Chapter 3.

Data Structures in Python 14 Marks

Lists:

- Python offers a range of compound datatypes often referred to as sequences. List is one of the most frequently used and very versatile datatype used in Python.
- The list is a fundamentally adaptable information sort accessible in python which can be composed as a ***list of comma isolated values between square brackets***.
- Things in a list require ***not be of a similar types.***
- List in Python are ordered and have a definite count. The elements in a list are indexed according to a definite sequence and the indexing of a list is done with 0 being the first index.
- Each element in the list has its definite place in the list, which allows duplicating of elements in the list, with each element having its own distinct place and credibility.

Defining Lists:

- 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.).

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]
```

Also, a list can even have another list as an item. This is called nested list.

```
# nested list

my_list = ["mouse", [8, 4, 6], ['a']]
```

Accessing Values in List-

- ***To get values in list, utilize the square brackets with the index or the list to acquire esteem accessible at that index***.

Example:

List1=[‘physics’, ‘chemistry’, 1997, 2000]

List2=[1,2,3,4,5]

print(“List1[0] :”, List1[0])

print (“List2[1:5] :”, List2[1:5])

Output of the above code is

List1[0] : physics

List2[1:5] : [2,3,4,5]

Example:

```
my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Error! Only integer can be used for indexing
# my_list[4.0]

# Nested List
n_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: a
print(n_list[0][1])

# Output: 5
print(n_list[1][3])
```

Example of Negative indexing

```
my_list = ['p','r','o','b','e']

# Output: e
print(my_list[-1])

# Output: p
print(my_list[-5])
```

Update or add elements to a list:

List are mutable, meaning, their elements can be changed unlike string or tuple.

We can use assignment operator (=) to change an item or a range of items.

```
# mistake values
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

# Output: [1, 4, 6, 8]
print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]
print(odd)
```

We can add one item to a list using `append()` method or add several items using `extend()` method.

```
odd = [1, 3, 5]

odd.append(7)

# Output: [1, 3, 5, 7]
print(odd)

odd.extend([9, 11, 13])

# Output: [1, 3, 5, 7, 9, 11, 13]
print(odd)
```

We can also use + operator to combine two lists. This is also called concatenation.

The * operator repeats a list for the given number of times.

```
odd = [1, 3, 5]

# Output: [1, 3, 5, 9, 7, 5]
print(odd + [9, 7, 5])

#Output: ["re", "re", "re"]
print(["re"] * 3)
```

Furthermore, we can insert one item at a desired location by using the method `insert()` or insert multiple items by squeezing it into an empty slice of a list.

```
odd = [1, 9]
odd.insert(1,3)

# Output: [1, 3, 9]
print(odd)

odd[2:2] = [5, 7]

# Output: [1, 3, 5, 7, 9]
print(odd)
```

Deleting values in list:

We can delete one or more items from a list using the keyword `del`. It can even delete the list entirely.

```
my_list = ['p','r','o','b','l','e','m']

# delete one item
delmy_list[2]

# Output: ['p', 'r', 'b', 'l', 'e', 'm']
print(my_list)

# delete multiple items
delmy_list[1:5]

# Output: ['p', 'm']
print(my_list)

# delete entire list
delmy_list

# Error: List not defined
print(my_list)
```

- We can use `remove()` method to remove the given item or `pop()` method to remove an item at the given index.
- The `pop()` method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).
- We can also use the `clear()` method to empty a list.

Example:

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```

Basic List Operations-

- List react to + and * operators much like strings.

Python Expression	**Results**	**Description**
len([1,2,3])	3	Length
[1,2,3]+[4,5,6]	[1,2,3,4,5,6]	Concatenation
[‘Hi!’]*4	[‘Hi!’,‘Hi!’,‘Hi!’,‘Hi!’]	Repetition
3 in [1,2,3]	True	Membership
for x in [1,2,3]:print x,	1 2 3	Iteration

Indexing, Slicing and Matrixes-

Assume the following input

L=[‘spam’, ‘Spam’, SPAM!’]

Python Expression	Results	Description
L[2]	‘SPAM!’	Offset starts at zero
L[-2]	‘Spam’	Negative:count from right
L[1:]	[‘Spam’, ‘SPAM!’]	Slicing fetches section

Built-in List Functions and Methods-

- Python includes the following list functions.

Function	Description
cmp(list1,list2)	Compares elements of both lists
len(list)	Gives the total length of the list
max(list)	returns item from the list with max value
min(list)	returns item from the list with min value
list(seq)	converts a tuple into list

len(list) –

Description

The function len() returns the number of elements in the

list.

Syntax

len(list)

Parameters

list- This is a list for which number of elements to be counted.

Return value

It returns the number of elements in the list.

Example

```
list1=[123,'xyz','pqrs']
print (len(list1))
```

Output-

3

max(list) –

Description

The function max() returns the number of elements from the

list with maximum value.

Syntax

