Experiment No. 3
To explore basic data types of python like strings, list, dictionaries and tuples
Name : Vaidehi D. Gadag
Branch/Div.: Comps-1 (C47)
Date of Performance: 31/01/2024
Date of Submission: 07/02/2024

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

LIST

Code:

```
#Creating a list
list=[]
print("Creating a list : ",list)
list=[1,"Koala",2,"Hamster",3,"Cat",4,"Squirrel"]
print("Adding elements : ",list)
#Accesing element from the list
print("Accessing from the list : ",list[3])
#Length of the list
print("Length of the list : ",len(list))
#Making new list
list1=[5,"Duck",6,"Bear",7,"Rabbit"]
print("New List : ",list1)
#Appending the lists
print("Appending the lists : ")
list.append(list1)
print(list)
#Popping the element
list.pop()
print("Pops the list : ",list)
list.pop()
print("Pops the last element : ",list)
```

Output:

```
Creating a list :  []
Adding elements :  [1, 'Koala', 2, 'Hamster', 3, 'Cat', 4, 'Squirrel']
Accessing from the list :  Hamster
Length of the list :  8
New List :  [5, 'Duck', 6, 'Bear', 7, 'Rabbit']
Appending the lists :
[1, 'Koala', 2, 'Hamster', 3, 'Cat', 4, 'Squirrel', [5, 'Duck', 6, 'Bear', 7, 'Rabbit']]
Pops the list :  [1, 'Koala', 2, 'Hamster', 3, 'Cat', 4, 'Squirrel']
Pops the last element :  [1, 'Koala', 2, 'Hamster', 3, 'Cat', 4]
```

TUPLE

Code:

```
#Creating a tuple
tuple=()
print("Creating a tuple : ",tuple)
tuple=("Superman","Batman","Spiderman","Ironman")
print("Adding elements : ",tuple)
#Accesing element from the tuple
print("Accessing from the tuple : ",tuple[3])
#Length of the tuple
print("Length of the tuple : ",len(tuple))
#Making new tuple
tuple1=("Captain America","Hulk","Thor")
print("New tuple : ",tuple1)
#Concatenating the tuples
tuple=tuple + tuple1
print("Concatenating the tuples : ",tuple)
#Removing elements until tuple is empty
while tuple:
   print("Removing elements : ", tuple[1:])
   tuple = tuple[1:]
```

Output:

```
Creating a tuple :  ()
Adding elements :  ('Superman', 'Batman', 'Spiderman', 'Ironman')
Accessing from the tuple :  Ironman
Length of the tuple :  4
New tuple :  ('Captain America', 'Hulk', 'Thor')
Concatenating the tuples :  ('Superman', 'Batman', 'Spiderman', 'Ironman', 'Captain America', 'Hulk', 'Thor')
Removing elements :  ('Batman', 'Spiderman', 'Ironman', 'Captain America', 'Hulk', 'Thor')
Removing elements :  ('Spiderman', 'Ironman', 'Captain America', 'Hulk', 'Thor')
Removing elements :  ('Ironman', 'Captain America', 'Hulk', 'Thor')
Removing elements :  ('Captain America', 'Hulk', 'Thor')
Removing elements :  ('Hulk', 'Thor')
Removing elements :  ('Thor',)
Removing elements :  ()
```

SET

Code:

```
#Creating a Set
set={}
print("Creating a set : ",set)
set={"Tiger Flower","Carolina Allspice","Spirea","Buttercup"}
print("Adding elements : ",set)
#Check if an element is present in the set
if "Spirea" in set:
   print("Spirea is present in the set.")
else:
   print("Spirea is not present in the set.")
#Length of the Set
print("Length of the set : ",len(set))
#Making new Set
set1={"Clematis","Larch","Rumex"}
print("New set : ",set1)
#Joining the sets
set=set.union(set1)
print("Joining the sets : ",set)
#Removing an element
set.remove("Rumex")
print("Removes an element : ",set)
#Popping the random element
set.pop()
print("Pops the random element : ",set)
```

Output:

```
Creating a set :  {}
Adding elements :  {'Carolina Allspice', 'Spirea', 'Tiger Flower', 'Buttercup'}
Spirea is present in the set.
Length of the set :  4
New set :  {'Clematis', 'Rumex', 'Larch'}
Joining the sets :  {'Clematis', 'Carolina Allspice', 'Rumex', 'Tiger Flower', 'Spirea', 'Buttercup',
'Larch'}
Removes an element :  {'Clematis', 'Carolina Allspice', 'Tiger Flower', 'Spirea', 'Buttercup',
'Larch'}
Pops the random element :  {'Carolina Allspice', 'Tiger Flower', 'Spirea', 'Buttercup', 'Larch'}
```

DICTIONARY

Code:

```
#Creating a Dictionary
dict={}
print("Creating a dictionary : ",dict)
dict={1: 'Iron', 2: 'Platinum', 3: 'Copper'}
print("Adding elements : ",dict)
#Accessing the element
x = dict[3]
print("Accessing the element in a dictionary : ",x)
#Length of the dictionary
print("Length of the dictionary : ",len(dict))
#Making new dictionary
dict1={4: 'Gold', 5: 'Silver', 6: 'Bronze'}
print("New dictionary : ",dict1)
#Merging the dictionaries
dict.update(dict1)
print("Merging dictionary : ",dict)
#Empty The Dictionary
dict.clear()
print("Clearing the dictionary : ",dict)
```

Output:

```
Creating a dictionary :  {}
Adding elements :  {1: 'Iron', 2: 'Platinum', 3: 'Copper'}
Accessing the element in a dictionary :  Copper
Length of the dictionary :  3
New dictionary :  {4: 'Gold', 5: 'Silver', 6: 'Bronze'}
Merging dictionary :  {1: 'Iron', 2: 'Platinum', 3: 'Copper', 4: 'Gold', 5: 'Silver', 6: 'Bronze'}
Clearing the dictionary :  {}
```

Conclusion:

Python's rich variety of built-in data types provides flexibility and efficiency for programmers to manipulate data effectively. With dynamically typed variables and automatic type inference, Python makes it easy to work with different types of data.

Python provides several mutable data types, including lists, dictionaries, and sets, which allow for values to be changed after creation. On the other hand, immutable data types, such as integers, floats, tuples, and strings, have values that cannot be changed after creation.

The type of an object can be checked using the type() function, and type casting can be performed using constructors of different data types. Each data type in Python comes with its own set of methods and functions that can be used to manipulate and operate on data of that type efficiently.

Understanding the implementation of basic data types in Python is essential for writing efficient and effective code. By mastering these data types, programmers can take advantage of Python's powerful features and capabilities to build high-quality software applications.