Experiment 02

Write a python program to understand

2.1) Different List, Tuple, Dictionary operations using Built-in functions

2.2) Built-in set and String, Range functions

Roll No.	61
Name	V Krishnasubramaniam
Class	D10-A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language LO2: Interpret advanced data types and functions in python

Aim:

Write a python program to understand
2.1) Different List, Tuple, Dictionary operations using Built-in functions
2.2) Built-in set and String, Range functions

Introduction:

Compound data types:

A compound data type is one that holds multiple independent values i.e. they are made of one or more objects of other datatypes. It provides ways to organize and manage data values of any data type. There are no constants for compound data types. There are 4 compound data types in Python:

1. List
2. Tuple
3. Dictionary
4. Set

List:

The list is an ordered sequence of items or objects. It is one of the most used data types in Python and is very flexible. In Python programming, a list is created by placing all the items (elements) inside a square bracket [], separated by commas. It can have any number of items and they may be of different types (integer, float, string, etc.).

Examples:

```
my_list = [1,2,3,4]             #list of integers
my_list = [1, "Hello",  4.2]    #list with mixed datatypes
```

We can use the index operator [] to access an item in a list. Index starts from 0 till (number of elements - 1). Trying to access an element other than this will raise an IndexError. Python allows negative indexing for its sequences. The index of -1 refers to the last item, and so on. Slicing allows us to access a range of elements from the list. It is done using the slice operator in Python.

Examples:

```
my_list = ['h','e','l','l','o']
print(my_list[0])           #Output: h
print(my_list[5])           #Output: IndexError: list index out of range
print(my_list[-1])          #Output: o
print(my_list[-3])          #Output: l
print(my_list[0:3])         #Output: ['h', 'e', 'l']
```

List are mutable, meaning, their elements can be changed, unlike string or tuple. We can use the assignment operator (=) or use the append() method or use the extend() method. Furthermore, we can insert one item at a desired location by using the method insert().

Examples:

```
my_list = [11,2,3,4]
my_list[0] = 1
print(my_list)              #Output: [1,2,3,4]
my_list.append(5)
print(my_list)              #Output: [1,2,3,4,5]
my_list.extend([6,7,8])
print(my_list)              #Output: [1,2,3,4,5,6,7,8]
my_list = [1,2,3]
my_list.insert(1,4)
print(my_list)              #Output: [1,4,2,3]
```

We can delete one or more items from a list using the keyword del. We can use remove() method to remove the given item or pop() method to remove an item at the given index. We can also use the clear() method to empty a list.

Examples:

```
my_list = ['p','r','o','b','l','e','m']
del my_list[2]
print(my_list)              #Output: ['p', 'r', 'b', 'l', 'e', 'm']
my_list.remove('p')
print(my_list)              #Output: ['r', 'b', 'l', 'e', 'm']
print(my_list.pop(1))       #Output: 'b'
print(my_list)              #Output: ['r', 'l', 'e', 'm']
my_list.clear()
print(my_list)              #Output: []
```

Tuple:

A tuple is a collection of objects which ordered and immutable. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma.

Example:

```
tup1 = (1, 2, 3, 4, 5)
tup2 = "a", "b", "c", "d"
```

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. Python allows negative indexing for tuples.

Example:

```
tup1 = (1, 2, 3, 4, 5, 6, 7)
print(tup1[0])              #Output: 1
```

```
print (tup1[1:5])        #Output: (2, 3, 4, 5)
print(tup1[-6])          #Output: 2
```

Tuples are immutable which means you cannot update or change the values of tuple elements. But, if the element is itself a mutable datatype like list, its nested items can be changed.

Example:

```
tup1 = (4, 2, 3, [6, 5])
tup1[3][0] = 9
print(tup1)              #Output: (4, 2, 3, [9, 5])
```

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded. To explicitly remove an entire tuple, just use the del statement, like, del tup1.

