Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

O-LEVEL (M3-R5.1)
Python Programming By
Mrs. Geeta
Python Instructor
Zed-King Group of Institutions
Kaithal

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

O-LEVEL (M3-R5.1)

Python Programming

Video Tutorial

O-LEVEL (M3-R5.1) Full Video Playlist Available on YouTube Channel **Zed-king Computer Institute Kaithal**

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

Free Online Computer Classes On YouTube Channel

https://www.youtube.com/@zedkinginstitute

For Free PDF Notes

Our Telegram Channel

t.me/boost/Zedking_Institute

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

O-LEVEL (M3-R5.1)
CHAPTER-5

SEQUENCE DATA TYPES

List:-

Python has 'List' data structure. List elements have a sequence and are mutable (changeable). The elements in Python's list are called 'items'. Every item in the list is separated with comma (,). And the entire items are closed inside square brackets ([]). Mixed data types can also be used in the list.

Example: I=[1,2,3,4,5,6] print(I) print(type(I))

Note:- 1. List data type allow heterogeneous element

2. The list data type follows the insertion order, that is, the data will be printed in the order in which you input the data.

3. List data creates mutable object, that is, it allows modification in its content.

4. List data creates mutable object, that is, it allows modification in its content.

5. list support both positive & Negative indexing.

```
list1 = [1, 2, 3, 4, 5]                    #Integer List
list2 = [1.5, 2.4, 3.8, 4.4, 5.7];          #Float List
list3 = [1, 2, "Hello", 4.5];               #Mixed Data Types List
list4 = [1, 2, [1, 2], 4.5]                 #Nested List
```

Creating List in Python

There are many ways to prepare lists in Python.

1. Empty List:-

To create an empty list, simply square bracket ([]) is used.

Example:-

```
my_list = []

print(my_list)                # [ ]

print(type(my_list)           # list
```

2. List with Items:-

To create a list with List with Items, items are placed in square brackets, separated by commas (,).

Example:-

```
I=[10,20,30,40]

print(I)
```

3. Dynamic list creation:-

```
l=eval(input("Enter list element in square bracket ")
print(l)
print(type(l))
```

4. Using List function:-

List function is used to Separate character by character

```
list1=list(input("enter element of list "))
print(list1)
```

5. Using split function:-

Split function is used to separate word by word.

```
list2="ak it solution"
print(list2)
l=list2.split()
print(l)
print(type(l))
```

Accessing Element of List:-

Two methods are used to access data from the list-

1. **By using index number**
2. **By using slice operator**

1. **By using index number:-**

List1	**10**	**20**	**30**	**40**	**50**	**60**
Postive index	0	1	2	3	4	5
Negative index	-6	-5	-4	-3	-2	-1

Note:- Python supports both positive & negative indexing.

positive index means left to right

```
print(list1[0])          # 10
print(list1[3])          # 40
```

Negative index means right to left

```
print(list1[-1])         #60
print(list1[-3])         #40
```

2. By using slice operator:-

Syntax:- [start:stop:step]

Start:- This indicates the starting index which has a default value of (0).

Stop:- It indicates the last index in the list whose by default value is one less than the maximum length in the list.

Step:- This is an increment value, its default value is

List1	**10**	**20**	**'Anil'**	**40**	**'Ram'**	**60**
Postive index	0	1	2	3	4	5
Negative index	-6	-5	-4	-3	-2	-1

```
print(list1[0:6])          # [10, 20, 'anil', 40, 'ram', 60]
print(list1[2:4])          # ['anil', 40]
print(list1[:5:])          # [10, 20, 'anil', 40, 'ram']
print(list1[2::])          # ['anil', 40, 'ram', 60]
print(list1[-1:-4:-1])     # [60, 'ram', 40]
```

Traversing Elements of List:-

Sequential accessing of each element in a list is called traversing.

Using 'in' operator inside for loop:-

In operation is used to iterate each element through for loop in List sequence.

```
list1=[10,20,'anil',40,'ram',60]

for x in list1:

    print(x)

color=['red','blue','green','orange','purple','cyan','teal']

if 'purple' in color:

    print("Yes")
```

Using range() Function:-

This function provides the sequence between 2 integer numbers.

```
for i in range(5,15):
    print(i)
```

Note:- You can convert it into a list by using the list() function.

```
print(list(range(1,5)))                    # [1, 2, 3, 4]
```

Using while loop:-

```
number=[10,20,30,40,50,60]
i=0
while i<6:
   print(number[i],end=' ')
   i=i+1
# 10 20 30 40 50 60
```

List Operations:-

Many operations are performed on the list which are as follows-

1. Joining lists
2. Repeating lists
3. Slicing lists
4. Comparing lists

Joining Lists:-

Joining list means to concatenate (add) two lists. By using + operator, you can create a new list by joining two lists together.

```
x=[10,20,30]
y=[100,150,200]
c=x+y
print(c)
# [10, 20, 30, 100, 150, 200]
x=[10,20,30]
y=['apple','banana','mango']
c=x+y
print(c)
# [10, 20, 30, 'apple', 'banana', 'mango']
```

