Module-2

List, Dictonaries And Tuples, Sets

⇒ **List:** A list is an ordered sequence of multiple values.

* It is a data structure in python
* The values inside the lists can be of any type (like int, float, strings, lists, tuples, dictionaries) are called as elements or items.
* A list value looks like : ['cat', 'bat', 'rat', 'elephant']
* A list need not contain data of same type. We can have mixed type of elements in list.

Ex ls 1 = [10, -4, 25, 13]

ls 2 = ["Tiger", "Lion", "cheetah"]

Ex ls 3 = [3.5, "Tiger", 10, [3,4]]

Here ls 3 contains float, a string, an integer and a list.

* This illustrates that a list can be nested as well.

Who to create a list in Python?

1. An empty list
2. List with elements
3. List with elements and methods

→ **An empty list:-** Can be created any of the foll ways

```
>>> ls = [ ] → empty list (which does not contains anything]
>>> type(ls)
    <class 'list'>
>>>ls=list ()    #list() Constructor with Zero arguments
>>> type (ls)
    <class 'list'>
```

→ List with elements :- Can be created any of the following

```
>>> ls = [10, -4, 25, 13]  : static way of creating
>>> type(ls)
    <class 'list'>
```

* A new list can be created using list() constructor by passing arguments to it as shown

```
>>> ls2 = list([9,3,4])  # list() Constructor with argument
>>> print (ls2)
    [9,3,4]
```

Getting a values in a list with Indexes :-

* The elements in the list can be accessed using a numeric index value of list.

1) Double Indexing:
2) forward indexing:
3) Backward indexing:

```
List = [1, 5.2, 9.6, "FACE", "Python", 8.6]
```

⇓ View of list → forward indexing

0	1	2	3	4	5
1	5.2	9.6	FACE	PYTHON	8.6
-6	-5	-4	-3	-2	-1

→ backward indexing

ex: 1.

```
>>> ls = [34, 'hi', [2,3], -5]
>>> print (ls[1])
    hi
>>> print (ls[2])
    [2,3]
```

If we wanted like to access the elements within inner list, we need to use double indexing as shown

```
>> print (ls[2][0])
    2
>> print (ls[2][1])
    3
```

→ Lists are mutable:-

* Yes lists are mutable
* That is, by using indexing, we can modify any value within list.
* In the foll ex, the 3rd element (index is 2) is being modified

```
>>> ls=[34,'hi', [2,3],-5]
>>> ls[2]="Hello"
>>> print (ls)
    [34,'hi',"hello",-5]
```

The index for extracting list elements has following properties

*Any integer expression can be an index.

```
>>> ls = [34,'hi',[2,3] -5]
>>> Print(ls[2*1])
   [2,3]
```

* Attempt to access non index present

```
>>> ls=[34, 'hi', [2,3], -5]
>>> print (ls[-5])
 error: index is not present
```

* A negative indexing counts from backwards

```
>>> ls= [34,'hi', [2,3],-5]
>>> Print (ls[-1])
    -5
>>> print [ls[-4])
    34
```

* The in operator applied on lists will results in a Boolean value

```
>>> ls = [34,'hi', [2,3],-5]
   Print>>>34 in ls
        True
>>> -2 in ls
 False
```

[∵ in means belong (∈) or not belong (∉)]

Getting a list's length with len():-

* the len() function will returns the number

List operations:{}

→Two operations are: list concatenation (using '+' operator)
List Replication (using '*' operator)

* The plus '+' operator can combine two lists to create a new list value in the same way it combines two strings into a new string value.

* The star '*' operator can also be used with a list and an integer value to replicate the list (i.e, returns a list by repeating itself for n times)

ex: ls=[] # empty list
type (ls)
<<class 'list'>>

* ls1=[2,3,4,"hi",[4,6]] # creating a list
Print (ls1)

* [2, 3, 4, 'hi', [4,6]]
ls2=["ISE", "CSE"]
ls3=[s1+ls2 # list concatenation
Print (ls3)
[2,3,4,'hi',[4,6],'ISE','CSE']

* ls2 = ls2*3 # list replication
['ISE','CSE','ISE','CSE','ISE','CSE']

* len (ls2)
6

* ls2 [4]
ISE

⇒ Traversing a list :-

* A list can be traversed using for loop
* If we need to use each element in the list, we can use the for loop and in operator as below
* Program: To print the elements in a list

```
ls = [34, 'hi', [2,3], -5]

for item in ls :
    print (item)
```

34, 'hi', [2,3], -5

* list elements can be accessed with the combination of the for loop. loop with range() & len() function as well

ex:
```
ls = [1,2,3,4]
for i in range (len (ls)):
    ls[i] = ls[i] ** 2
Print (ls)
```
output: 1,4,9,16

* ls = [1,2,3,4], sum = 0

```
for item in ls :
    sum = sum + item
print (ls)
print (sum)
```
output: [1,2,3,4]
10

*
```
ls = [1,2,3,4]
sum = 0
for i in range (len (ls)):
    sum = sum + ls[i]
Print (ls)
Print (sum)
```
output: [1,2,3,4]
10

⇒ List Methods :-

A python method is like a Python function, but it must be called on an objective (value)

* The method part comes after the value (object), seperated by a period.

