Python

Experiment No. 2

Aim: To Study set, range, List, Tuple, Dictionary and string data types in python.

Problem Statements:

1. Implement a program to store a matrix using nested list and fetch index element of matrix
2. Create a list of 10 random players in football team. perform following operations on list and discuss on each output 1. Display list, Sorts the list, use remove to Remove the first item with the specified value
3. Write a program to randomly select 10 integer elements from range 100 to 200 and find the minimum and maximum among all.
4. Create a dictionary of 5 countries with their currency details and display them

Theory:

Python Number:

Number data types store numeric values. Number objects are created when you assign a value to them.

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

1. Set

A Set in Python programming is an unordered collection data type that is iterable, mutable and has no duplicate elements.

Set are represented by { } (values enclosed in curly braces)

The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a hash table. Since sets are unordered, we cannot access items using indexes as we do in lists.

Example of Python Sets

```
var = {"Geeks", "for", "Geeks"}

type(var)
```

Python sets can store **heterogeneous elements** in it, i.e., a set can store a mixture of string, integer, boolean, etc datatypes.

Frozen set

While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

```
# A frozen set

frozen_set = frozenset(["e", "f", "g"])

print("\nFrozen Set")

print(frozen_set)

 # Uncommenting below line would cause error as

# we are trying to add element to a frozen set

# frozen_set.add("h")
```

Methods for Sets

Adding elements to Python Sets

Insertion in the set is done through the set.add() function, where an appropriate record value is created to store in the hash table. Set1.add(2);

Union operation on Python Sets

Two sets can be merged using union() function or | operator. Both Hash Table values are accessed and traversed with merge operation perform on them to combine the elements, at the same time duplicates are removed.

Set1.union(set2) or set1|set2

Intersection operation on Python Sets

This can be done through intersection() or & operator. Common Elements are selected. They are similar to iteration over the Hash lists and combining the same values on both the Table.

Set1.intersection(set2) or set1&set2

Finding Differences of Sets in Python

To find differences between sets. Similar to finding differences in the linked list.

set1.difference(set2) or set1-set2

Set Clear() method empties the whole set inplace.

Append Multiple Elements in Set in Python

update() Method

In this method, we will use Python's in-built set update() function to get all the elements in the list aligned with the existing set.

Using | Operator (Pipe Operator)

The pipe operator internally calls the union() function, which can be used to perform the task of updating the Python set with new elements.

here are two major pitfalls in Python sets:

- The set doesn't maintain elements in any particular order.
- Only instances of immutable types can be added to a Python set.

Range

The Python range() function is used to generate a sequence of numbers. Depending on how many arguments the user is passing to the function, the user can decide where that series of numbers will begin and end as well as how big the difference will be between one number and the next.range() takes mainly three arguments.

start: integer starting from which the sequence of integers is to be returned

stop: integer before which the sequence of integers is to be returned. The range of integers ends at a stop – 1.

step: integer value which determines the increment between each integer in the sequence.

2. Python List:

A list is a collection of things, enclosed in [ ] and separated by commas.

The list is a sequence data type which is used to store the collection of data. Tuples and String are other types of sequence data types.

Example:

```
var = ["Geeks", "for", "Geeks"]

print(Var)
```

Lists are the simplest containers that are an integral part of the Python language. Lists need not be homogeneous always which makes it the most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

Creating a List in Python

Lists in Python can be created by just placing the sequence inside the square brackets[]. Unlike Sets, a list doesn't need a built-in function for its creation of a list.

Note: Unlike Sets, the list may contain mutable elements.

Example:

Code:

```
# Python program to demonstrate
# Creation of List

# Creating a List
List = []
print("Blank List: ")
print(List)

# Creating a List of numbers
List = [10,20,14]
print("\nList of numbers: ")
print(List)

# Creating a List of strings and accessing
# using index
List = ["Geeks","For","Geeks"]
print("\nList Items: ")
print(List[0])
```

```
print(List[2])
```

Output:

```
Blank List:
[]

List of numbers:
[10, 20, 14]

List Items:
Geeks
Geeks
```

Creating a list with multiple distinct or duplicate elements

A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

Code:

```
# Creating a List with the use of Numbers (Having duplicate
values)
List = [1,2,4,4,3,3,3,6,5]
print("\nList with the use of Numbers: ")
print(List)
 # Creating a List with  mixed type of values
# (Having numbers and strings)
List = [1,2,'Name',4,'Chaitanya',6,'Shah']
print("\nList with the use of Mixed Values: ")
print(List)
```

Output:

```
List with the use of Numbers:
[1, 2, 4, 4, 3, 3, 3, 6, 5]

List with the use of Mixed Values:
[1, 2, 'Name', 4, 'Chaitanya', 6, 'Shah']
```

Accessing elements from the List

In order to access the list items refer to the index number. Use the index operator [ ] to access an item in a list. The index must be an integer. Nested lists are accessed using nested indexing.