```
max(list)
```

Parameters

list- This is a list from which max valued elements to be returned.

Return value

It returns the elements from the list with maximum value.

Example

```
list1=[123,'xyz','pqrs']
List2=[456,700,200]
```

print (max(list1))

print (max(list2))

Output-

pqrs

700

min(list) –

Description

The function min() returns the number of elements from the

list with minimum value.

Syntax

min(list)

Parameters

list- This is a list from which min valued elements to be returned.

Return value

It returns the elements from the list with minimum value.

Example

list1=[123,’xyz’,’pqrs’]

List2=[456,700,200]

print (max(list1))

print (max(list2))

Output-

123

Python List Methods

append() - Add an element to the end of the list

extend() - Add all elements of a list to the another list

insert() - Insert an item at the defined index

remove() - Removes an item from the list

pop() - Removes and returns an element at the given index

clear() - Removes all items from the list

index() - Returns the index of the first matched item

count() - Returns the count of number of items passed as an argument

sort() - Sort items in a list in ascending order

reverse() - Reverse the order of items in the list

copy() - Returns a shallow copy of the list

List Methods –

list.append(obj)

Description:

This method appends a passed obj into the existing list.

Syntax:

list.append(obj)

Parameters:

Obj-Object to be appended in the list.

Return Value

It does not return value but updates existing list.

Example

```
list1=[123,'xyz','pqrs']
list1.append(2009)
print(list1)
```

Output-

[123,'xyz','pqrs',2009]

list.count(obj)

Description:

This method returns count of how many times obj occurs in list.

Syntax:

list.append(obj)

Parameters:

Obj-Object to be counted in the list.

Return Value

It returns count of how many obj occurs in list.

Example

```
list1=[123,'xyz','pqrs', 123]
print (list1.count(123))
print (list1.count('xyz'))
```

Output-

2

1

list.extend(seq)

Description:

This method appends the contents of sequence to list.

Syntax:

list.extend(seq)

Parameters:

seq-List of elements.

Return Value

It does not return any value but add the contenet to existing list.

Example

```
list1=[123,'xyz','pqrs']
list2=[2009,'abc']
list1.extend(list2)
print (list1)
```

Output-

[123,'xyz','pqrs',2009,'abc']

list.index(obj)

Description:

This method returns the lowest index in list that obj appears.

Syntax:

list.index(obj)

Parameters:

obj- Object to be find out.

Return Value

It returns index of found object otherwise raise an exception

indicating that value does not found.

Example

```
list1=[123,'xyz','pqrs']
print(list1.index('xyz'))
print(list1.index('pqrs'))
```

Output-

```
1
2
```

list.insert(index,obj)

Description:

This method inserts obj into list at offset index.

Syntax:

list.insert(index,obj)

Parameters:

obj- Object to be inserted into given list

Index- Index where obj need to be inserted.

Return Value

It does not return any value but rather it embeds the given element in the list.

Example

```
list1=[123,'xyz','pqrs‘,’abc’]

list1.insert(3,2009)

print(list1)
```

Output-

[123,’xyz’,’pqrs’,2009,’abc’]

list.pop(obj=list[-1])

Description:

This method removes and returns last object or obj from the

list.

Syntax:

```
list.pop(obj=list[-1])
```

Parameters:

obj- index of the object to be removed from the list.

Return Value

It returns the detached object from the list.

Example

```
list1=[123,'xyz','pqrs‘,’abc’]
print(list1.pop())
print(list1.pop(2))
```

Output-

abc

pqrs

list.remove(obj)

Description:

This method removes object from the list.

Syntax:

```
list.remove(obj)
```

Parameters:

obj- object to be removed from the list.

Return Value

It does not return the value but removes the given object from the list.

Example

```
list1=[123,'xyz','pqrs‘,’abc’,’xyz’]
list1.remove(‘xyz’))
print(list1)
```

Output-

[123, 'pqrs', 'abc', 'xyz']

list.reverse()

Description:

This method reverses objects of list in place.

Syntax:

list.reverse()

Return Value

It does not return the value but reverses the objects from the

list.

Example

list1=[123,'xyz','pqrs‘,’abc’,’xyz’]

list1.reverse()

print(list1)

Output-

['xyz', 'abc', 'pqrs', 'xyz', 123]

list.sort([func])

Description:

This method sort is used to sort a list in ascending,

descending or user defined order.

Syntax:

list.sort() #for ascending

list.sort(reverse=True) #for descending

list.sort(key=..,reverse=..) # user defined order

Return Value

It returns the list values in ascending, descending or user defined

order.

Example