Syntax: object.method-name (<args>) # method call

* Each data type has its own operations

→ Types of list methods:-

* Adding values of lists with the append() & insert()

*

1. Append(): This method is used to add a new element at the end of a list.

```
>>> ls=[1,2,3]
>>> ls.append("hi")
```

```
*res=ls.append(23.4)
print(res)
None
```

```
*ls.append("hello",45.6)
error: Type error [∵ because append takes only one element]
```

```
* ls1=["hello",45.6]  # create a list
  ls.extend(ls1)  # extend() appends the elements of the list argument to the end of the list.
  print(ls)
```

2. Index(): this method is used to get the index position of a particular value in the list.

* If that the value exists in the list, the index of the value is returned. If the value is not in the list then python value error.

```
ex: ls=[2,7,4,2,7,8,5]  # creating list
    ls.index(2)  # index of a particular element
    ls.index(10)  # Answer is value error
    ls.index(7,2,5)  # 7 is element, 2 4 5 denotes the range, search in the list from index 2 to 4
```

3) Sort(): It is used to sort the contents of the list. By default the function will sort the items in ascending order.

```
>>> ls=[3,10,5,16,-2]
>>> ls.sort()
>>> print(ls)
    [-2,3,5,10,16]
```

* we want to be sorted in descending order, we need to set the argument as shown

```
>>> ls.sort(reverse=True)
>>> print(ls)
[16,10,9,3,-2]
```

* ls = ['a', 'z', 'A', 'B']
  ls.sort()
  print(ls)
  ['A', 'B', 'a', 'z']

* ls.sort(key=str.lower) [# ascii betical order]
  Print(ls)
  ['A', 'a', 'B', 'z']

4) Reverse(): This method can be used to reverse the given list

```
ls=[3,4,6,7]
ls.~~sort()~~ reverse()
Print(ls)
[7,6,4,3]
```

5) Count(): This method is used to ~~xxxxx~~ count number of occurance of a particular value withtin list.

```
ls = [1,2,5,2,1,3,2,10]
ls.~~count~~) count()
3
```

6) Clear(): It removes all the elements in the list and

```
ls = [1,2,3]
ls.clear()
Print(ls)
[]
```

7) POP(): It deletes the last element in the list, by default and returns the deleted element

```
ls = [3,6,-2,8,10]
x=ls.pop()
Print(ls)
[3,6,-2,8]
```

- ⑧ when an element at a particular index position has to be deleted, then we can give that position as argument to pop()

```
t= ['a','b','c']
x=t.pop(1)
print(t)
['a','c]
print(x)
'b'
```

8) remove(): When we don't know the index; but known the value to be removed, then this method can be used.

```
ls=[5,8,-12,34,2]
ls.remove(34)
print(ls)
[5,8,-12,2]
```

Note: if the function will remove only the first occurance of the specified value, but not all occurances.

* Unlike pop() function, the remove function does not display which have been deleted.

Pgm1:

```
#Program to find sum of all the element in the list.*/
ls=[10,12,14,16]
Sum=sum(ls)
print("sum is",sum)
```

```
#Pgm to find sum of all the elements being read from the user
without using built in function */
ls=list()   # ls=ls[]
sum=0
while(True):
    x=input("Enter a number:")
    if x=='done':
        break
```

```
    ls.append(int(x))
for item in ls:
    sum = sum + item
print("Sum is", sum)
```

/* Program to print the numbers of a specified list after removing even numbers from it */

```
ls1 = [10, -6, 11, 7, 10, 6]
ls2 = []
for x in ls1:
    if x % 2 != 0
        ls2.append(x)
print("Resultant list in ls2").
```

7611533515

1. 0
2. a
3. b
4. c
5. d
6. 4
7. 3
8. 1
9. 2
10. 1

/* Program to illustrate operations of stack using list */

```
stack = [10, -6, 11, 9]
print("original stack", stack)
stack.append(12)
stak.append(45)
print("stack after push operation is", stack)
stack.pop()
print("stack after pop operation is, stack).
print("Value obtained after peek operation is",
                stack[len(stack-1)])
```

List Slicing:- To access any single element in a list, we use Indexing.

* To access a range of items in a list, we need to slice a list.
* One way to do this is to use the simple slicing operator.
* With this operator we can specify where to start the slicing, where to end a specify the step.

Slicing a list:-

If L is a list

Syntax: L[start : stop : step]

* The stop value indicates the index value to stop at. The element at this index is not included in the slice.
* The step value refers to the number of steps to jump.
* The start value indicates the starting index value.

These three values are optional when used together:

* when start value is missing, it has a default value 0.
* when step is missing, it takes the default value 1.
* when stop value is missing, it takes the value of length of list.

Ex: L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

Print (L[2:7])

L[2:7] = ['c', 'd', 'e', 'f', 'g']

start ↑ ↑ end

We can also specify the negative slicing a list.

