apple': 50, 'cherry': 30, 'banana': 20}
```

4.Write a Python script to create a dictionary that contains three dictionaries.

```
# Creating nested dictionaries
dict1 = {'a': 1, 'b': 2}
dict2 = {'c': 3, 'd': 4}
dict3 = {'e': 5, 'f': 6}
nested_dict = {'dict1': dict1, 'dict2': dict2, 'dict3': dict3}
print("Dictionary with three dictionaries:", nested_dict)
```

OUTPUT:

```
Dictionary with three dictionaries: {'dict1': {'a': 1, 'b': 2}, 'dict2': {'c': 3, 'd': 4}, 'dict3': {'e': 5, 'f': 6}}
```

5.Write a Python script to access the 2nd item of 2nd dictionary.

```
# Accessing the 2nd item of 2nd dictionary in nested_dict
second_item = list(nested_dict['dict2'].values())[1]
print("2nd item of 2nd dictionary:", second_item)
```

OUTPUT:

```
2nd item of 2nd dictionary: 4
```

6.Write a Python script to concatenate following dictionaries to create a new one.

```
dic1 = {1:10, 2:20}

dic2 = {3:30, 4:40}

dic3 = {5:50,6:60}
```

```
# Given dictionaries
dic1 = {1: 10, 2: 20}
dic2 = {3: 30, 4: 40}
dic3 = {5: 50, 6: 60}
# Concatenating dictionaries
concatenated_dict = {**dic1, **dic2, **dic3}
print("Concatenated Dictionary:", concatenated_dict)
```

OUTPUT:

```
Concatenated Dictionary: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
```

PRACTICAL No.2(A) [User Defined Functions]

1.Write a Python function that takes a number as a parameter and check the number is prime or not.

```
def is_prime(number):
    if number <= 1:
        return False
    for i in range(2, int(number ** 0.5) + 1):
        if number % i == 0:
            return False
    return True
# Test the function
print(is_prime(7))
print(is_prime(10))
```

OUTPUT:

```
True
False
```

2.Write a Python function to calculate the factorial of a number (a non- negative integer). The function accepts the number as an argument.

```
def factorial(number):
    if number < 0:
        return "Factorial is not defined for negative numbers."
    elif number == 0 or number == 1:
        return 1
    else:
        result = 1
        for i in range(2, number + 1):
            result *= i
        return result
# Test the function
print(factorial(5))
print(factorial(0))
```

OUTPUT:

```
120
1
```

3.Write a Python function that accepts a string and calculate the number of upper case letters and lower case letters.

```
def count_case(string):
    upper_count = sum(1 for char in string if char.isupper())
    lower_count = sum(1 for char in string if char.islower())
    return {"Upper case": upper_count, "Lower case": lower_count}

# Test the function
result = count_case("Hello World")
print(result)
```

OUTPUT:

```
{'Upper case': 2, 'Lower case': 8}
```

PRACTICAL No.2(B) [MODULES]

1.Write a Python program to create a user defined module that will ask your college name and will display the name of the college.

```
# 1.    Create the module (college_module.py):
# college_module.py

# 2.    Use the module:
import college_module
college_module.ask_college_name()
```

OUTPUT:

```
Enter your college name:  dawood
College Name: dawood
```

2.Write a Python program that will calculate area and circumference of circle using inbuilt Math Module

```
import math

def circle_properties(radius):
    area = math.pi * radius ** 2
    circumference = 2 * math.pi * radius
    return area, circumferenc
# Example usage
radius = float(input("Enter the radius of the circle: "))
area, circumference = circle_properties(radius)
print("Area of Circle:", area)
print("Circumference of Circle:", circumference)
```

OUTPUT:

```
Enter the radius of the circle:  7
Area of Circle: 153.93804002589985
Circumference of Circle: 43.982297150257104
```

3.Write a Python program that will display Calendar of given month using Calendar Module

```
import calendar

def display_calendar(year, month):
    print(calendar.month(year, month))
# Example usage
year = int(input("Enter the year: "))
month = int(input("Enter the month (1-12): "))
display_calendar(year, month)
```

OUTPUT:

```
Enter the year:  2024
Enter the month (1-12):  10
    October 2024
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

4.Write a Python program that will calculate square root of number using Math Module

```
import math

def calculate_square_root(number):
    return math.sqrt(number)
# Example usage
number = float(input("Enter a number: "))
print("Square root of", number, "is:", calculate_square_root(number))
```

OUTPUT:

```
Enter a number:  25
Square root of 25.0 is: 5.0
```

PRACTICAL No.3 [NUMPY & PANDAS]

1.Write a Python program to create two matrices and perform addition, subtraction, multiplication and division operation on matrix.

```
import numpy as np
# Create two matrices
matrix1 = np.array([[1, 2], [3, 4]])
matrix2 = np.array([[5, 6], [7, 8]])
# Perform addition
addition = matrix1 + matrix2
print("Addition:\n", addition)

# Perform subtraction
subtraction = matrix1 - matrix2
print("Subtraction:\n", subtraction)

# Perform element-wise multiplication
multiplication = matrix1 * matrix2
print("Multiplication:\n", multiplication)

# Perform element-wise division
division = matrix1 / matrix2
print("Division:\n", division)
```

OUTPUT:

```
Addition:
 [[ 6  8]
 [10 12]]
Subtraction:
 [[-4 -4]
 [-4 -4]]
Multiplication:
 [[ 5 12]
 [21 32]]
Division:
 [[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

2.Write a NumPy program to generate six random integers between 10 and 30.

