NAME:SUPRIYA SUBHASH VIRKAR CLASS:SE-3 ROLL NO:58

Experiment No. 3
To explore basic data types of python like strings, list, dictionaries and tuples
Date of Performance:
Date of Submission:

Experiment No. 3

Title: To explore basic data types of python like strings, list, dictionaries and tuples.

Aim: To study and explore basic data types of python like strings, list, dictionaries and tuples.

Objective: To introduce basic data types of python

Theory:

Lists: are just like dynamic sized arrays, declared in other languages (vector in C++ and ArrayList in Java). Lists need not be homogeneous always which makes it a most powerful tool in Python.

Tuple: A Tuple is a collection of Python objects separated by commas. In someways a tuple is similar to a list in terms of indexing, nested objects and repetition but a tuple is immutable unlike lists that are mutable.

Set: A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements. Python's set class represents the mathematical notion of a set.

Dictionary: in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key value is provided in the dictionary to make it more optimized.

List, Tuple, Set, and Dictionary are the data structures in python that are used to store and organize the data in an efficient manner.

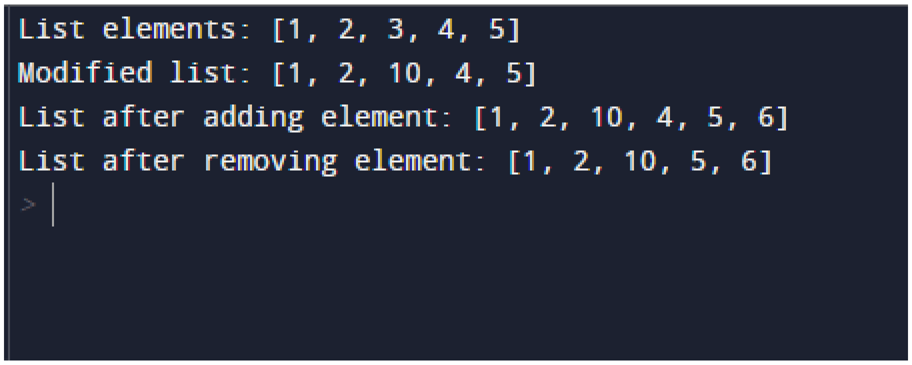
List	**Tuple**	**Set**	**Dictionary**
List is a non-homogeneous data structure which stores the elements in single row and multiple rows and columns	Tuple is also a non-homogeneous data structure which stores single row and multiple rows and columns	Set data structure is also non-homogeneous data structure but stores in single row	Dictionary is also a non-homogeneous data structure which stores key value pairs
List can be represented by [ ]	Tuple can be represented by ( )	Set can be represented by { }	Dictionary can be represented by { }
List allows duplicate elements	Tuple allows duplicate elements	Set will not allow duplicate elements	Set will not allow duplicate elements but keys are not duplicated
List can use nested among all	Tuple can use nested among all	Set can use nested among all	Dictionary can use nested among all
Example: [1, 2, 3, 4, 5]	Example: (1, 2, 3, 4, 5)	Example: {1, 2, 3, 4, 5}	Example: {1, 2, 3, 4, 5}
List can be created using **list()** function	Tuple can be created using **tuple()** function.	Set can be created using **set()** function	Dictionary can be created using **dict()** function.
List is mutable i.e we can make any changes in list.	Tuple is immutable i.e we can not make any changes in tuple	Set is mutable i.e we can make any changes in set. But elements are not duplicated.	Dictionary is mutable. But Keys are not duplicated.
List is ordered	Tuple is ordered	Set is unordered	Dictionary is ordered

		Creating a set
		a=set()
Creating an empty list	Creating an empty Tuple	
l=[]	t=()	b=set(a)

code:

```
#list program

my_list = [1, 2, 3, 4, 5]

print("List elements:", my_list)

my_list[2] = 10

print("Modified list:", my_list)

my_list.append(6)

print("List after adding element:", my_list)

my_list.remove(4)

print("List after removing element:", my_list)
```

output:

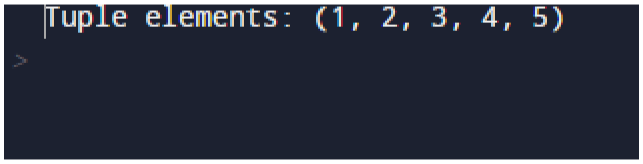


```
List elements: [1, 2, 3, 4, 5]
Modified list: [1, 2, 10, 4, 5]
List after adding element: [1, 2, 10, 4, 5, 6]
List after removing element: [1, 2, 10, 5, 6]
>
```

# Tuple program

my_tuple = (1, 2, 3, 4, 5)

print("Tuple elements:", my_tuple)

output: 

```
Tuple elements: (1, 2, 3, 4, 5)
>
```

# Set program

my_set = {1, 2, 3, 4, 5}

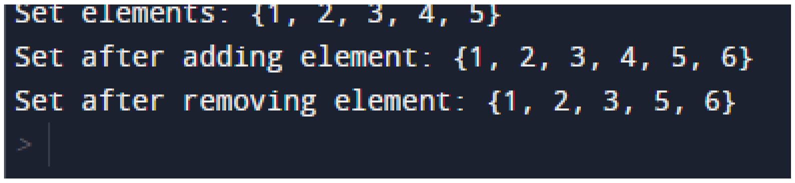
print("Set elements:", my_set)

my_set.add(6)

print("Set after adding element:", my_set)

my_set.remove(4)

print("Set after removing element:", my_set)

output: 

```
Set elements: {1, 2, 3, 4, 5}
Set after adding element: {1, 2, 3, 4, 5, 6}
Set after removing element: {1, 2, 3, 5, 6}
>
```