L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

Print (L[-7:-2])

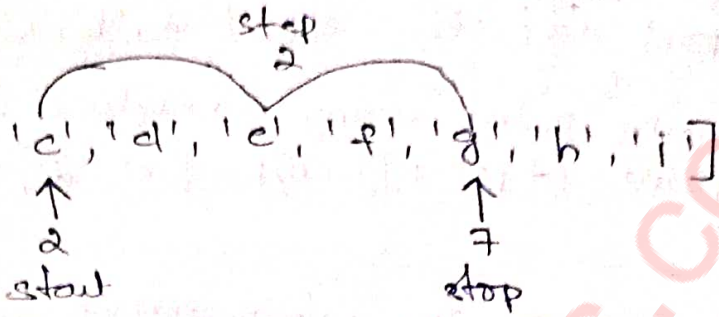
L[-7:-2] = ['a' (-9), 'b' (-8), 'c' (-7), 'd' (-6), 'e' (-5), 'f' (-4), 'g' (-3), 'h' (-2), 'i' (-1)]

start ↑ ↑ end

We can also specify the step of the slicing using step parameter.

ex: L=['a','b','c','d','e','f','g','h','i']

Print (L[2:7:2]

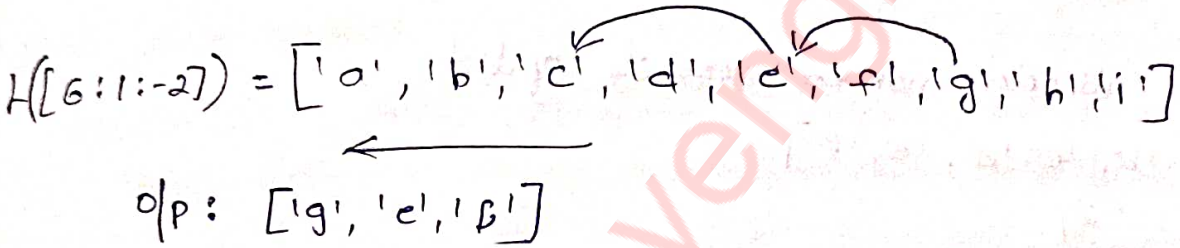


We can even specify the negative slicing.

* negative step values reverse the direction size.

ex: L=['a','b','c','d','e','f','g','h','i']

Print (L[6:1:-2])



o/p: ['g','e','b']

Pgms ex:

Consider a list t given below, and series of examples following based on this object. t = ['a','b','c','d','e']

* Extracting full list without using any index,

```
>>> print (t[:])
o/p: ['a','b','c','d','e',]
```

* Extracting elements from the second position,

```
>>> print (t[1:])
o/p ['b','c','d','e']
```

* Extracting first three elements

```
>>> print (t[:3])
['a','b','c']
```

* Selecting some middle elements
>>> print (t[0:4]) o/p = ['c','d']

* Using negative indexing
>>>print (t[:-2]) o/p = ['a','b','c']

* Reversing a list using a negative Index.
>>> print (t[6:-1]) o/p = ['e','d','c','b','a']

- del: This is an operator which is used to delete when more than item to be removed at a time.

* ls = [3,6,+2,8,1]
del ls[2]
Print (ls)
[3,6,8,1]

* ls = [3,6,-2,8,1]
del ls[1:4]
Print (ls)
[3,1]

- ex: delete all odd indexed elements of list.

t=['a','b','c','d','e','f']
del t[1::2]
Print (t)
['a','c','e']

Converting types with list():-

When we use the function str(42) will return 42, the string representation of the Integer 42.

* Similarly there is a method list() that will return list version of the value passed to them.

To convert a string into a list, we use the method list() as below

s="hello"
ls=list (s)
Print (ls)
['h','e','l','l','o']

* If we want a list of words from a sentence, we can use the code

```
s = "Hello how are you ?"
ls = s.split()
print (ls)
["Hello', 'how', 'are', 'you?']
```

Note: when no argument is provided, the split() function takes the delimiter as white spaces.

* If we want a list of splitting the lines. we can use as shown below

```
dt = "23/03/2000"
ls = dt.split ('/')
Print (ls)
['20', '3', '2000']
```

* There is a method join() which behaves opposite to split()
* It takes a list of strings as arguments, and joins all the strings into a single string based on the delimiter provided.

```
ls = ["Hello", "how", "are", "you?"]
d = ''
d.join(ls)
Print (ls)   o/p: 'Hello how are you'
```

References: when an list object is assigned to other using assignment operator, both of them will refer to same object in the memory.

```
ls1 = [1,2,3]
ls2 = ls1          [∵ Here ls2 is said to be reference of ls1].
ls1 is ls2
True.
```

* An object with more than one reference has more than one name, hence we say that object is aliased. If the aliased object is mutable, changes made in one alias will reflect the other.

```
ls2[1] = 34
Print (ls1)
[1,34,3]
```

Creating a tuple:

* Creating a tuple with one elements is a bit tricky.
* having one elements within parentheses is not enough.
* we will need a trailing comma to indicate that it is in fact a tuple.
* An empty tuple can be created either using a pair of parentheses or using the constructor tuple() as below

```
t1=()
type(t1)
<class 'tuple'>
    (or)
t2=tuple()
type(t2)
<class 'tuple'>
```

* If we provide an argument of type sequence (a list, a string etc) to the method tuple(), then a tuple with the elements in given sequence will be created.