Membership of an element can be tested using the in operator. count() method returns the number of items that is equal its parameter, and index() returns the index of first item that is equal its parameter

Example:

```
my_tuple = ('a','p','p','l','e')
print('a' in my_tuple)          #Output: True
print('g' not in my_tuple)      #Output: True
print(my_tuple.count('p'))      #Output: 2
print(my_tuple.index('l'))      #Output: 3
```

Dictionary:

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair. Dictionaries are optimized to retrieve values when the key is known. Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.
Creating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value. To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. get() method can also be used.
Example:

```
my_dict = {'name':'Jack', 'age': 26}
print(my_dict['name'])          #Output: Jack
print(my_dict.get('age'))       #Output: 26
```

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry.

Example:

```
my_dict = {'name':'Jack', 'age': 26}
my_dict['age'] = 27
my_dict['address'] = 'Downtown'
print(my_dict)          #Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
```

You can either remove individual dictionary elements or clear the entire contents of a dictionary, or by using the pop() function. You can also delete entire dictionary in a single operation. To explicitly remove an entire dictionary, just use the del statement.

Example:

```
my_dict = {'name':'Jack', 'age': 26, 'class':'First'}
del my_dict['name']
my_dict.pop(age)        #Output: 26
print(my_dict)          #Output: {'class':'First'}
my_dict.clear()
print(my_dict)          #Output: {}
del my_dict
```

Set:

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed). However, the set itself is mutable. Sets can be used to perform mathematical set operations.

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set(). We cannot access individual values in a set. We can only access all the elements together as shown above.

Example:

```
my_set1 = {1,2,2,3}
my_set2 = set([1,2,3,4,5])
print(my_set1)          #Output: {1,2,3}
print(my_set2)          #Output: {1,2,3,4,5}
```

We can add single element using the add() method and multiple elements using the update() method. A particular item can be removed from set using methods, discard() and remove(). We can also remove all items from a set using clear().

Example:

```
my_set = {1,2,3}
my_set.add(4)
print(my_set)           #Output: {1,2,3,4}
my_set.update([5,6])
print(my_set)           #Output: {1,2,3,4,5,6}
```

```
my_set.discard(4)
print(my_set)          #Output: {1,2,3,5,6}
my_set.remove(6)
print(my_set)          #Output: {1,2,3,5}
my_set.clear()
print(my_set)          #Output: set()
```

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference.

Example:

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
print(A | B)               #Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A.union(B))          #Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A & B)               #Output: {4, 5}
print(A.intersection(B))   #Output: {4, 5}
print(A - B)               #Output: {1, 2, 3}
print(B.difference(A))     #Output: {8, 6, 7}
print(A ^ B)               #Output: {1, 2, 3, 6, 7, 8}
```

Results:

Operations on Lists:

```
>>> my_list = ['h','e','l','l','o']
>>> print(my_list[0])
h
>>> print(my_list[4])
o
>>> print(my_list[5])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> print(my_list[2.3])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list indices must be integers or slices, not float
>>> print(my_list[-1])#negative indexing
o
>>> print(my_list[-3])
l
>>> print(my_list[0:3])#slicing
['h', 'e', 'l']
>>> print(my_list[2:])
['l', 'l', 'o']
```

```
>>> print(my_list[:-2])
['h', 'e', 'l']
>>> my_list = ["Happy",[2,0,2,1]]#nested list
>>> print(my_list[0][1])
a
>>> print(my_list[1][2])
2
>>> my_list = [11,2,3,4]
>>> my_list[0] = 1#updating
>>> print(my_list)
[1, 2, 3, 4]
>>> my_list.append(5)#inserting one element
>>> print(my_list)
[1, 2, 3, 4, 5]
>>> my_list.extend([6,7,8])#inserting multiple elements
>>> print(my_list)
[1, 2, 3, 4, 5, 6, 7, 8]
>>> my_list = [1,2,3]
>>> my_list.insert(1,4) #inserting at a position
>>> print(my_list)
[1, 4, 2, 3]
>>> print(my_list + [5,6,7])#concatenation
[1, 4, 2, 3, 5, 6, 7]
>>> print(["re"]*3)#Repeating a list
['re', 're', 're']
>>> my_list = ['p','r','o','b','l','e','m']
>>> del my_list[2]#deleting one element
>>> print(my_list)
['p', 'r', 'b', 'l', 'e', 'm']
>>> my_list.remove('p') #deleting a given element
>>> print(my_list)
['r', 'b', 'l', 'e', 'm']
>>> print(my_list.pop())#deleting the last element
m
>>> print(my_list)
['r', 'b', 'l', 'e']
>>> my_list.clear() #deleting all elements
>>> print(my_list)
[]
>>> pow2 = [2**x for x in range(5)] #list comprehension
>>> print(pow2)
[1, 2, 4, 8, 16]
```

Operations on Tuples:

```
>>> tup1 = (1, 2, 3, 4, 5, 6, 7)
>>> print(tup1[0])
1
>>> print(tup1[1:5]) #slicing
(2, 3, 4, 5)
>>> print(tup1[2:])
(3, 4, 5, 6, 7)
>>> print(tup1[:-3])
(1, 2, 3, 4)
>>> print(tup1[::-1])
(7, 6, 5, 4, 3, 2, 1)
>>> print(tup1[-6]) #negative indexing
2
>>> tup1 = (4, 2, 3, [6, 5])
>>> tup1[3][0] = 9 #updating a mutable element
>>> print(tup1)
(4, 2, 3, [9, 5])
>>> del tup1 #deleting whole tuple
>>> my_tuple = ('a','p','p','l','e')
>>> print('a' in my_tuple) #membership test
True
>>> print('b' in my_tuple)
False
>>> print('g' not in my_tuple)
True
>>> print(type(my_tuple))
<class 'tuple'>
>>> print(len(my_tuple)) #length of tuple
5
>>> print(my_tuple.count('p')) #count occurrance of an element
2
>>> print(my_tuple.index('l')) #index of an element
3
>>> tup1 = (3,5,2,4,1)
>>> print(sorted(tup1)) #sorting
[1, 2, 3, 4, 5]
>>> tuple1 = (0, 1, 2, 3)
>>> tuple2 = ('python', 'geek')
>>> print(tuple1 + tuple2) #tuple concatenation
(0, 1, 2, 3, 'python', 'geek')
>>> tuple3 = (tuple1, tuple2) #nested tuples
>>> print(sum(tuple1))
6
>>> print(tuple3)
((0, 1, 2, 3), ('python', 'geek'))
>>> my_tuple = ('test',)*3 #repetition
>>> print(my_tuple)
('test', 'test', 'test')
>>> list1 = [1,2,3,4]
>>> print(tuple(list1)) #list to tuple
(1, 2, 3, 4)
```

```
>>> print(tuple('python')) #string to tuple
('p', 'y', 't', 'h', 'o', 'n')
>>> print(max(my_tuple)) #largest element
test
>>> print(min(my_tuple)) #smallest element
test
>>> tuple1 = ('python', 'test')
>>> tuple2 = ('coder', 1)
>>> tuple3 = ('coder', 1)
>>> print(tuple1==tuple2) #comparing tuples
False
>>> print(tuple2==tuple3)
True
>>> print((1,2,3)>(4,5,6))
False
>>> print((1,2)==('1','2'))
False
```

Operations on Dictionary:

```
>>> my_dict = {} #empty dictionary
>>> print(my_dict)
{}
>>> my_dict = {'Name': 'Test', 1: [1, 2, 3, 4]} #mixed keys
>>> print(my_dict)
{'Name': 'Test', 1: [1, 2, 3, 4]}
>>> my_dict = dict([(1,'Hello'),(2,'World')]) #dictionary from list of tuples
>>> print(my_dict)
{1: 'Hello', 2: 'World'}
>>> my_dict = {1: 'Hello', 2: 'World', 3:{'A':'Welcome', 'B':'To', 'C': 'Python'}}
>>> print(my_dict)
{1: 'Hello', 2: 'World', 3: {'A': 'Welcome', 'B': 'To', 'C': 'Python'}}
>>> my_dict = {'name':'Jack', 'age': 26}
>>> print(my_dict.keys())
dict_keys(['name', 'age'])
>>> print(my_dict.values())
dict_values(['Jack', 26])
```