Example 1:

Code:

```
# Python program to demonstrate # accessing of element from list
# Creating a List with the use of multiple values
List = ["Hello","World"]
# accessing a element from the list using index number
print("Accessing a element from the list")
print(List[0])
print(List[1])
```

Output:

```
Accessing a element from the list
Hello
World
```

Example 2: Accessing elements from a multi-dimensional list

Code:

```
# Creating a Multi-Dimensional List # (By Nesting a list inside a
List)
List = [['Hello','World'],['Heyyy']]
# accessing an element from the # Multi-Dimensional List using #
index number
print("Accessing a element from a Multi-Dimensional list")
print(List[0][1])
print(List[1][0])
```

Output:

```
Accessing a element from a Multi-Dimensional list
World
Heyyy
```

Negative indexing

In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

Code:

```
List = [1,2,'Geeks',4,'For',6,'Geeks']
 # accessing an element using negative indexing
print("Accessing element using negative indexing")
 # print the last element of list
print(List[-1])
 # print the third last element of list
print(List[-3])
```

Output:

```
Accessing element using negative indexing
Geeks
For
```

Getting the size of Python list

Python len() is used to get the length of the list.

Code:

```
# Creating a List
List1 = []
print(len(List1))
 # Creating a List of numbers
List2 = [10,20,14]
print(len(List2))
```

Output:

```
0
3
```

Taking Input of a Python List

We can take the input of a list of elements as string, integer, float, etc. But the default one is a string.

```
string = input("Enter elements (Space-Separated): ")

 # split the strings and store it to a list

lst = string.split()

print('The list is:', lst)   # printing the list
```

Adding Elements to a Python List

Method 1: Using append() method

Elements can be added to the List by using the built-in append() function. Only one element at a time can be added to the list by using the append() method, for the addition of multiple elements with the append() method, loops are used. Tuples can also be added to the list with the use of the append method because tuples are immutable. Unlike Sets, Lists can also be added to the existing list with the use of the append() method.

 # Python program to demonstrate

 # Addition of elements in a List

Code:

```
# Creating a List
List = []
print("Initial blank List: ")
print(List)

# Addition of Elements in the List
List.append(1)
List.append(2)
List.append(4)
print("\nList after Addition of Three elements: ")
print(List)

# Adding elements to the List using Iterator
foriinrange(1,4):
  List.append(i)
print("\nList after Addition of elements from 1-3: ")
print(List)
```

```
# Adding Tuples to the List
List.append((5,6))
print("\nList after Addition of a Tuple: ")
print(List)

# Addition of List to a List
List2 = ['For','Geeks']
List.append(List2)
print("\nList after Addition of a List: ")
print(List)
```

Output:

```
Initial blank List:
[]

List after Addition of Three elements:
[1, 2, 4]

List after Addition of elements from 1-3:
[1, 2, 4, 1, 2, 3]

List after Addition of a Tuple:
[1, 2, 4, 1, 2, 3, (5, 6)]

List after Addition of a List:
[1, 2, 4, 1, 2, 3, (5, 6), ['For', 'Geeks']]
```

Method 2: Using insert() method

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used. Unlike append() which takes only one argument, the insert() method requires two arguments(position, value).

Code:

```
List = [1,2,3,4]
print("Initial List: ")
print(List)

# Addition of Element at specific Position (using Insert Method)
List.insert(3,12)
List.insert(0,'Geeks')
print("\nList after performing Insert Operation: ")
print(List)
```

Output:

```
Initial List:
[1, 2, 3, 4]

List after performing Insert Operation:
['Geeks', 1, 2, 3, 12, 4]
```

Method 3: Using extend() method

Other than append() and insert() methods, there's one more method for the Addition of elements, extend(), this method is used to add multiple elements at the same time at the end of the list.