```
list1=[444,123,456,989,]

list1.sort()

print(list1)
```

Output-

[123, 444, 456, 989]

Example

```
list1=[444,123,456,989,]

list1.sort(reverse=True)

print(list1)
```

Output-

[989, 456, 444, 123]

Tuples:

- ***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 which are mutable***.
- A tuple is a grouping of permanent objects. They are sequences much the same as lists.
- The difference between tuples and lists are that tuples can’t be changed like list.

- Tuples uses parenthesis and lists uses square brackets.
- Syntactically, they are coded in parentheses instead of square brackets, and they support arbitrary types, arbitrary nesting, and the usual sequence operations:

For example

Tup1=(‘physics’,’chemistry’,’1997,2000)

Tup2=(1,2,3,4,5)

- Python often recognizes that a ***tuple is intended even if the parentheses are missing***:

Tup3=‘a’, ‘b’, ‘c’, ‘d’

- For completeness, 0- and 1-element tuples can be defined, but have special syntax:

a = () # 0-tuple (empty tuple)

b = (item,) # 1-tuple (note the trailing comma)

c = item, # 1-tuple (note the trailing comma)

e.g tup1=(50,)

Accessing Values in Tuples-

- ***To get values in tuple, utilize the square brackets with the index or the tuple to acquire esteem accessible at that index***.

tup1=(‘physics’, ‘chemistry’, 1997, 2000)

tup2=(1,2,3,4,5)

print(“tup1[0] :”, tup1[0])

print(“tup2[1:5] :”, tup2[1:5])

Output of the above code is

tup1[0] : physics

tup2[1:5] : [2,3,4,5]

Updating Tuples-

- ***Tuples are permanent which implies you can't refresh or change the estimations of tuple components .You can take parts of existing tuples to make new tuples.***

tup1=(12, 34.56)

tup2=('abc','xyz')

following action is not valid for tuples

tup1[0]=100

to create new tuple

tup3=tup1+tup2

print(tup3)

Output of the above code is

(12, 34.56, 'abc', 'xyz')

Deleting Tuple Elements-

- ***Removing individual tuple components is inrealistic***.
- To explicitly remove an entire tuple, just use the del statement

tup1=(12, 34.56)

print(tup3)

del tup1

print(“After deleting tup1”)

print(tup)

Output:

Traceback (most recent call last):

File "F:\2019-2020\PYTH prog\tup_ex.py", line 2, in <module>

print(tup3)

NameError: name 'tup3' is not

- Although tuples ***support most of the same operations as lists*** (such as indexing, slicing, and concatenation), the ***contents of a tuple cannot be modified after creation*** (that is, you cannot replace, delete, or append new elements to an existing tuple).

Basic Tuples Operations-

- List react to + and * operators much like strings.

Python Expression	**Results**	**Description**
len((1,2,3))	3	Length
(1,2,3)+(4,5,6)	(1,2,3,4,5,6)	Concatenation
(‘Hi!’)*4	(‘Hi!’,‘Hi!’,‘Hi!’,‘Hi!’)	Repetition
3 in *(1,2,3)	True	Membership
for x in (1,2,3):print x,	1 2 3	Iteration

Indexing, Slicing and Matrixes-

Assume the following input

L=(‘spam’, ‘Spam’, SPAM!’)

Python Expression	Results	Description
L[2]	‘SPAM!’	Offset starts at zero
L[-2]	‘Spam’	Negative:count from right
L[1:]	(‘Spam’, ‘SPAM!’)	Slicing fetches section

Advantages of Tuple over List

Since tuples are quite similar to lists, both of them are used in similar situations as well.

However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:

- We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
- Since tuples are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
- Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Built-in Tuple Functions -

- Python includes the following tuple functions.

Function	Description
len(tuple)	Gives the total length of the list
max(tuple)	returns item from the tuple with max value
min(tuple)	returns item from the tuple with min value
tuple(seq)	converts a list into tuple

len(tuple) –

Description

The function len() returns the number of elements in the tuple.

Syntax

len(tuple)

Parameters

tuple- This is a tuple for which number of elements to be counted.

Return value

It returns the number of elements in the tuple.

Example

```
tup1,tup2=(123,'XYZ','PQR'),(456,'abc')
print(len(tup1))
print(len(tup2))
```

Output-

3

2

max(tuple) –

Description

The function max() returns the number of elements from the tuple with maximum value.

Syntax

max(tuple)

Parameters

tuple- This is a tuple from which max valued elements to be returned.

Return value

It returns the elements from the tuple with maximum value.

Example

```
tup1=('abc','xyz','pqrs')
tup2=(456,700,200)
print(max(tup1))
print(max(tup2))
```

Output-

xyz

700

min(tuple) –

Description

The function min() returns the number of elements from the tuple with minimum value.

Syntax

min(tuple)

Parameters

tuple- This is a tuple from which min valued elements to be returned.

Return value

It returns the elements from the tuple with minimum value.

Example