```
>>> print(my_dict['name']) #accessing using key
Jack
>>> print(my_dict.get('age')) #accessing using get() method
26
>>> my_dict['age'] = 27 #updating
>>> print(my_dict)
{'name': 'Jack', 'age': 27}
>>> my_dict['address'] = 'Downtown' #inserting
>>> print(my_dict)
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
>>> del my_dict['name']
>>> print(my_dict)
{'age': 27, 'address': 'Downtown'}
>>> my_dict.pop('age')
27
>>> print(my_dict)
{'address': 'Downtown'}
>>> my_dict.clear()
>>> print(my_dict)
{}
>>> del my_dict
>>> marks = {}.fromkeys(['Math','English','Science'],0)
>>> print(marks)
{'Math': 0, 'English': 0, 'Science': 0}
>>> print(list(sorted(marks.keys())))
['English', 'Math', 'Science']
>>> squares = {x: x*x for x in range(6)} #dictionary comprehension
>>> print(squares)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> print(1 in squares) #membership test
True
>>> print(6 not in squares)
True
>>> print(len(squares)) #length of dictionary
6
>>> print(sorted(squares)) #sorted list of keys
[0, 1, 2, 3, 4, 5]
>>> print(type(squares))
<class 'dict'>
>>> print(all(squares))
False
>>> print(any(squares))
True
>>> new_list = squares.copy()
>>> print(new_list)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Operations and Built-In Function of Sets:

```
>>> my_set = {1,2,2,3}
>>> my_set.add(4) #insert
>>> print(my_set)
{1, 2, 3, 4}
>>> my_set.update([5,6]) #update using list
>>> print(my_set)
{1, 2, 3, 4, 5, 6}
>>> my_set.update([6,7], {1,6,8}) #update using set
>>> print(my_set)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> print(7 in my_set) #membership test
True
>>> print(5 not in my_set)
False
>>> print(len(my_set))
8
>>> print(type(my_set))
<class 'set'>
>>> new_set = my_set.copy()
>>> print(new_set)
{1, 2, 3, 4, 5, 6, 7, 8}
>>> print(max(my_set))
8
>>> print(min(my_set))
1
>>> my_set.discard(4)
>>> print(my_set)
{1, 2, 3, 5, 6, 7, 8}
>>> my_set.remove(6)
>>> print(my_set)
{1, 2, 3, 5, 7, 8}
>>> print(my_set.pop()) #deleting a random element
1
>>> print(my_set)
{2, 3, 5, 7, 8}
>>> my_set.clear()
>>> print(my_set)
set()
```

```
>>> A = {1, 2, 3, 4, 5}
>>> B = {4, 5, 6, 7, 8}
>>> C = {6, 7, 8}
>>> print(A | B) #set union
{1, 2, 3, 4, 5, 6, 7, 8}
>>> print(A.union(B))
{1, 2, 3, 4, 5, 6, 7, 8}
>>> print(A & B) #set intersection
{4, 5}
>>> print(A.intersection(B))
{4, 5}
>>> print(A - B) #set difference
{1, 2, 3}
>>> print(B.difference(A))
{8, 6, 7}
>>> print(A ^ B) #set symmetric difference
{1, 2, 3, 6, 7, 8}
>>> print(B.symmetric_difference(A))
{1, 2, 3, 6, 7, 8}
>>> print(A.isdisjoint(B))
False
>>> print(A.isdisjoint(C))
True
>>> print(C.issubset(B))
True
>>> print(C.issubset(A))
False
```