Create tuple using string:

```
t=tuple('Hello')
print(t)
('H','e','l','l','o')
```

Create tuple using list:

```
t=tuple([3,[12,5],'Hi'])
Print(t)
(3,[12,5],'Hi')
```

Creating a tuple using another tuple:

```
t=('Mango',34,'hi')
t1=tuple(t)
Print(t1)
('Mango',34,'hi')
```

Accessing elements in a tuple:

* We can access the values in the tuple with their index, similar to lists.
* Nested tuples are accessed using nested indexing.
* Negative indexing can be applied to tuples similar to lists.
* We can access a range of items in a tuple by using the slicing operator.
* Trying to reassign a value in a tuple causes a type error.

Ex: ① marks = (23, 45, 32)

```
Print (marks[0])
Print (marks[2])
```

23
32

② nested tuple:

```
n-tuple = ("SIKANDER", [8,4,6], (1,2,3))
Print (n-tuple[0])
Print (n-tuple[1])

Print (n-tuple[0][0])
Print (n-tuple[1][0])
```

SIKANDER
[8,4,6]
S
8

Using Copy Module:-

* Python provides a module named copy that provides both the copy() and deepcopy() functions.
* The first of these Copy.copy(), can be used to make a duplicate copy of a mutable value.

ex: import copy # Copy Module
ls = [1,2,3]
lsl = copy.copy(ls) # copy()
print (id(ls))
Print (id(lsl))

import copy # Copy Module
ls = [1,2,3, [7.8, 8.9]]
lsl = copy.deepcopy(ls)
Print (id(ls))
Peint (id(lsl))

→ Write a python pgm to copy of a list.

```
import copy
Origional_list=[2,3,4,5,6]
dup_list= copy.copy(origional_list)
Print ("Original list", orgtional_list)
Print ("Copy of ~~cop~~ original_list",)dup_list).
```

/* Write Python program to Create a list of words that are longer than length 'n' from a given Sentence */

```
str = input ("Enter a Sentence")
n = int (input ("enter the length"))
long_list = [ ]
list = str.split ( )
for x in list :
  if len (x) > n :
     long list. append (x)
Print (long_list)
```

OP:

Enter a sentence Rama Seeta and Lakshman went to forst

Enter the length 3

[Rama Seetha Lakshman Went forest]

/* Write a python program that takes 2 lists & prints True if they have atleast one common member */

```
import sys
ls1 = [1,2,3,4,5]
ls2 = [5,6,7,8,9]
result = False
for x in ls1 :
  for y in ls2 :
       if x == y :
            result = True
Print (result)
Print (result)
break sys.exit ( )
```

/* Write a python program to print last five & first five elements */

ls=[10, 3, 2, 4, 6, 8, 5, 4, 12, 23, 45]

ls1=ls[0:5]

ls2=ls[8:-5:0] or ls2=ls[:4:-1] or ls2=ls[8:]

Print (" First five elements of list are", ls1)

Print (" Last five elements of list are", ls2)

Tuple: Tuple is a sequence of items, similar to lists

* The value stored in the tuple can be of any type & they are indexed using integers.
* Unlike lists, tuples are immutable. That is values within tuples cannot be modified.
* A tuple can be created in python as a comma separated list of items - may or may not be enclosed within parenthesis rather than square brackets.

Advantages of Tuple over list:-

* We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
* Tuples that contain immutable elements can be used as key for a dictionary with list, this is not possible.
* If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Creating a tuple:-

* A tuple is created by placing all the items inside a parenthesis (), separated by comma.
* The parenthesis are optional but is a good practice to write it
* A tuple can have any number of items and they may be of different type (integer, float, list, string).

Tuple operations:

* we can use + operator to combine two tuples. This is also called concatenation.
* we can also repeat the elements in a tuple for a given no of times using the * operator.
* Both + and * operations results into a new tuple.

e.g. Print (1,2,3)+(4,5,6)
Print (("Repeat")*3)
(1,2,3,4,5,6)
('Repeat', 'Repeat', 'Repeat')

Modifying Tuples:-

* unlike lists, tuples are immutable.
* This means that elements of a tuple cannot be changed once it has been assigned.

t=("Mango", "Pineapple", "Orange")
t[0] = 'Kiwi'
Type error

* But tuple can be replaced with another tuple involving required modifications

t=("Kiwi",) + t[1:]
Print (t)
('Kiwi', 'Banana', 'Apple')

Deleting a Tuple:-

* We cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple.
* But deleting a tuple entirely is possible using the keyword del

my-tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
del my-tupple [3]
Type error
del my-tuple
Name error

Tuple methods:- Methods that add items or remove items are not available with tuple. Only the following two methods are available.

count(x) → return the number of items that is equal to x.
index(x) → return index of first item that is equal to x.

ex: my_tuple = ('a', 'p', 'p', 'l', 'e')
Print ("Total count of element p is", my_tuple.count('p'))
Print ("Index of l is", my_tuple.index('l'))
Total count of elements p is 2
Index of l is 3.

