Name: Meet Raut
Batch: S21
Roll number: 2201084

PYTHON

EXPERIMENT: 1-A

Aim: To write a program to implement compound structures like Lists, Tuples, Dictionaries and Sets.

Objective Mapped: LO1

Theory:

In Python, compound data types are fundamental structures that allow programmers to organize, manipulate, and represent data in various ways. These data types, including lists, dictionaries, tuples, and sets, provide versatile tools for handling complex datasets and solving a wide range of programming problems.

1. Lists:

Lists are one of the most commonly used data structures in Python. They are ordered collections of items, where each item can be of any data type, and they can even contain other lists (nested lists). Lists are mutable, meaning their elements can be modified after creation. This flexibility makes lists ideal for tasks such as storing sequences of data, managing dynamic collections, and facilitating iteration and manipulation of elements. With operations like appending, extending, slicing, and sorting, lists provide powerful tools for data organization and manipulation.

2. Dictionaries:

Dictionaries are another essential compound data type in Python. Unlike lists, which are indexed by integer values, dictionaries are indexed by keys, allowing for fast lookup and retrieval of values based on unique identifiers. Each element in a dictionary is a key-value pair, where the key is immutable (typically a string or number) and the value can be of any data type. Dictionaries are highly efficient for mapping relationships between items, representing structured data, and implementing associative arrays. They support operations such as adding, updating, and deleting key-value pairs, making them invaluable for tasks like data modelling, configuration management, and rapid information retrieval.

3. Tuples:

Tuples are ordered collections similar to lists but with one crucial difference: they are immutable, meaning their elements cannot be modified after creation. Tuples are typically used to group together related data elements that should not be changed, such as coordinates, records, or function arguments. While tuples lack the extensive mutability of lists, their immutability provides benefits such as improved performance, data integrity, and protection against unintended modifications. Tuples support operations like indexing, slicing, and unpacking, making them versatile for tasks like data serialization, function return values, and representing fixed sequences of data.

4. Sets:

Sets are unordered collections of unique elements, designed to support mathematical set operations like union, intersection, and difference. Sets are mutable, allowing for dynamic modification of elements, but each element within a set must be hashable and immutable to ensure uniqueness. Sets are particularly useful for tasks requiring membership testing, duplicate removal, or filtering unique elements from a dataset. With operations like adding, removing, and testing for membership, sets provide efficient tools for data deduplication, filtering, and set-based computations.

Code:

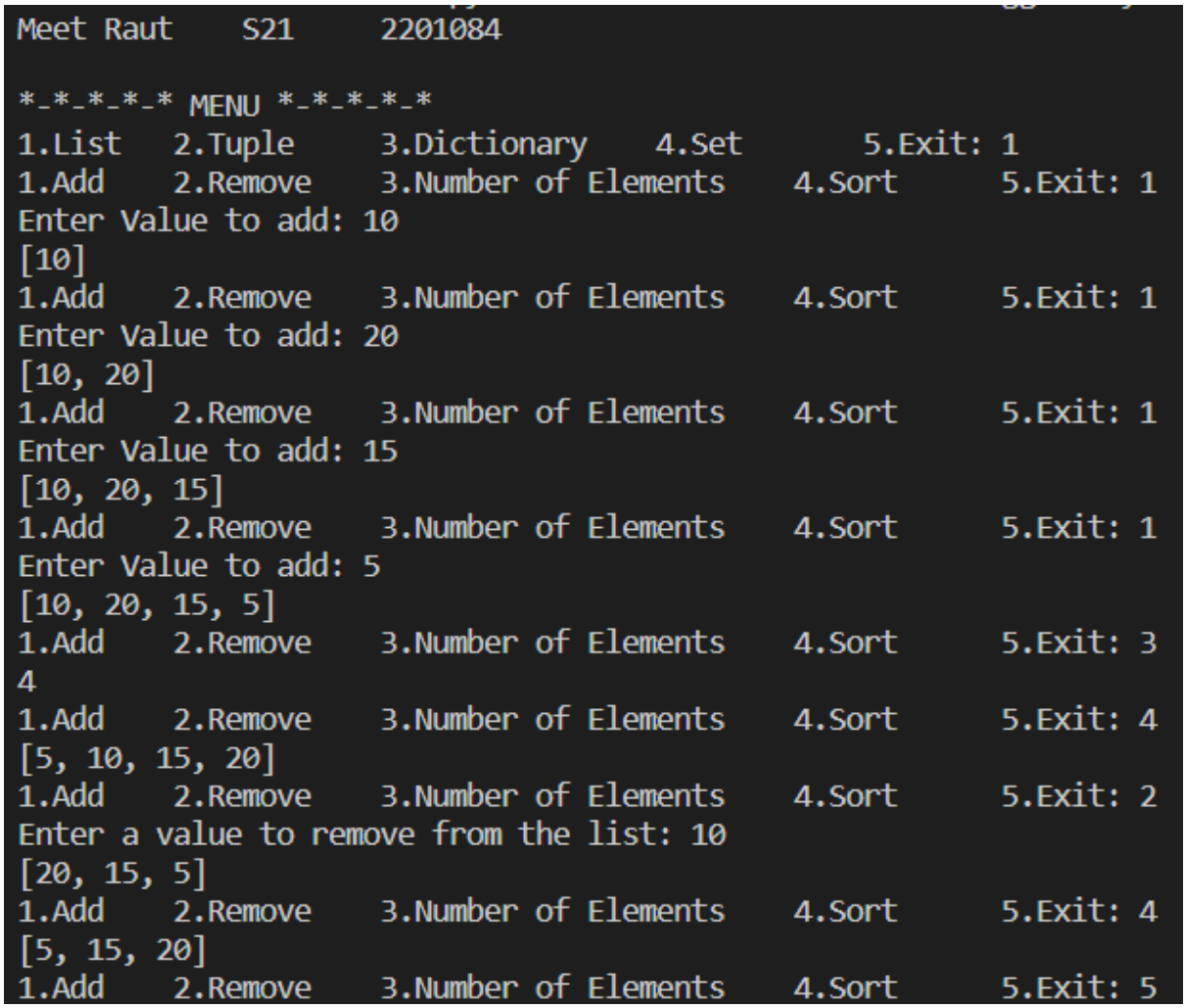
```
while True:
    print("Meet Raut    S21     2201084")
    print("\n*-*-*-*-* MENU *-*-*-*-*")
    n = int(input("1.List   2.Tuple     3.Dictionary    4.Set       5.Exit:
"))

    if n == 1:
        l = []
        while True:
            a = int(input("1.Add    2.Remove    3.Number of
Elements    4.Sort      5.Exit: "))
            if a == 1:
                l.append(int(input("Enter Value to add: ")))
                print(l)
            elif a == 2:
                try:
                    value = int(input("Enter a value to remove from the list:
"))
                    l.remove(value)
                    print(l)
                except ValueError:
                    print("Value not found in the list.")
            elif a == 3:
                print(len(l))
            elif a == 4:
                print(sorted(l))
            else:
                break

    elif n == 2:
        t = tuple(map(int, input("Enter values separated by space:
").split()))
        while True:
            a = int(input("1.Print      2.Number of
Elements    3.Sort      4.Exit: "))
            if a == 1:
                print(t)
            elif a == 2:
                print(len(t))
            elif a == 3:
                print(tuple(sorted(t)))
            else:
                break

    elif n == 3:
        d = {}
```

```
        while True:
            a = int(input("1.Add    2.Remove    3.Number of
Elements    4.Sort      5.Exit: "))
            if a == 1:
                key, value = input("Enter key value pair separated by space:
").split()
                d[key] = value
                print(d)
            elif a == 2:
                try:
                    key = input("Enter a key to remove from the dictionary: ")
                    d.pop(key)
                    print(d)
                except KeyError:
                    print("Key not found in the dictionary.")
            elif a == 3:
                print(len(d))
            elif a == 4:
                print(sorted(d.items()))
            else:
                break

    elif n == 4:
        s = set()
        while True:
            a = int(input("1.Add    2.Remove    3.Number of
Elements    4.Sort      5.Exit: "))
            if a == 1:
                s.add(int(input("Enter Value to add: ")))
                print(s)
            elif a == 2:
                try:
                    value = int(input("Enter a value to remove from the set:
"))
                    s.remove(value)
                    print(s)
                except KeyError:
                    print("Value not found in the set.")
            elif a == 3:
                print(len(s))
            elif a == 4:
                print(sorted(s))
            else:
                break

    elif n == 5:
        break
```

OUTPUT:

Below are the screenshots that show the output of each operation of the data types.