```
tup1=('abc','xyz','pqrs')

tup2=(456,700,200)

print(min(tup1))

print(min(tup2))
```

Output-

```
abc

200
```

tuple(seq) –

Description

The function converts a list into tuple.

Syntax

tuple(seq)

Parameters

seq- This is a list which is to be converted into tuple.

Return value

It returns the tuple .

Example

list1=[456,700,200]

print(tuple(list1))

Output-

(456, 700, 200)

Methods of Tuple: (Not in syllabus read once)

count- Returns the number of times a specified value occurs in a tuple.

index()- Searches the tuple for a specified value and returns the position of where it was found.

Example

tuple1=(123,789,123)

print(tuple1.index(789))

print(tuple1.count(123))

Output-

1

2

Sets :

- Sets are lists with no duplicate entries.
- In Python, **Set** is an unordered ***collection of data type*** that is ***iterable, mutable*** and has ***no duplicate elements***.
- The ***order of elements in a set is undefined*** though it may consist of various elements i.e.It is an unordered collection of unique data elements.
- ***Duplicates will be removed***. i.e. set will be a collection of similar data items without repetition.
- The ***elements of sets are unindexed*** that means we cannot access set items by referring to an index.
- Sets are changeable(mutuable) i.e. we can change or update a set as and when required. Type of elements in a set need not be same, various mixed data type values can also be passed to the set.
- 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.

Accessing elements in Set–

- A set contains an unordered collections of items. Set is defined by values separated by comma inside braces {}. Another method is by making use of ()

Example–

```
# method1 to define set using {}

set1={'x','y','z'}

print(set1)

# method2 to define set using ()
```

```
set2=set(['x','y','x','z'])
print(set2)
```

Output-

```
{'z', 'y', 'x'}
{'z', 'y', 'x'}
```

- ***Index cannot be used*** in sets to access the elements of set as it is ***unordered***.
- ***Loops*** are used to access the elements as shown in example below

Example–

```
set1=set(['p','q','r','s'])
for count in set1:
   print(count)
```

output-

```
p
r
q
s
```

Deleting values in Set–

- There are different ways to delete values in a set.
- Elements/items in set can be deleted using **pop**, **remove** or **discard** method as shown in example.
- **pop** method removes random item from a set and return it.

- **remove** method is used to remove element from a set. Similarly **discard** method is also used remove element from a set but the ***only difference between them is that remove method raises keyerror if the specified item is not present in the set.***

Example–

```
set1=set(['p','q','r','s'])

print(set1)

set1.pop()

print(set1)

set1.discard('q')

print(set1)

set1.remove('a')
```

output-

```
{'r', 'p', 'q', 's'}

{'p', 'q', 's'}

{'p', 's'}

Traceback (most recent call last):

  File "D:\python programs\trial.py", line 7, in <module>

    set1.remove('a')

KeyError: 'a'
```

Updating Sets–

- Elements/items in set can be added by using **add** or **update** method as shown in example.
- The add() method takes one argument which the element we want to add in set.
- Loops can be used to add multiple elements at a time using add() method.

Example–

```
set1={'p','q','r','s'}

set1.add('E')

print(set1)
```

output-

{'r', 'p', 'q', 's', 'E'}

- The **update()** method accepts lists, strings, tuples as well as other sets as its arguments. In all of these cases, duplicate elements are avoided.

Example–

```
set1={'p','q','r','s'}

set1.add('E')

print(set1)

set1.update({'G','A'})

print(set1)
```

output-

{'r', 'E', 'p', 's', 'q'}

{'r', 'E', 'A', 'G', 'p', 's', 'q'}

Frozen Sets– (not in syllabus read once)

- These sets **cannot be redefined** once they are declared as **frozenset**. Hence they are called as **immutable (unchangeable) sets**. Example is as given below.

Example–

```
set2=frozenset({'a','b','c'})
print(set2)
set2.add('g')  # cannot add item as it is frozenset
print(set2)
```

output-

```
frozenset({'a', 'c', 'b'})
Traceback (most recent call last):
  File "D:\python programs\sets1.py", line 4, in <module>
    set2.add('g')
AttributeError: 'frozenset' object has no attribute 'add'
```

Basic Set Operations–

- Set has four basic operations which are enlisted below-

Union, Intersection, Difference and Symmetric Difference

Union-

- Union operation performed on two sets ***returns all the elements from both the sets.***

- Union of set A and set B is defined as the ***set that consists of all elements belonging to either set A or set B(or both)***. It is performed by using **|** operator.

Example–

```
A={1,2,4,6,8}                OR        A=set([1,2,4,6,8])
B={1,2,3,4,5}                          B=set([1,2,3,4,5])
C=A|B                                  C=A|B
print(C)                               print(C)
```

Output-

{1, 2, 3, 4, 5, 6, 8}

Intersection-

- Intersection operation performed on two sets ***returns all the elements which are common or in both the sets.***
- Intersection of set A and set B is defined as the ***set that consists of all elements that belong to both A and B***. It is performed by using **&** operator.

Example–