Bulit In functions with Tuple:-

1) len()
2) max()
3) min()
4) sum()
5) any()
6) all()
7) sorted()
8) tuple()

Iterating a tuple:

```
names = ("Amrith", "John", "Mary")
for name in names:
    print ("Hello", name)
```

Cloning of a tuple using copy:-

```
import copy
names = ("Amrith", "John", "Mary")
dup_names = copy.copy(names)
Print (id(names))
Print (id(dup_names))
```

* Copy.copy() & copy.deepcopy() just copy the reference for an immutable object like a tuple.
* Bcz a tuple is immutable, there's really no rationale to create another copy of that exact names.

Tuple assignment :-

* Tuple has a unique feature of having it at LHS of assignment operator.
* This allows us to assign values to multiple values at a time.

ex.
```
x,y = 10,20
print (x)
print (y)
```

* When we have list of items, they can be extracted and stored into multiple variables

```
ls = ["hello", "World"]
x,y = ls
print (x)  # print hello
print (y)  # print world
```

* Swapping of two values.

```
x,y = 10,20
Print ("Before swapping the values are")
Print ("x =", x)
Print ("y=", y)
temp = x
x = y
y = temp
x,y = y,x
Print ("After Swapping the values are")
Print ("x=", x)
Print ("y=", y)
```

Returning multiple values in Python :-

1) Using object
2) Using tuple
3) Using list
4) Using dictionary

} four ways to returning multiple values.

Ex: Pgm1

Write a function calculation () such that it can add and subtraction.

```
def calculation (num1, num2):
    add = num1 + num2
    sub = num1 - num2
    return add, sub @ add /sub
# reading 2 numbers
n1 = int (input ("Enter a first number:"))
n2 = int (input ("Enter second number:"))
# function call
sum, sub1 = calculation (n1, n2)
# Print the values
Print ("Addition=", sum)
Print ("Subtraction=", Sub1)
```

⇒ Dictionaries :- It is a collection of unordered collection of elements.

* While other compound data types have only value as an element, a dictionary has a key : value pair.
* A dictionary is a set of key:value pairs, with the requirments that are unique in a dictionary.
* The values are accessed using keys.
* A key in dictionary can be an immutable type like strings, numbers & tuples.
* Each key maps to a value.
* A dictionary is mapping blw set of elements & keys.

Creating a empty dictionary :-

* An empty dictionary can be created using 2 ways

1) d={} # using empty curly parenthesis

or

d=dict() # using dict() constructor without argument

* To add items to dictionary, we can use square brackets

```
>> d={}
   d["Mango"]="fruit"
      dictionary
   d["Banana"]="fruit"
   d["Cucumber"]="Veg"
   Print (d)
```

O/P:

{"Mango':'fruit', 'Banana':'fruit', 'Cucumber':'Veg'}

Creating a dictionary:-

* Creating a dictionary is as simple as placing items inside curly braces{} seperated by comma.
* Each element in a dictionary is represented by a key: value pair.
* While value can be of any data type & can repeat

Ex: tel_dir = {'Tom' : 3491 , 'Jerry' : 8135}
Print (tel_dir)

{'Tom' : 3491, 'Jerry' : 8135}

Accessing Elements from a Dictionary:-

The len() function on dictionary object gives the number of key values

```
d = {'social' : 45 , 'Maths' : 35, 'Science' : 40}
Print ("Hindi")
```

Key error.

Modifing of Dictionary :-

```
# Creating empty dict
d={}
Print (type(d))   # Instance of class dict
<class 'dict'>
d=dict ()   # using constructor
Print (type(d))
<class 'dict'>
```

populate the dictionary

d["Tom"] = 2345

d["Jerry"] = 5678

Print (d) # prints the dictionary with 2 key/value pairs

{'Tom': 2345, 'Jerry': 5678}

- len (d) # returns the length, i.e no of key/value pairs

2

* Print (d["Jerry"]) # prints the value associated with 'jerry'

5678

* Print (d["Donald"]) # Error

KeyError: Donald

* d.get ("Tom")

2345

* d.get ("Donald") # return none

Print (d.get ("Donald"))

None

* print (d)

{'Tom': 2345, 'Jerry': 5678}

* d["Tom"] = 3456 # Dictionary mutable

Print (d)

{'Tom': 3456, 'Jerry': 5678}

* Dictionary are mutable. We can add new item or change the value of existing item using assignment operator.

Deleting elements from a dictionary:-

* We can remove a particular item in a dictionary by using method pop(). This method removes as item with the provided key & returns the value.
* All the items can be removed at once using clear() method.
* We can also use del keyword to remove individual item
* Popitem() is used to remove or return the the particular item.