1. LISTS



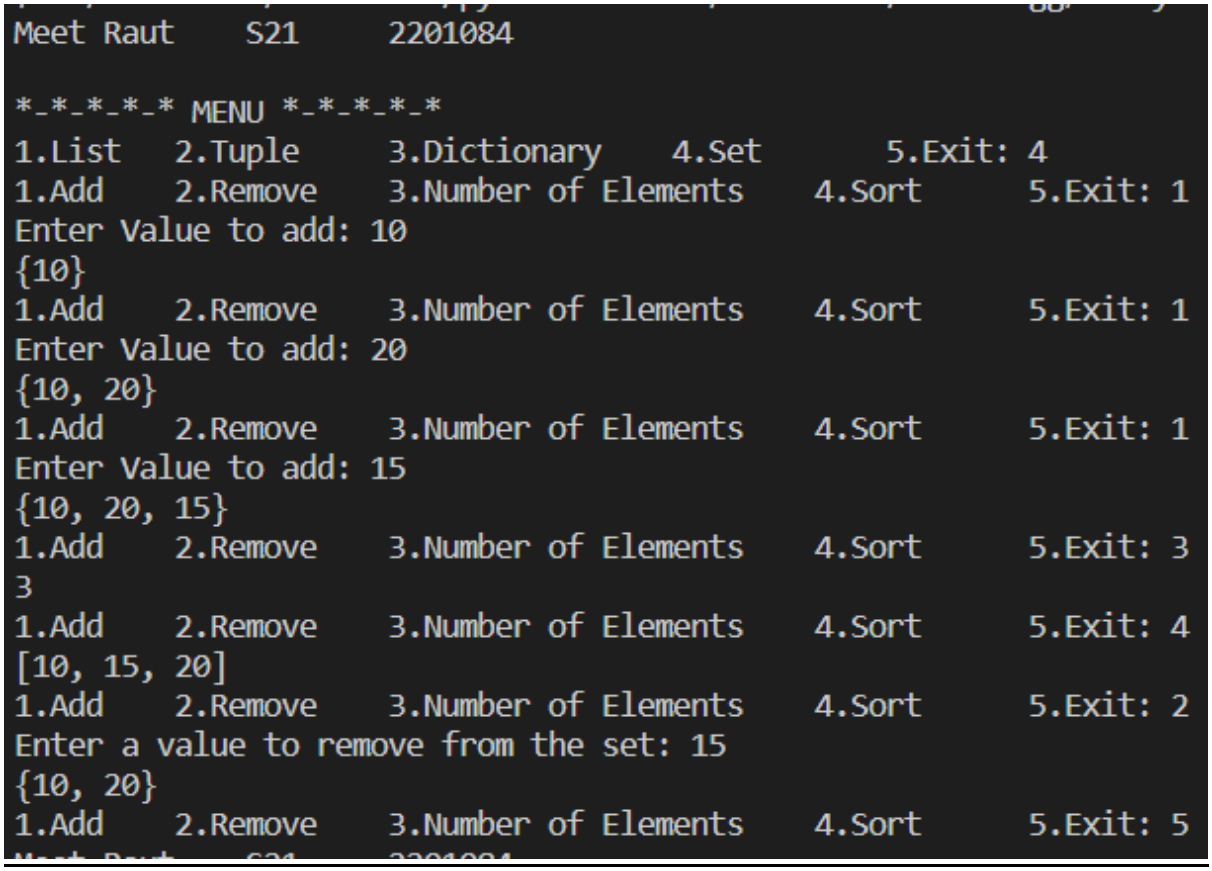
2. TUPLES

```
Meet Raut    S21     2201084

*_*_*_*_* MENU *_*_*_*_*
1.List   2.Tuple     3.Dictionary    4.Set       5.Exit: 2
Enter values separated by space: 10 20 15 5 35 12
1.Print      2.Number of Elements    3.Sort      4.Exit: 1
(10, 20, 15, 5, 35, 12)
1.Print      2.Number of Elements    3.Sort      4.Exit: 3
(5, 10, 12, 15, 20, 35)
1.Print      2.Number of Elements    3.Sort      4.Exit: 2
6
1.Print      2.Number of Elements    3.Sort      4.Exit: 4
```

3. DICTIONARY

```
Meet Raut    S21     2201084

*_*_*_*_* MENU *_*_*_*_*
1.List   2.Tuple     3.Dictionary    4.Set       5.Exit: 3
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 1
Enter key value pair separated by space: 1 Apple
{'1': 'Apple'}
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 1
Enter key value pair separated by space: 2 Banana
{'1': 'Apple', '2': 'Banana'}
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 3
2
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 4
[('1', 'Apple'), ('2', 'Banana')]
1.Add    2.Remove    3.Number of Elements    4.Sort      5.Exit: 2
Enter a key to remove from the dictionary: 1
{'2': 'Banana'}
```

4. SETS



CONCLUSION:

In conclusion, compound data types in Python are essential building blocks for organising and manipulating data effectively. Whether you're working with lists, dictionaries, tuples, or sets, understanding their characteristics, capabilities, and appropriate use cases empowers you to write cleaner, more efficient, and more expressive Python code.