```
A={1,2,4,6,8}
B={1,2,3,4,5}
C=A&B
print(C)
```

Output-

{1, 2, 4}

Difference-

- Difference operation performed on two sets ***returns all the elements which are present in set 1 but not in set2.***
- It is performed by using **-** operator.

Example–

A={1,2,4,6,8}

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

C=A-B

print(C)

Output-

{8, 6}

Symmetric Difference-

- Symmetric difference operation is performed by using ^ operator.
- The Symmetric difference of two sets ***returns all the elements which belongs either to set 1 or set2 but not both.***
- It is performed by using - operator.

A={1,2,4,6,8}

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

C=A^B

print(C)

Output-

{3, 5, 6, 8}

 # Program to perform different set operations

A = {0, 2, 4, 6, 8}

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

 # union

print("Union :", A | B)

 # intersection

print("Intersection :", A & B)

 # difference

print("Difference :", A - B)

 # symmetric difference

print("Symmetric difference :", A ^ B)

Output-

('Union :', set([0, 1, 2, 3, 4, 5, 6, 8]))

('Intersection :', set([2, 4]))

('Difference :', set([8, 0, 6]))

('Symmetric difference :', set([0, 1, 3, 5, 6, 8]))

Built-in Set Methods-

Add() – Adds an element to a set

Clear()- Removes all elements from the set

e.g.

A={1,2,4,6,8}

B={2,4}

```
print(A)
A.clear()
print(A)
```

Output-

```
{1, 2, 4, 6, 8}
set()
```

union() – Returns a new set containing the union of two or more sets.

intersection()- Returns a new set which is the intersection of two or more sets

e.g.

```
A={1,2,4,6,8}
B={2,4}
print(A.union(B))
print(A.intersection(B))
```

Output-

```
{1, 2, 4, 6, 8}
{2, 4}
```

difference() – Returns a new set containing the difference between two or more set.

e.g.

A = {0, 2, 4, 6, 8}

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

print(A.difference(B))

Output-

Difference : {0, 8, 6}

symmetric_difference()- Returns a new set with the symmetric differences of two or more sets.

e.g.

A = {0, 2, 4, 6, 8}

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

print(A.symmetric_difference(B))

Output-

{0, 1, 3, 5, 6, 8}

isdisjoint() – Return true if two sets have null intersection.

e.g.

A={1,2,4,6,8}

B={3,5}

print(A.isdisjoint(B))

Output-

True

issubset()- Report whether another set contains this set.

e.g.

A={1,2,4,6,8}

B={2,6}

print(B.issubset(A))

Output-

True

issuperset() – Determines whether one set is a superset of the

other.

e.g.

A={1,2,4,6,8}

B={1,2,4}

print(A.issuperset(B))

Output-

True

pop()- Remove and return an arbitrary set element. Raises keyerror if the set is empty.

e.g.

A={5,2,4,6,8}

print(A)

print(A.pop())

Output-

{2, 4, 5, 6, 8}

2

remove() – Remove an element from a set, which must be a member. If the element is not a member, raise a keyerror.

e.g.

```
A={5,2,4,6,8}
print(A)
A.remove(4)
print(A)
```

Output-

{2, 4, 5, 6, 8}

{2, 5, 6, 8}

e.g.

```
# Python program for Deletion of elements in a Set
set1 = set([1,2,3,4,5,6,7,8,9,10,11,12])
print("Intial Set: ")
print(set1)
```

 # Removing elements from Set using Remove() method

```
set1.remove(5)
set1.remove(6)
print("\nSet after Removal of two elements: ")
print(set1)
```

 # Removing elements from Set using Discard() method

```
set1.discard(8)
set1.discard(9)
print("\nSet after Discarding two elements: ")
print(set1)
```

 # Removing elements from Set using iterator method

```
for i in range(1, 5):
    set1.remove(i)
print("\nSet after Removing a range of elements: ")
print(set1)
```

Output-.

```
Intial Set:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
Set after Removal of two elements:
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}
```

Set after Discarding two elements:

{1, 2, 3, 4, 7, 10, 11, 12}

Set after Removing a range of elements:

{7, 10, 11, 12}

Built-in Set Functions-

all() – Return True if all elements of the set are True(or if the set is empty)

e.g.

```
set1={'p','q','r','s'}
print(all(set1))
set2={}
print(all(set2))
```

Output-

True

True

any() – Return True if any elements of the set is True. If the set is empty, return False.

e.g.

```
set1={'p','q','r','s'}
print(any(set1))
set2={}
```

print(any(set2))

Output-

True

False

len() – Return the length (number of items) in the set.

e.g.

set1={'p','q','r','s'}

print(len(set1))

set2={}

print(len(set2))

Output-

4

0

max() – Return the largest item in the set.

e.g.

set1={'p','s','r','q'}

print(max(set1))

set2={4,5,1,2}

print(max(set2))

Output-

s

5

min() – Return the smallest item in the set.

e.g.