```
random_integers = np.random.randint(10, 30, 6)
print("Random integers:", random_integers)
```

OUTPUT:

```
Random integers: [14 27 24 23 21 28]
```

3.Write a python code to read a csv file using pandas module and print the first five and last five lines of a file.

```
import pandas as pd
# Read CSV file
df = pd.read_csv("coffee.csv")

# Print the first five lines
print("First five lines:\n", df.head())

# Print the last five lines
print("Last five lines:\n", df.tail())
```

OUTPUT:

```
First five lines:
          Day Coffee Type  Units Sold
0     Monday    Espresso          25
1     Monday       Latte          15
2    Tuesday    Espresso          30
3    Tuesday       Latte          20
4  Wednesday    Espresso          35
Last five lines:
          Day Coffee Type  Units Sold
9     Friday       Latte          35
10  Saturday    Espresso          45
11  Saturday       Latte          35
12    Sunday    Espresso          45
13    Sunday       Latte          35
```

4.Write a program code to compute summary statistics such as mean, median, mode, standard deviation and variance of the given dataset.

```
from scipy import stats
# Sample dataset
data = [10, 20, 20, 30, 40, 50, 60, 70]

# Mean
mean = np.mean(data)
print("Mean:", mean)

# Median
median = np.median(data)
print("Median:", median)

# Mode
mode = stats.mode(data, keepdims=True).mode[0]  # Ensure mode is returned as an array
print("Mode:", mode)

# Standard Deviation
std_dev = np.std(data)
print("Standard Deviation:", std_dev)

# Variance
variance = np.var(data)
print("Variance:", variance)
```

OUTPUT:

```
Mean: 37.5
Median: 35.0
Mode: 20
Standard Deviation: 19.84313483298443
Variance: 393.75
```

5.Write a python code to read a csv file using pandas module and preprocess the dataset for null, duplicate values.

```
df = pd.read_csv('missing.csv')
df.head()
```

	ID	COUNTRY	COLOR	SKILL	SKILL_POINTS	UTILIZATION	IS_VALID
0	1	France	Signal violet	marksmanship	14.0	0.19240	1
1	1	France	Signal violet	marksmanship	14.0	0.19240	1
2	2	Solomon Islands	Pearl violet	NaN	4.0	NaN	1
3	3	Germany	NaN	calligraphy	12.0	0.88646	0
4	4	Brazil	Rose quartz	archery	9.0	0.34520	1

```
# Check for null values and fill or drop as needed
print("Null Values in Each Column:\n",df.isnull().sum())
```

OUTPUT:

```
Null Values in Each Column:
 ID              0
COUNTRY          0
COLOR            1
SKILL            1
SKILL_POINTS     4
UTILIZATION      4
IS_VALID         0
dtype: int64
```

```
# Check for duplicate values and remove them
duplicates = df.duplicated().sum()
print("Duplicate values:", duplicates)
```

OUTPUT:

```
Duplicate values: 1
```

```
df.drop_duplicates(inplace=True)
print("Duplicate values:",df.duplicated().sum())
```

OUTPUT:

```
Duplicate values: 0
```

```
# Show the cleaned dataset
df
```

	ID	COUNTRY	COLOR	SKILL	SKILL_POINTS	UTILIZATION	IS_VALID
0	1	France	Signal violet	marksmanship	14.0	0.1924	1
4	4	Brazil	Rose quartz	archery	9.0	0.3452	1
5	5	Canada	Denim blue	fencing	15.0	0.6734	1
7	7	Japan	Red poppy	painting	8.0	0.2345	1
9	9	Italy	Olive green	knitting	10.0	0.5566	0
10	10	Argentina	Azure blue	marksmanship	14.0	0.1924	1
11	11	Portugal	Fuchsia pink	dancing	7.0	0.7294	0
12	12	Australia	Turquoise	sculpting	11.0	0.4231	1
14	14	South Africa	Coral red	archery	12.0	0.5530	0
17	17	Mexico	Emerald green	pottery	13.0	0.7845	1
19	19	UK	Ocean blue	painting	9.0	0.4573	1

PRACTICAL No.4 [BREATH FIRST SEARCH]

1. Write a Program to Implement Breadth First Search without goal state using Python.

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        vertex = queue.popleft()
        print(vertex, end=" ")

        for neighbor in graph[vertex]:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)

# Example usage:
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F', 'G'],
    'D': ['B'],
    'E': ['B', 'H'],
    'F': ['C'],
    'G': ['C'],
    'H': ['E']
}
start_vertex = 'A'
print("BFS Traversal:")
bfs(graph, start_vertex)
```

OUTPUT:

```
BFS Traversal:
A B C D E F G H
```

2. Write a Program to Implement Breadth First Search with goal state using Python.

```
from collections import deque

def bfs_with_goal(graph, start, goal):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        current_vertex = queue.popleft()
        print(current_vertex, end=" ")

        if current_vertex == goal:
            print("\nGoal state reached!")
            return

        for neighbor in graph[current_vertex]:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
        print("\nGoal state not reached.")
# Example usage:
graph = {
    '1': ['2', '3'],
    '2': ['1', '4', '5'],
    '3': ['1', '6'],
    '4': ['2'],
    '5': ['2', '7'],
    '6': ['3', '8'],
    '7': ['5'],
    '8': ['6']
}

start_state = '1'
goal_state = '4'
print("\nBFS Traversal:")
bfs_with_goal(graph, start_state, goal_state)
```

OUTPUT:

```
BFS Traversal:
1
Goal state not reached.
2
Goal state not reached.
3
Goal state not reached.
4
Goal state reached!
```