Code:

```
List = [1,2,3,4]
print("Initial List: ")
print(List)
# Addition of multiple elements to the List at the end (using
Extend Method)
List.extend([8,'Geeks','Always'])
print("\nList after performing Extend Operation: ")
print(List)
```

Output:

```
Initial List:
[1, 2, 3, 4]

List after performing Extend Operation:
[1, 2, 3, 4, 8, 'Geeks', 'Always']
```

Reversing a List

Method 1: A list can be reversed by using the reverse() method in Python.

```
mylist = [1, 2, 3, 4, 5, 'Geek', 'Python']

mylist.reverse()

print(mylist)
```

Method 2: Using the reversed() function:

The reversed() function returns a reverse iterator, which can be converted to a list using the list() function.

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

reversed_list = list(reversed(my_list))

print(reversed_list)
```

Removing Elements from the List

Method 1: Using remove() method

Elements can be removed from the List by using the built-in remove() function but an Error arises if the element doesn't exist in the list. Remove() method only removes one element at a time, to remove a range of elements, the iterator is used. The remove() method removes the specified item.

Note: Remove method in List will only remove the first occurrence of the searched element.

```
List = [1, 2, 3, 4, 5, 6,

      7, 8, 9, 10, 11, 12]

print("Initial List: ")

print(List)

 # Removing elements from List using Remove() method

List.remove(5)

List.remove(6)

print("\nList after Removal of two elements: ")

print(List)
```

Method 2: Using pop() method

pop() function can also be used to remove and return an element from the list, but by default it removes only the last element of the list, to remove an element from a specific position of the List, the index of the element is passed as an argument to the pop() method.

```
List = [1, 2, 3, 4, 5]

 # Removing element from the Set using the pop() method

List.pop()

print("\nList after popping an element: ")

print(List)

 # Removing element at a specific location from the Set using the pop() method

List.pop(2)

print("\nList after popping a specific element: ")

print(List)
```

Slicing of a List

We can get substrings and sublists using a slice. In Python List, there are multiple ways to print the whole list with all the elements, but to print a specific range of elements from the list, we use the Slice operation.

Slice operation is performed on Lists with the use of a colon(:).

To print elements from beginning to a range use:
[: Index]

To print elements from end-use:
[:-Index]

To print elements from a specific Index till the end use
[Index:]

To print the whole list in reverse order, use
[::-1]

Note – To print elements of List from rear-end, use Negative Indexes.

List Comprehension

Python List comprehensions are used for creating new lists from other iterables like tuples, strings, arrays, lists, etc. A list comprehension consists of brackets containing the expression, which is executed for each element along with the for loop to iterate over each element.

Syntax:

```
newList = [ expression(element) for element in oldList if condition ]

odd_square = [x ** 2 for x in range(1, 11) if x % 2 == 1]

print(odd_square)
```

3. Python Tuple:

Tuple is a collection of Python objects much like a list. The sequence of values stored in a tuple can be of any type, and they are indexed by integers.

Values of a tuple are syntactically separated by 'commas'. Although it is not necessary, it is more common to define a tuple by closing the sequence of values in parentheses. This helps in understanding the Python tuples more easily.

Creating a Tuple

In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping the data sequence.

Note: Creation of Python tuple without the use of parentheses is known as Tuple Packing.

```
Tuple1 = ()

print("Initial empty Tuple: ")

print(Tuple1)
```

Code:

```
# Creating a Tuple
# with the use of string
Tuple1 = ('Geeks','For')
print("\nTuple with the use of String: ")
print(Tuple1)

# Creating a Tuple with
# the use of list
list1 = [1,2,4,5,6]
print("\nTuple using List: ")
print(tuple(list1))

# Creating a Tuple
# with the use of built-in function
Tuple1 = tuple('Geeks')
```

```
print("\nTuple with the use of function: ")
print(Tuple1)
```

Output:

```
Tuple with the use of String:
('Geeks', 'For')

Tuple using List:
(1, 2, 4, 5, 6)

Tuple with the use of function:
('G', 'e', 'e', 'k', 's')
```

Creating a Tuple with Mixed Datatypes.

Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.). Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing 'comma' to make it a tuple.

Code:

```
# Creating a Tuple# with Mixed Datatype
Tuple1 = (5,'Welcome',7,'Geeks')
print("\nTuple with Mixed Datatypes: ")
print(Tuple1)
# Creating a Tuple# with nested tuples
Tuple1 = (0,1,2,3)
Tuple2 = ('python','geek')
Tuple3 = (Tuple1, Tuple2)
print("\nTuple with nested tuples: ")
print(Tuple3)
# Creating a Tuple# with repetition
Tuple1 = ('Geeks',) * 3
print("\nTuple with repetition: ")
print(Tuple1)
```

Output:

```
Tuple with Mixed Datatypes:
(5, 'Welcome', 7, 'Geeks')