Built-In Function of Strings:

```
>>> myString = "GATTACA"
>>> myString[0]
'G'
>>> myString[1]
'A'
>>> myString[-1] #negative indexing
'A'
>>> myString[-2]
'C'
>>> myString[1:3] #accessing substrings
'AT'
>>> myString[:3]
'GAT'
>>> myString[4:]
'ACA'
>>> len(myString) #length of string
7
```

```
>>> "GAT" in myString #substring test
True
>>> "AGT" in myString
False
>>> "Hello" + "World" #concatenation
'HelloWorld'
>>> "A" * 10 #repeat
'AAAAAAAAAA'
>>> #String methods
>>> myString.find("ATT")
1
>>> myString.count("T")
2
>>> myString.lower()
'gattaca'
>>> myString.upper()
'GATTACA'
>>> myString.replace("G","U")
'UATTACA'
>>> myString.replace("AT","**")
'G**TACA'
>>> myString = "GATTACA"
>>> myString.startswith("G")
True
>>> myString.startswith("g")
False
>>> myString.endswith("A")
True
>>> str = "this is string example!"
>>> print(str.capitalize())
This is string example!
>>> print(str.count("i", 4, 20))
2
>>> print(str.count("is"))
2
>>> print(str.index("exam"))
15
```

```
>>> print(str.split())
['this', 'is', 'string', 'example!']
>>> str="this123"
>>> print(str.isalnum())
True
>>> print(str.isdigit())
False
>>> print(str.isalpha())
False
>>> print(str.isidentifier())
True
>>> print(str.islower())
True
>>> print(str.isupper())
False
>>> print(str.isnumeric())
False
>>> str="     example123   "
>>> print(str.lstrip())
example123
>>> str = "Hello, And Welcome To My World!"
>>> x = str.casefold()
>>> print(x)
hello, and welcome to my world!
>>> str = "banana"
>>> x = str.center(20)
>>> print(x)
       banana
>>> txt = "Hello My Name Is PETER"
>>> x = txt.swapcase()
>>> print(x)
hELLO mY nAME iS peter
>>> str = "H\te\tl\tl\to"
>>> x =  str.expandtabs(2)
>>> print(x)
H e l l o
>>> str = "For only {price:.2f} dollars!"
>>> print(str.format(price = 49))
For only 49.00 dollars!
>>> txt = "Hello, welcome to my world."
>>> x = txt.index("welcome")
>>> print(x)
7
>>> txt = "Hello! Are you #1?"
>>> x = txt.isprintable()
>>> print(x)
True
>>> myTuple = ("John", "Peter", "Vicky")
>>> x = "#".join(myTuple)
>>> print(x)
John#Peter#Vicky
>>> txt = "banana"
>>> x = txt.ljust(20)
>>> print(x, "is my favorite fruit.")
banana               is my favorite fruit.
```

```
>>> txt = "banana"
>>> x = txt.rjust(20)
>>> print(x, "is my favorite fruit.")
             banana is my favorite fruit.
>>> txt = "Mi casa, su casa."
>>> x = txt.rindex("casa")
>>> print(x)
12
>>> txt = "Hello Sam!"
>>> mytable = txt.maketrans("S", "P")
>>> print(txt.translate(mytable))
Hello Pam!
>>> txt = "I could eat bananas all day"
>>> x = txt.partition("bananas")
>>> print(x)
('I could eat ', 'bananas', ' all day')
>>> txt = "50"
>>> x = txt.zfill(10)
>>> print(x)
0000000050
>>> mydict = {83:  80}
>>> txt = "Hello Sam!"
>>> print(txt.translate(mydict))
Hello Pam!
>>> txt = "Welcome to my world"
>>> x = txt.title()
>>> print(x)
Welcome To My World
```

Ranges:

```
>>> print(type(range(3)))
<class 'range'>
>>> print(list(range(10)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(list(range(1, 10)))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print(list(reversed(range(1, 10))))
[9, 8, 7, 6, 5, 4, 3, 2, 1]
>>> print(list(range(2, 14, 2)))
[2, 4, 6, 8, 10, 12]
>>> print(list(range(25, -6, -3)))
[25, 22, 19, 16, 13, 10, 7, 4, 1, -2, -5]
```

```
>>> print(list(range(2, -14, -2)))
[2, 0, -2, -4, -6, -8, -10, -12]
>>> res = chain(range(5), range(10, 20, 2))
>>> print(list(res))
[0, 1, 2, 3, 4, 10, 12, 14, 16, 18]
>>> print(range(10)[0])
0
>>> print(range(10)[4])
4
>>> print(range(10)[-1])
9
```

Conclusion:

Thus, we have understood operations on List, Tuple and Dictionary using Built-in functions, along with built-in Set, String and Range functions.