Note:- To concatenate two lists, both the arguments of + operator must be of list type.

```
x=[10,20,30]
y=2
print(x+y)
# TypeError: can only concatenate list (not "int") to list
```

Repeating List:-

Repetition operation is used to repeat the elements of the list up to the given numbers.

```
a=[10,20,30]
b=a*3
print(b)
# [10, 20, 30, 10, 20, 30, 10, 20, 30]
```

Slicing List:-

Slice operator also works on list

```
a=[10,20,30,'a', 'b', 'c']
print(a[1:3])
# [20, 30]
```

Comparison of list:-

Comparison operator is used to compare list objects. To compare, number of element, order of element and list should be case sensitive.

```
x=['Dog', 'Cat', 'Rat']

y=['dog', 'cat', 'rat']

z=['DOG', 'CAT', 'RAT']

print(x==y)          #False

print(x==z)          # False

print(x!=z)          # True
```

Note:- When comparing list objects between relational operators (>, <, >=, <=). So only the comparison of the first element of the list object is performed.

```
a=[50, 60,70, 80]

b= [40, 60, 70, 80, 90]

print(a>b)           # True

print(a<b)           # False
```

Aliasing List Object:-

When the already available list is given to another reference variable, this process is called aliasing.

```
a=[2,4,6,8]
b=a
print(a)                # [2, 4, 6, 8]
print(b)                # [2, 4, 6, 8]
```

When an element of one reference variable is changed in the aliasing list object, those changes are also reflected in the other reference variables.

```
a=[2,4,6,8]
b=a
print(a)                # [2, 4, 6, 8]
print(b)                # [2, 4, 6, 8]
b[1]=10
print(a)                # [2, 10, 6, 8]
print(b)                # [2, 10, 6, 8]
```

Cloning of List Object:-

When an exact duplicate of a list object is created as an independent list object, it is called cloning.

```
x=[2,4,6,8]
y=x[:]
print(x)                    # [2, 4, 6, 8]
print(y)                    # [2, 4, 6, 8]
y[1]=10
print(y)                    # [2, 10, 6, 8]
print(x)                    # [2, 4, 6, 8]
```

Nested List:-

Sometimes we can take one list inside another list. This type is called nested list.

```
a=[10,20,[30,40],50]

print(a)

print(a[0])

print(a[1])

print(a[2])

print(a[2][0])

print(a[2][1])
```

List Functions & Methods:-

Python provides a variety of methods to operate the list.

Syntax:- **<listObject>.<method name>()**

1. len():-

This function is used to find the length of the list.

```
a=[10,20,50]
print(len(a))          # 3
```

2. count():-

This method is used to know how many times a value appears in a list.

```
a=[10,20,50,10,30,40,50]
print(a.count(10))     # 2
```

3. append():-

This method is used to add an item to the last of a list. By using loop, multiple elements can be added together through append.

```
a=[10,15]
a.append(20)
print(a)                # [10, 15, 20]
```

multiple element add

```
a=[]
for i in range(5):
    b=input("Enter element ")
    a.append(b)
print(a)
```

4. extend():-

This method is used to add another list to the end of the already created list.

Or

This method is used to add multiple elements to the last of the list at the same time.

```
a=[10,20,30]
b=[50,60,70]
a.extend(b)
print(a)                    # [10, 20, 30, 50, 60, 70]
```

Note:- If the iterable object is a string then each character is added individually.

```
a=[10,20,30]
a.extend('word')
print(a)                    # [10, 20, 30, 'w', 'o', 'r', 'd']
```

Note:- append() and extend() method appends only the last element.

5. insert():-

This method is used to add the element at any index number. Whatever remaining element is shifted to the right side.

Syntax:- list.insert(position, element)

```
a=[10,20,30]
a.insert(1,50)
print(a)                    # [10, 50, 20, 30]
```

6. pop():-

This method is used to remove the item from the list at a given position number. If index number is not given then the last item is removed from the list.

Syntax:- pop(index number)

1. No Index specified

```
a=[10,20,30]
a.pop()
print(a)                    # [10, 20]
```

2. Index specified

```
a=[10,20,30]
a.pop(1)
print(a)                    # [10, 30]
```

7. remove():-

This method is used to remove an element from the list. If an item appears multiple times in the list, the first item is removed.

Syntax:- list.remove(element)

```
a=[10,20,30]
a.remove(10)
print(a)                                    # [20, 30]
```

Note: remove method removes only one item. That too the previous item

```
a=[10,20,30,10,50,20,60]
a.remove(10)
print(a)                                    # [20, 30, 10, 50, 20, 60]
```

8. del Statement:-

del statement is used to remove specific items from the list. Or it is used to remove all the items identified by the slice operator.

```
a=[10,20,30]
del a[1]
print(a)                                    # [10, 30]
a=[10,20,30,40,50,30]
del a[1:4]
print(a)                                    # [10, 50, 30]
a=[10,20,30,40,50,30]
del a[:]
print(a)                                    # [ ]
a=[10,20,30,40,50,30]
del a                                       # deleting list
```

9. reverse():-

This method is used to reverse the order of elements in the list.

```
a=[10,20,30]
a.reverse()
print(a)                    # [30, 20, 10]
```

10. sort():-

This method is used to sort items in the list. Through sort, you can arrange the list in ascending order or descending order.