Tuple with nested tuples:
((0, 1, 2, 3), ('python', 'geek'))

Tuple with repetition:
('Geeks', 'Geeks', 'Geeks')
```

Accessing of Tuples

Tuples are immutable, and usually, they contain a sequence of heterogeneous elements that are accessed via unpacking or indexing (or even by attribute in the case of named tuples). Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

Note: In unpacking of tuple number of variables on the left-hand side should be equal to a number of values in given tuple a.

Code:

```
# Accessing Tuple
Tuple1 = tuple("Geeks")
print("\nFirst element of Tuple: ")
print(Tuple1[0])

# Tuple unpacking
Tuple1 = ("Geeks","For","Geeks")

# This line unpack
# values of Tuple1
a, b, c = Tuple1
print("\nValues after unpacking: ")
print(a)
print(b)
print(c)
```

Output:

```
First element of Tuple:
G

Values after unpacking:
Geeks
For
Geeks
```

Concatenation of Tuples

Concatenation of tuple is the process of joining two or more Tuples. Concatenation is done by the use of '+' operator. Concatenation of tuples is done always from the end of the original tuple. Other arithmetic operations do not apply on Tuples.

```
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('Geeks', 'For', 'Geeks')
```

```
Tuple3 = Tuple1 + Tuple2
```

Deleting a Tuple

Tuples are immutable and hence they do not allow deletion of a part of it. The entire tuple gets deleted by the use of del() method.

Note- Printing of Tuple after deletion results in an Error.

```
Tuple1 = (0, 1, 2, 3, 4)
del Tuple1
print(Tuple1)
```

Built-In Methods

Built-in-Method	Description
index( )	Find in the tuple and returns the index of the given value where it's available
count( )	Returns the frequency of occurrence of a specified value

Tuples VS Lists:

Similarities	Differences
Functions that can be used for both lists and tuples: len(), max(), min(), sum(), any(), all(), sorted()	Methods that cannot be used for tuples: append(), insert(), remove(), pop(), clear(), sort(), reverse()
Methods that can be used for both lists and tuples: count(), Index()	we generally use 'tuples' for heterogeneous (different) data types and 'lists' for homogeneous (similar) data types.
Tuples can be stored in lists.	Iterating through a 'tuple' is faster than in a 'list'.

Lists can be stored in tuples.	‘Lists’ are mutable whereas ‘tuples’ are immutable.
Both ‘tuples’ and ‘lists’ can be nested.	Tuples that contain immutable elements can be used as a key for a dictionary.

4. Python Dictionary:

Dictionary in Python is a collection of keys values, used to store data values like a map, which, unlike other data types which hold only a single value as an element.

Example of Dictionary in Python

Dictionary holds key:value pair. Key-Value is provided in the dictionary to make it more optimized.

```
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}

print(Dict)
```

Creating a Dictionary

In Python, a dictionary can be created by placing a sequence of elements within curly {} braces, separated by ‘comma’. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its Key:value. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be immutable.

```
# Creating a Dictionary with Integer Keys

Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}

print("\nDictionary with the use of Integer Keys: ")

print(Dict)

  # Creating a Dictionary with Mixed keys

Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}

print("\nDictionary with the use of Mixed Keys: ")

print(Dict)
```

Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing to curly braces{}.

```
# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")
```

```
print(Dict)
  # Creating a Dictionary# with dict() method
Dict = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})
print("\nDictionary with the use of dict(): ")
print(Dict)
```

Nested Dictionary

```
# Creating a Dictionary# with each item as a Pair
Dict = dict([(1, 'Geeks'), (2, 'For')])
print("\nDictionary with each item as a pair: ")
print(Dict)
# Creating a Nested Dictionary# as shown in the below image
Dict = {1: 'Geeks', 2: 'For',
      3: {'A': 'Welcome', 'B': 'To', 'C': 'Geeks'}}
print(Dict)
```

Adding elements to a Dictionary

Addition of elements can be done in multiple ways. One value at a time can be added to a Dictionary by defining value along with the key e.g. Dict[Key] = 'Value'. Updating an existing value in a Dictionary can be done by using the built-in update() method. Nested key values can also be added to an existing Dictionary.

Note- While adding a value, if the key-value already exists, the value gets updated otherwise a new Key with the value is added to the Dictionary.

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Adding elements one at a time
Dict[0] = 'Geeks'
Dict[2] = 'For'
Dict[3] = 1
```

```
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values
# to a single Key
Dict['Value_set'] = 2, 3, 4
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[2] = 'Welcome'
print("\nUpdated key value: ")
print(Dict)

# Adding Nested Key value to Dictionary
Dict[5] = {'Nested': {'1': 'Life', '2': 'Geeks'}}
print("\nAdding a Nested Key: ")
print(Dict)
```