```
set1={‘r','q',‘p','s'}
print(min(set1))
set2={4,5,1,2}
print(min(set2))
```

Output-

```
p
1
```

sorted() – Return a new ***sorted list*** from elements in the set.

e.g.

```
set1={4,5,1,2}
print(sorted(set1))
set1={'f','m','a','k'}
print(set1)
print(sorted(set1))
```

Output-

```
[1, 2, 4, 5]
{'a', 'm', 'f', 'k'}
['a', 'f', 'k', 'm']
```

sum() – Return the sum of all elements in the set.

e.g.

```
set1={4,5,1,2}
print(sum(set1))
```

Output-

```
12
```

#write a python program to create a set, add member(s) in a set and remove one item from a set

```
s1={5,8,10,5,7}
print(s1)
s1.add(3)
print(s1)
s1.add((2,6))
print(s1)
s1.remove(10)
print(s1)
s1.pop()
print(s1)
```

Output-

```
{8, 10, 5, 7}
{3, 5, 7, 8, 10}
```

{3, (2, 6), 5, 7, 8, 10}

{3, (2, 6), 5, 7, 8}

{(2, 6), 5, 7, 8}

 # Program to find maximum and the minimum value in a set

```
set1={'f','m','a','k'}
set2={4,9,6,2,8}
print(set1)
print(max(set1))  #print item with maximum value from set1
print(min(set1))  #print item with minimum value from set1
print(set2)
print(max(set2))  #print item with maximum value from set2
print(min(set2))  #print item with minimum value from set2
```

Output-

{'m', 'a', 'f', 'k'}

m

a

{2, 4, 6, 8, 9}

9

2

 # Program to find the length of a set

```
set1={'f','m','a','k'}
```

```
set2={4,9,6,2,8}

print(set1)

print(len(set1))

print(set2)

print(len(set2))
```

Output-

```
{'k', 'f', 'm', 'a'}

4

{2, 4, 6, 8, 9}

5
```

Dictionaries –

- **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 data structure is ***used to store key value pairs indexed by keys***.
- Items stored in a dictionary are ***not kept in any particular order***.
- It has a **key:value** pair where each value is associated with a key.
- A key and its value are seperated by a **colon(:)** between them.
- The items or elements in dictionary are ***separated by commas*** and all the elements must be ***enclosed in curly braces***.
- ***A Dictionary in Python works similar to the Dictionary in a real world. Keys of a Dictionary must be unique and of immutable data type such as Strings, Integers and tuples, but the key-values can be repeated and be of any type***.

Creating Dictionaries –

- The syntax for defining/creating a dictionary in python is

  <dictionary_name>={key1:value1,key2:value2,….keyN:value}

- A dictionary can be used to store a collection of data values in a way that allows them to be individually referenced.
- Rather than using an index to identify a data value, each item in a dictionary is stored as a key value pair.
- The simple method to create a dictionary is simple assign the pair of **key:value** pairs to the dictionary using operator(=).

Example to create dictionaries –

```
# Creating an empty Dictionary

Dict = {}

print("Empty Dictionary: ")

print(Dict)


# Creating a Dictionary with Integer Keys

Dict= {1: 'Amol', 2: 'Ramesh', 3: 'Arun'}

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

print(Dict)


# Creating a Dictionary with Mixed keys

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

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

```
print(Dict)

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

# Creating a Dictionary with each item as a Pair
Dict = dict([(1, 'Amol'), (2, 'Arun')])
print("\nDictionary with each item as a pair: ")
print(Dict)
```

Output–

```
Empty Dictionary:
{}
Dictionary with the use of Integer Keys:
{1: 'Amol', 2: 'Ramesh', 3: 'Arun'}
Dictionary with the use of Mixed Keys:
{'Name': 'Amol', 1: [1, 2, 3, 4]}
Dictionary with the use of dict():
{1: 'Amol', 2: 'Ramesh', 3: 'Arun'}
Dictionary with each item as a pair:
{1: 'Amol', 2: 'Arun'}
```

Accessing values in a dictionary –

- We can access the items of a dictionary by following ways:

 # Accesing a element from a Dictionary

```
Dict = {1: 'Amol', 'name': 'Ramesh', 3: 'Arun'}


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


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


# accessing a element using get() method
print("Acessing a element using get:")
print(Dict.get(3))
```

Output:

```
Acessing a element using key:
Ramesh
Acessing a element using key:
Amol
Acessing a element using get:
Arun
```

Deleting values in a dictionary –

- We can remove a particular item in a dictionary by using the method **pop()**. This method removes as item with the provided key and returns the value.
- The method, **popitem()** can be used to remove an arbitrary item(key,value) form the dictionary.
- All items can be removed at once using **clear()** method.
- The **del** keyword is also used to remove individual items or the entire dictionary itself.