#dictionary program

my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

print("Dictionary elements:", my_dict)

my_dict['age'] = 26

print("Modified dictionary:", my_dict)

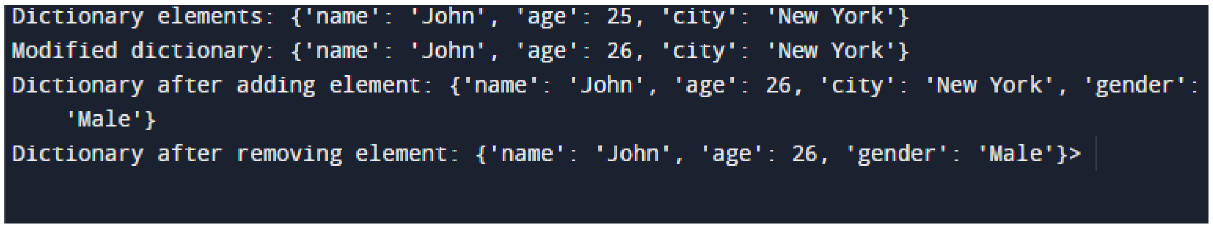
my_dict['gender'] = 'Male'

print("Dictionary after adding element:", my_dict)

my_dict.pop('city')

print("Dictionary after removing element:", my_dict)

output:



```
Dictionary elements: {'name': 'John', 'age': 25, 'city': 'New York'}
Modified dictionary: {'name': 'John', 'age': 26, 'city': 'New York'}
Dictionary after adding element: {'name': 'John', 'age': 26, 'city': 'New York', 'gender':
    'Male'}
Dictionary after removing element: {'name': 'John', 'age': 26, 'gender': 'Male'}>
```

Code:program for list,tuple,set,and dictionary

my_list = [1, 2, 3, 4, 5]

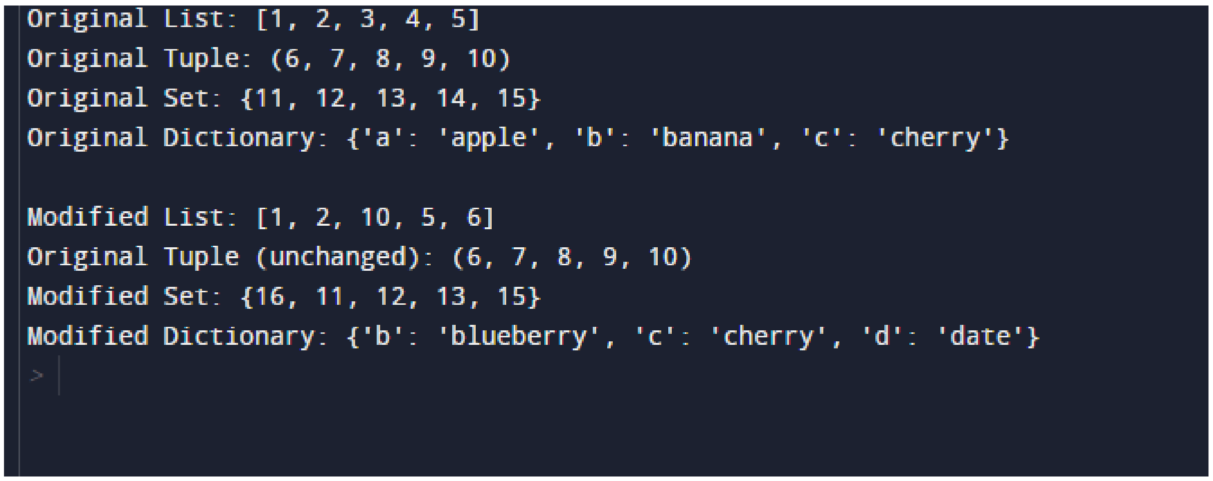
my_tuple = (6, 7, 8, 9, 10)

my_set = {11, 12, 13, 14, 15}

my_dict = {'a': 'apple', 'b': 'banana', 'c': 'cherry'}

```
print("Original List:", my_list)

print("Original Tuple:", my_tuple)

print("Original Set:", my_set)

print("Original Dictionary:", my_dict)

my_list.append(6)

my_list[2] = 10

my_list.remove(4)

my_set.add(16)

my_set.remove(14)

my_dict['d'] = 'date'

my_dict['b'] = 'blueberry'

my_dict.pop('a')

print("\nModified List:", my_list)

print("Original Tuple (unchanged):", my_tuple)

print("Modified Set:", my_set)

print("Modified Dictionary:", my_dict)
```

output:



```
Original List: [1, 2, 3, 4, 5]
Original Tuple: (6, 7, 8, 9, 10)
Original Set: {11, 12, 13, 14, 15}
Original Dictionary: {'a': 'apple', 'b': 'banana', 'c': 'cherry'}

Modified List: [1, 2, 10, 5, 6]
Original Tuple (unchanged): (6, 7, 8, 9, 10)
Modified Set: {16, 11, 12, 13, 15}
Modified Dictionary: {'b': 'blueberry', 'c': 'cherry', 'd': 'date'}
>
```

Conclusion:

- Understanding the basic data types in Python—strings, lists, dictionaries, and tuples—is essential for effective programming.

- Each data type has its unique properties, behaviors, and use cases, making them suitable for different scenarios.

- Choosing the appropriate data type based on the requirements of your program can lead to more efficient and readable code.

- Mastery of these basic data types is fundamental to becoming proficient in Python programming and building more complex data structures and algorithms.