Ex: ① d["Tom"] = 3456
d["Jerry"] = 5678
res = d.pop("Tom") # pop() method & res Contain 3456
Print (d)
Print (res)
{"Jerry": 5678, 'Donald': 3456}
3456

* res = d.pop ("Mickey")
Keyerror.

* del d["Jerry"]
{'Donald': 3456}

② d["Tom"] = 3456
d["Jerry"] = 5678
res: d.popitem() # removes random element
Print (d)
{'Donald': 3456, 'Tom': 3456}

* d.clear # empty dictionary
Print (d)
{}

* d = {'Tom': 3456}
del d
print(d) # name error
name error : d not defined.

Dictionary Membership test :-

* We can test if a key is in dictionary or not using the keyword in.

Notice the membership test is for keys only, not for values.

* The in operator can be used to check whether any key (not value) appears in the dictionary object.

Ex: Squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
Print (" 1 in Squares :", 1 in Squares) → True
Print (" 2 in Squares:", 2 not in Squares) → True
Print (" 49 in squares :", 49 in Squares) { this is value it returns false.

Iterating through a dictionary :-

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

Ex:
tel_dir = {'Tom': 3491, 'Jerry': 8135, 'Mickey': 1253}
for name in tel_dir:
    Print (name)
O/P {Tom, Mickey, Jerry}

* If we want to print key and values seperately, we need use the

* tel_dir = {'Tom': 3491, 'Jerry': 8135, 'Mickey': 1253}
for name in tel_dir :-
    Print (name, ":", tel_dir (name))
O/P tom: 3491
Jerry: 8135
Mickey: 1253.

* 'tom' in tel_dir # membership test
  True

* '4569' in tel_dir # membership test (its value is False)

The keys(), values(), items() Methods:-

* There are three dictionary methods that will return list like value of the dictionary's keys, values or both keys and values: keys(), values() & items()

* The values returned by these methods are not true lists: they cannot be modified and do not have append() method.

* Using the keys(), values(), items() methods a for loop can iterate over the keys, values or key-value pairs in a dictionary.

* (dict_keys, dict_values, dict_items) can be used in for loop.

ex:- tel_dir.keys()
  dict_keys (['tom', 'Jerry', 'Micky'])

* tel_dir.values()
  dict_values ([2345, 4569, 3849])

* tel_dir.items()
  dict_items ([('tom', 2345), ('Jerry', 4569), ('Micky' 3849)])

The GET() Method:-

* We need to check whether a key exists in a dictionary before accessing that key's value.
* In dictionaries, we have a get() method that takes two arguments: they key & a fallback value.
* If key is present it returns its value else it returns the default value if that key does not exists.
* If dc

```
# Write a python pgm to print the frequency of characters in a string */
s=input ("Enter a string:")          #read
d=dict()                             # create empty dict
for ch in s:                         # traverse through string
    d[ch]=d.get(ch,0)+1              # using get() method with default value
print(d)
```

The SET Default () Method:-

* We can set a value in a dictionary for a certain key only if that they does not already have a value.
* The setdefault() method offers a way to do this in one line of code.
* The first argument passed to the method is the key to check for, and the second argument value to set at key if the key does not exist. If the key does exists, the setdefaullt() method returns the key's value.

eg: d={'name':'Vishnu','age':20}
res= d.setdefault("Qualification","Degree")
Print(d)

e) s=input("Enter a string:")
d=dict()
for ch in s:
d.setdefault(ch,0)
d[ch]+=1 (or) d[ch]=d[ch]+1
Print(d)

→

Ex: d={"Tom":3456, "Jerry": 4644, "Mickey": 5657}

```
Print (d.get ("Jerry", 0)) #default value is useful, when key doesn't exists
```

o/p: 4644

```
d={"Tom":3456, "Jerry":4644, "Mickey": 5657}
Print (d.pop("Jerry",0))
4644
Print (d)
```

o/p {'Tom':3456, 'Mickey':5657}

Working of fromkeys()

```
ls=['apple','mango', 'grapes'] # list to act as keys
Value = 25
fruits = dict.fromkeys (ls, Value) # first arg is key and second argu
Print (fruits)
```

o/p {'apple':25, 'mango':25, 'grapes':25}

```
d1 = dict.fromkeys (d,25)
Print (d)
Print (d1)
```

{'Tom' : 3456 , ' Mickey' : 5657}

{'Tom' : 25 , 'Mickey' :25}

```
alphabets = dict.fromkeys ("Sai Vidya",0)
Print (alphabets)
```

{'s':0, 'a':0, 'i':0, 'v':0, 'd':0, 'y':0}

Working of update method :-

```
d= {"one":1 , "Two":2}
d1 = {"Two":"two"}
d.update (d1)  # update ()
Print (d)
```

op: {'one' :1 , 'Two' : two}

```
# copy () method
dup-d = d.copy ()
Print(id(dup-d))
Print (id (d))
```