Example–

```
# Initial Dictionary
Dict = { 5 : 'A', 6 : 'To', 7 : 'Z',
    'A' : {1 : 'Amol', 2 : 'Ramesh', 3 : 'Arun'},
    'B' : {1 : 'Amit', 2 : 'Rahul'}}
print("Initial Dictionary: ")
print(Dict)
```

o/p-

```
Initial Dictionary:
{5: 'A', 6: 'To', 7: 'Z', 'A': {1: 'Amol', 2: 'Ramesh', 3: 'Arun'}, 'B': {1: 'Amit', 2: 'Rahul'}}
```

```
# Deleting a Key value
del Dict[6]
print("\nDeleting a specific key: ")
print(Dict)
```

o/p-

Deleting a specific key:

{5: 'A', 7: 'Z', 'A': {1: 'Amol', 2: 'Ramesh', 3: 'Arun'}, 'B': {1: 'Amit', 2: 'Rahul'}}

 # Deleting a Key from Nested Dictionary

del Dict['A'][2]

print("\nDeleting a key from Nested Dictionary: ")

print(Dict)

o/p-

Deleting a key from Nested Dictionary:

{5: ‘A', 7: 'Z', 'A': {1: 'Amol', 3: 'Arun'}, 'B': {1: 'Amit', 2: 'Rahul'}}

 # Deleting a Key using pop()

Dict.pop(5)

print("\nPopping specific element: ")

print(Dict)

o/p-

Popping specific element:

{7: 'Z', 'A': {1: 'Amol', 3: 'Arun'}, 'B': {1: 'Amit', 2: 'Rahul'}}

 # Deleting an arbitrary Key-value pair using popitem()

Dict.popitem()

```
print("\nPops an arbitrary key-value pair: ")
print(Dict)
```

o/p-

```
Pops an arbitrary key-value pair:
{7: 'Z', 'A': {1: 'Amol', 3: 'Arun'}}
```

 # Deleting entire Dictionary

```
Dict.clear()
print("\nDeleting Entire Dictionary: ")
print(Dict)
```

o/p-

```
Deleting Entire Dictionary:
{}
```

Updating dictionary –

- You can update a dictionary by including another entry or a key esteem match.
- It is flexible data type that can be modified according to the requirements which makes it is a mutable data type.
- The ***dictionary are mutable means changeable***. We ***can add new items or change the value of existing items*** using assignment operator.
- If the ***key is already present, value gets updated***, else a new key:value pair is added to the dictionary.

Example-

```
dict1={'name':'vijay','age':40}
dict1['age']=35
print(dict1)
```

o/p-

```
{'name': 'vijay', 'age': 35}
```

Example-

```
dict1={'name':'vijay','age':40}
dict1['add']='Kolhapur'
print(dict1)
```

o/p-

```
{'name': 'vijay', 'age': 40, 'add': 'Kolhapur'}
```

Basic Dictionary Operations–

Dictionary membership Test:

- We can test if a key is in a dictionary or not using keyword **in.**
- It is the test for keys not for values**.**

Example-

```
dict1={1:1,2:4,3:9,4:16,5:25}
print(dict1)
print(1 in dict1)
```

```
print(6 in dict1)
```

o/p-

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
True
False
```

Traversing Dictionary:

- Using a for loop we can iterate through each key in a dictionary**.**

Example-

```
dict1={1:1,2:4,3:9,4:16,5:25}
for i in dict1:
   print(i,dict1[i])
```

o/p-

```
1 1
2 4
3 9
4 16
5 25
```

Built-in Dictionary Functions :

Built-in functions like **all(), any(), len(), cmp(), sorted()** etc. are commonly used with dictionary to perform different tasks.

- **all() -** Return ***True*** if all keys of the dictionary are true (or if the dictionary is empty).
- **any() -** Return ***True*** if any key of the dictionary is true. If the ***dictionary is empty, return False***.
- **len() -** Return the length (the ***number of items***) in the dictionary.
- **sorted()** -Return a ***new sorted list of keys*** in the dictionary.

Example-

```
squares = {1: 1, 7: 9, 5: 25, 3: 49, 9: 81}

squares1={}

print(all(squares))

print(any(squares1))

print(len(squares))

print(sorted(squares))
```

Output-

```
True

False

5

[1, 3, 5, 7, 9]
```

Built-in Dictionary Methods –(Not in syllabus read once)

clear() - Removes all the elements from the dictionary.

copy() – Returns a copy of the dictionary

fromkeys() – It creates a new dictionary with default value for al specified keys. If the default value is not speciifed all the keys are set to none.

get() – Returns the value of the specified key.

items() – Returns a list containing a tuple for each key value pair.

keys() – Returns a list containing the dictionary keys.

pop() – Removes the element with the specified key.

popitem() – Removes the last inserted key-value pair.

setdefault() - It is similar to get(), but will set dict[key]=default if key is not already in dict.

update() – Updates the dictionary with key-value pairs of another dictionary.

values() – Returns a list of all the values in the dictionary.

Example-

 # Built-in Dictionary methods