Accessing elements of a Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets.

```
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# accessing a element using key
print("Accessing a element using key:")
print(Dict['name'])

# accessing a element using key
print("Accessing a element using key:")
print(Dict[1])
```

Accessing an element of a nested dictionary

In order to access the value of any key in the nested dictionary, use indexing [] syntax.

```
# Creating a Dictionary
Dict = {'Dict1': {1: 'Geeks'},
      'Dict2': {'Name': 'For'}}
 # Accessing element using key
print(Dict['Dict1'])
print(Dict['Dict1'][1])
print(Dict['Dict2']['Name'])
```

Deleting Elements using del Keyword

The items of the dictionary can be deleted by using the del keyword as given below.

```
Dict = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

print("Dictionary =")
print(Dict)
#Deleting some of the Dictionar data
del(Dict[1])
print("Data after deletion Dictionary=")
print(Dict)
```

.

Q.1 Implement a program to store a matrix using nested list and fetch index element of matrix

Code:

```
matrix = [[1,2,3],[4,5,6],[7,8,9]]

deffetch_element(matrix, row, col):
    if row <len(matrix)and col <len(matrix[0]):
        return matrix[row][col]
    else:
        return"Index out of bounds"

print("Matrix:")
for row in matrix:
    print(row)

print("\nFetching elements:")
print("Element at (0, 0):",fetch_element(matrix,0,0))
print("Element at (1, 1):",fetch_element(matrix,1,1))
print("Element at (2, 2):",fetch_element(matrix,2,2))
print("Element at (3, 3):",fetch_element(matrix,3,3))
```

Output:

```
Matrix:
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]

Fetching elements:
Element at (0, 0): 1
Element at (1, 1): 5
Element at (2, 2): 9
Element at (3, 3): Index out of bounds
```

Q.2 Create a list of 10 random players in football team. Perform following operations on list and discuss on each output. Display list, Sorts the list, use remove to Remove the first item with the specified value

Code:

```
players =
['player1','player9','player3','player10','player5','player6','pl
ayer7','player8','player2','player4']

print("Players List : ")
print(players)

players.sort()
print("\nSorted List : ")
print(players)

players.remove('player2')
print("\nList after removing player2 : ")
print(players)
```

Output:

```
Players List :
['player1', 'player9', 'player3', 'player10', 'player5', 'player6', 'player7', 'player8', 'player2', 'player4']

Sorted List :
['player1', 'player10', 'player2', 'player3', 'player4', 'player5', 'player6', 'player7', 'player8', 'player9']

List after removing player2 :
['player1', 'player10', 'player3', 'player4', 'player5', 'player6', 'player7', 'player8', 'player9']
```

Q.3 Write a program to randomly select 10 integer elements from range 100 to 200 and find the minimum and maximum among all.

Code:

```
import random

numbers = list(random.sample(range(100,201),10))

print("Randomly selected numbers : ")
print(numbers)

print("\nMinimum number : ",min(numbers))
print("Maximum number : ",max(numbers))
```

Output:

```
Randomly selected numbers :
[140, 198, 127, 106, 159, 115, 108, 177, 110, 173]

Minimum number :  106
Maximum number :  198
```

Q.4 Create a dictionary of 5 countries with their currency details and display them

Code:

```
countries = {
    "USA":{"currency":"US Dollar","code":"USD"},
    "India":{"currency":"Indian Rupee","code":"INR"},
    "Japan":{"currency":"Japanese Yen","code":"JPY"},
    "UK":{"currency":"Pound Sterling","code":"GBP"},
    "Brazil":{"currency":"Real","code":"BRL"}
}

print("Countries and their Currency Details:")
for country, details incountries.items():
    print(country+":")
    print("\tCurrency: ",details['currency'])
    print("\tCode: ",details['code'])
```

Output:

```
Countries and their Currency Details:
USA:
        Currency:  US Dollar
        Code:  USD
India:
        Currency:  Indian Rupee
        Code:  INR
Japan:
        Currency:  Japanese Yen
        Code:  JPY
UK:
        Currency:  Pound Sterling
        Code:  GBP
Brazil:
        Currency:  Real
        Code:  BRL
```

Lab Outcome: Thus, studied set, range, List, Tuple, Dictionary and string data types in python.