3064060873688
8168

Working of popitem ()

```
d.popitem ()
'Two': two)
.pop item ()
: {'one' : 1, 'two' : 2)
```

Using tuples as keys in Dictionaries:-

* We can have tuples as keys in dictionaries, as they are immutable.
* Ex: We May need to create a telephone directory where name of a person is firstname - lastname pair & value is the telephone number job.

```
names = (('Tom','Cat'), ('Jerry','mouse'), ('Doland','Duck')) # tuple acting as k
number = [3561, 4014, 9813]
tel Dir = { }

for i in range (len(number)):
    tel Dir [names [i]] = number [i] #populating the dictionary

for fname, lnames in telDir:
    Print (fname, lname, ":", tel Dir [fname, lname])
```

O/P
```
Tom Cat : 3561
Jerry mouse : 4014
Doland Duck : 9813.
```

Introduction:-

* String is a Sequence of characters enclosed either with a pair of single quotes or double quotes.

Ex: "Hello" or 'World'

* Each character of a string corresponding to an index number, starting with zero as shown

Ex: Str = "HELLO WORLD"

H	E	L	L	O	Space	W	O	R	L	D
0	1	2	3	4	5	6	7	8	9	10

{ [len - 1]

* The characters of a string can be accessed using index enclosed within square brackets

ex: word1 = "Hello"
word2 = "Hi"

x = word1[1]
Print (x)
output:

y = word2[0]
Print (x)
output:

Q: Whether python

Len() function:-

The len() function can be used to get the length of a string.

```
Var = "Hello"
len = len (var)
Print (ln)
```

o/p : Index error : string out of range

Note: The index range should be from 0 to -1.

Traversing a string with a loop:-

for loop is easier than while loop traversing.

Using for loop

```
str = "Hello"
for i in str:
    Print (i, end = "\t")
```

o/p: H e l l o

Using while loop

```
str = "Hello"
i = 0
while (i < len (st)):
    print (i, end = "\t")
    i = i+1
```

o/p: H e l l o

String Slices:-

A Segment or a portion of a string is called slice.

* only a required number of characters can be extracted from a string using colon (:) Symbol.

Syntax:

st [i : j : k]

[Hello world]

Ex: st [7:]

~~LOLOL~~

WORLD ⇒ ORLD

* st : [3:8]

LOWO

* st [3:8:2] = LWO

* st [1:8:3] = EOO

* st [5:-2] = _WOR

* st [-4:] = ORLD

* st [-8:-2]

* st [-1:]

* st [:-1]

* st [::-1] = ~~DLROW OLLEH~~

* st [::-2] = ~~LROW-O~~

Strings are immutable :-

The objects of string class are immutable, once the strings is created they cannot be modified.

No character in the string can be edited / deleted / added.

```
* str = "Hello World"
  str[3] = 't'
Type error: 'str' object
```

```
* ~~str = "Hello World"~~
  Str = "Python"
  Print (type (str))
o/p <class 'str'>
  ~~Print~~ (
* str = "Pythoo"
  str1 = str[:len (str)-1]+'N'
  Print (str1)
o/p : PythoN.
* Print(str*3)
  Python Python Python
* for i in str:
      Print (i)
    P
    y
    t
    h
    o
    n
* "el" in "Hello" # in operator - True
o/p True
* "eo" in "Hello"
o/p False
```

In Operator:- The In operator of python where it has the boolean value True or false.

* The string should be oresiding then it will be true.

ex: "el" in "Hello"

True

"eo" in "Hello"

false.

Pgm of Palindrome:-

```
word = "Python"
reverse = word [::-1]
Print (reverse)
if word = = reverse:
    Print ("Palindrome")
else:
    Print ("Not a palindrome")
```

nothtyp

Not a Palindrome.

To Count no of vowels:-

```
word = "Python programming"
vcount = 0
vowels = "aeiou"
for ch in word:
    if ch = vowels:
        Vcount += 1
Print ("Number of vowels", Vcount)
```

ex: Vowels = 4.

To check whether the function is anagram or not:-

```
def check (s1,s2):
    if (sorted (s1) == sorted (s2)):
        Print ("The strings are anagrams")
    else:
        Print ("The strings are not anagrams")

s1= listen
s2= silent
check (s1,s2)
```

o/p:- The strings are anagrams.

To remove the duplicate characters from character set array

```
def removeDuplicate (str,n):
    index=0
    for i in range (0,n):
        for j in range (0,i+1):
            if (str[i] == str[j]):
                break
            if (j == i):
                str[index] = str[i]
                index = index+1
        return "".join (str[:index])

str = "VishnuSai"
n= len(str)
Print (remove Duplicate (list (str),n))
```

o/p: Vishnua

(or)

```
str = input ("Enter a string")
str1 = " "
for i in str:
    if (i not in str1):
        str = str1+i
    else:
        continue
Print (str1)
```

```
def count_char(string,ch):
    count=0
    for i in string
```

Anagram Program:-

```
def areAnagram(str1, str2):
    n1= len(str1)
    n2= len(str2)

    if n1!=n2:
        return 0
    str1 = sorted(str1)
    str2= sorted(str2)
# for i in range(0,n1):
    if str1[i]!=str2[i]:
        return 0
    else:
        return 0.
str1 = "Rat"
str2 = "Tom"
if areAnagram(str1,str2):
    Print("the two strings are anagrams")
else:
    Prints("the two strings are not anagrams").
```

⇒ String Methods:-

capitalize(): This method returns a new string with the first letter capitalized and all other characters lowercased.