```
squares = {1:1, 2:4, 3:9, 4:16, 5:25}

squares3={}

print(squares.pop(4))

print(squares)

squares1=squares.copy()

print(squares1)

print(squares.popitem())

print(squares)

squares2=squares.fromkeys([1,3],5)

print(squares2)

print(squares.get(2))
```

```
for i in squares.items():
    print(i)
print(squares.keys())
squares1.setdefault(7,None)
print(squares1)
squares3.update(squares)
print(squares3)
print(squares.values())
squares.clear()
print(squares)
# delete the dictionary itself
del squares
# Throws Error
print(squares)
```

Output-

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{1: 5, 3: 5}
4
```

```
(1, 1)
(2, 4)
(3, 9)
dict_keys([1, 2, 3])
{1: 1, 2: 4, 3: 9, 5: 25, 7: None}
{1: 1, 2: 4, 3: 9}
dict_values([1, 4, 9])
{}
Traceback (most recent call last):
  File "F:\2019-2020\PYTH prog\dict_1.py", line 27, in <module>
    print(squares)
NameError: name 'squares' is not defined
```

- **write-a-program-to-input-any-two-tuples-and-interchange-the-tuple-values**

```
t1=tuple()
t2=tuple()
n =int(input("Total number of values m first tuple: "))
for i in range(n):
    a = input("Enter elements: ")
    t2 = t2 + (a, )
    print("First tuple")
    print(t1)
    print("Second tuple")
    print(t2)
    t1,t2= t2,t1
    print("After swapping")
    print("First tuple")
    print(t1)
    print("Second tuple")
    print(t2)
```

Output

```
Total number of values m first tuple: 4
Enter elements: 15
First tuple
()
```

Second tuple

('15',)

After swapping

First tuple

('15',)

Second tuple

()

Enter elements: 16

First tuple

('15',)

Second tuple

('16',)

After swapping

First tuple

('16',)

Second tuple

('15',)

Enter elements: 1

First tuple

('16',)

Second tuple

('15', '1')

After swapping

First tuple

('15', '1')

Second tuple

('16',)

Enter elements: 2

First tuple

('15', '1')

Second tuple

('16', '2')

After swapping

First tuple

('16', '2')

Second tuple

('15', '1')

- **Program to swap of key and value of dictionary**

```
# initializing dictionary
old_dict = {'A': 67, 'B': 23, 'C': 45, 'D': 56, 'E': 12, 'F': 69, 'G': 67, 'H': 23}


new_dict = dict([(value, key) for key, value in old_dict.items()])
```

```
# Printing original dictionary
print ("Original dictionary is : ")
print(old_dict)

print()

# Printing new dictionary after swapping keys and values
print ("Dictionary after swapping is :  ")
print("keys: values")
for i in new_dict:
    print(i, " :  ", new_dict[i])
```

Output

```
Original dictionary is :
{'A': 67, 'B': 23, 'C': 45, 'D': 56, 'E': 12, 'F': 69, 'G': 67, 'H': 23}

Dictionary after swapping is :
keys: values
67  :  G
23  :  H
45  :  C
56  :  D
12  :  E
```

69 : F

- **Python program to combine two dictionary by adding values for common keys**

```
# initializing two dictionaries
dict1 = {'a': 12, 'for': 25, 'c': 9}
dict2 = {'P': 100, 'W': 200, 'for': 300}

# adding the values with common key
for key in dict2:
    if key in dict1:
        dict2[key] = dict2[key] + dict1[key]
    else:
        pass

print(dict2)
```

Output

{'P': 100, 'W': 200, 'for': 325}

- **Python3 program to Count number of lists in a list of lists**

```
def countList(lst):
```

```
    return len(lst)

# Driver code
lst = [[1, 2, 3], [4, 5], [6, 7, 8, 9]]
print(countList(lst))
```

Output

3

- **Python3 code to demonstrate average of two lists using sum() + len() + "+" operator**

```
# initializing lists
test_list1 = [1, 3, 4, 5, 2, 6]
test_list2 = [3, 4, 8, 3, 10, 1]

# printing the original lists
print ("The original list 1 is : " + str(test_list1))
print ("The original list 2 is : " + str(test_list2))

# Average of two lists
# using sum() + len() + "+" operator
res = sum(test_list1 + test_list2) / len(test_list1 + test_list2)
```

```
# printing result
print ("The Average of both lists is : " + str(res))
```

Output

```
The original list 1 is : [1, 3, 4, 5, 2, 6]
The original list 2 is : [3, 4, 8, 3, 10, 1]
The Average of both lists is : 4.166666666666667
```