Upper(): This method converts the given into Upper case.

Lower(): This method converts the given into lower case.

Swapcase(): This method converts all Upper Case to lowercase and lower case to upper case().

title(): This method which returns a string with first letter of each word capitalized, a title cased string

Count() : This method returns the no of occurances of the substring in the given string.

Syntax: (Invoking) string.count(substring, start, end)

substring: String whose count is to be found.

start: starting index within the string where search starts

end: ending index within the string where search ends.

start with(): these methods returns true if the string value
end with() they are called on begins or ends the string.

types of string methods:-

isupper(), islower(), isalpha(), isdigit(), isalnum(), isspace(), istitle()

→ Pgm to check the password:

```
import sys
s=input("Enter the password:")
if len(s)<6 or len(s)>16:
    Print('length should be btw 6 to 16 characters")
    sys.exit('length should be btw 6 to 16 characters')
Spected = "$#@"
if ((s.islower()==False)and (s.isupper()==False)and (s.isalnum==
    False)and (s.find('$')!=-1 or s.find('@')!=-1 or
    (s.find('#')!=-1))
    Print(s,' is valid password')
else:
    Print("Password is not valid")
```

→ Panagram Program :-

```
import string
import sys

s=input ('Enter the string:')
alphabets = list (string . ascii lowercase)

for ch in alphabets:
    if ch not in s.lower():
        Print (s, 'is not pangram string')
        sys. exit()

Print (s,' is pangram string').
```

→

find() : This method accept a parameter of str type & return index of the sub-string if the string object contains the specified sub-string, else return -1

Syntax:
StringName . find (substring , [START], [END])

index() : Similar to find() method, the only difference is that it raises an Exception when it doesn't find the sub-string

replace() : This method returns a copy of the string where all occurances of a substring is replaced with another substring.

Syntax:
string replace . (old, [new], [

Print the longest word and having how many characters

```
def longest_word (sentence):
    ls = sentence.split()
    longest = max (ls, key=len)
    return longest, len (longest)

sentence = input (" enter the sentence")
res1, res2 = longest_word (sentence)
print (" the longest word of the sentence")
```

Program to add ing at the last ---*/

```
def string_end (str):
    length = len (str)
    if length > 2:
        if str [-3:] == 'ing':
            str += 'ly'
        else:
            str += 'ing'
    return str
word = input (" enter the word:")
print (string_end (word))
```

⇒ Working with Strings:-

* Typing string values in Python code is fairly straightforward They begin and end with a single quote.
* But then how can you use strings?

```
name = 'This is Alice's cat':
print (name)
```

o/p : invalid

usage of double quotes :

```
name = "This is Alice's cat":
print (name)
```

o/p: This is Alice's Cat

Double Quotes :-

* Strings can begin & end with double quotes, just as they do print the required one.

ex: name = "This is Alice's Cat"
Print (name)
o/p : This is Alice's Cat

Escape Characters :-

* An escape character consists of a backslash (\) followed by the character want to be added to string.

ex: name = 'this is Alice\'s cat' # backward slash
Print (name)
o/p : ~~invalid~~ This is Alice's cat

name = 'this is Alice\"s cat'
Print (name)
This is Alice"s Cat

\'	single quote	types of escape characters
\"	double quote	
\t	tab	
\n	newline	
\\	back slash.	

Raw Strings:-

* A raw string completly ignores all escape characters and prints any backslash that appears in string.

#Create a string that contains 'That is Carol's cat!'

name = r'That is carol's cat' #raw string.

print (name)

o/p: That is Carol's cat

Multiline Strings:-

multine Comments:-

* while the hash character (#) marks be beginning.

Copying & Pastings strings with the Pyperclip Module:-

* The Pyperclip module has copy() & paste() functions that can send text to & receive text from computer clipboard.

ex: import pyperclip
pyperclip.copy('Hello world!") # copy to clipboard.
pyperclip.paste() # Hello world!

o/p: 'Hello world!"

Password Locker:-

* Users have accounts on many different websites
* It's best to use password manager software on computer that uses one master password to unlock the password manager.
* Then copy any account password to the clipboard and paste it into the website's Password field.

ex: PASSWORDS = {'email': 'F7min', 'blog': 'Vm2Va',
'facebook': '123'}

import sys. pyperclip

~~#!~~
if len(sys.argv)<2:
print ('Usage: py pw.py [account]- Copy account password')
sys.exit()
account = sys.argv[1] # first command line
if account in PASSWORDS
pyperclip.copy(PASSWORD[account])
Print ('Password for' + account + ' coppied to clipboard')