1. ascending order:

```
a=[50,20,30,15,65,79,35]
a.sort()
print(a)                    # [15, 20, 30, 35, 50, 65, 79]
```

2. descending order:

```
a=[50,20,30,15,65,79,35]
a.sort(reverse=True)
print(a)                    # [79, 65, 50, 35, 30, 20, 15]
```

3. sort Not copy in another variable

```
a=[50,20,30,15,65,79,35]
b=a.sort()
print(b)                    # None
```

11. clear():-

This method is used to remove all the elements from the list. This is equivalent to del a[:] .

```
a=[50,20,30,15,65,79,35]
a.clear()
print(a)                    # [ ]
```

12. max() & min():-

This method is used to return the largest and lowest item from the list.

```
a=[50,20,30,15,65,79,35]
print(max(a))               # 79
a=[50,20,30,15,65,79,35]
print(min(a))               # 15
```

13. index():-

This method is used to find the index number of an object/element.

```
a=[10,30,20]
b=a.index(30)
print(b)                    # 1
```

14. sum():-

This method is used to add all the elements of the list.

```
a=[10,20,30]
print(sum(a))               # 60
```

Tuple:-

Tuple is a data structure of Python, which once defined cannot be changed. Tuples are a type of sequence but they are immutable like Strings. In Python we use tuple to store multiple data elements. The way List is specified through Square Brackets and Dictionary through Curly Braces. In the same way, Parenthesis is used to specify Tuples, and in Tuples we can store any type of data, can do nesting of any type of data.

```
newtuple= (“grapes”, “mango”, “cherry”)
print(newtuple)              #  (“grapes”, “mango”, “cherry”)


t1 = (“apple”, “banana”, “cherry”)
t2 = (1, 5, 7, 9, 3)
t3 = (True, False, False)
print(t1)                    # (‘apple’, ‘banana’, ‘cherry’)
print(t2)                    # (1, 5, 7, 9, 3)
print(t3)                    # (True, False, False)
```

Difference between List & Tuple:-

Sr. No.	List	Tuple
1	The list is placed inside square brackets [ ]	Tuple is placed inside parentheses ( )
2	List is mutable	Tuple is immutable
3	List is called read and write	Tuple is called read only
4	List is mutable in which individual item or entire item can be replaced.	Tuple is immutable like string
5	List uses heterogeneous datatype	Tuple holds both homogeneous & heterogeneous datatypes.
6	List is slower for iterating element	Tuple is faster than list

Properties of Tuple:-

i. To create Tuple object, use ( ) parenthesis.

ii. Tuple follows the insertion order.

iii. Duplicate items are allowed in Tuple

iv. Heterogeneous elements are allowed in tuple

v. Tuple supports both positive & negative indexes.

Note:- To represent elements in a tuple, the (,) separator is used, parenthesis ( ) is optional.

Eg.1. a=50,40,30,20

```
print(a)                    # (50, 40, 30, 20)
print(type(a))              # <class 'tuple'>
```

Eg. 2. a=50

```
print(a)                    # 50
print(type(a))              # <class 'int'>
```

Eg. 3. a=50,

```
print(a)                    # (50,)
print(type(a))              # <class 'tuple'>
```

Eg. 4. a=(50)

```
print(a)                    # 50
print(type(a))              # <class 'int'>
```

Eg. 5. a=(50,)

```
print(a)                    # (50,)
print(type(a))              # <class 'tuple'>
```

creating tuple

i. Empty tuple:-

```
a=()
print(a)                    # ( )
print(type(a))              # <class 'tuple'>
```

ii. Create Single value tuple:-

```
 a=5,
print(a)                    # (5,)
print(type(a))              # <class 'tuple'>
```

iii. Create Multi value tuple:-

```
a=5,10,15,20,25
print(a)                    # (5, 10, 15, 20, 25)
print(type(a))              # <class 'tuple'>
```

iv. Using Tuple function:-

```
a=[5,10,15,20,25]
print(a)                    # [5, 10, 15, 20, 25]
print(type(a))              # <class 'list'>
b=tuple(a)
print(b)                    # (5, 10, 15, 20, 25)
print(type(b))              # <class 'tuple'>
```

Accessing Elements of Tuples:-

Tuple से element को access करने के लिए 2 तरीके का use किया जाता है-

Two methods are used to access element from tuple-

i. By using index number

ii. By using slice operator

i. By using index number:-

a	**10**	**20**	**"apple"**	**30**	**40**	**"mango**
Postive index	0	1	2	3	4	5
Negative index	-6	-5	-4	-3	-2	-1

```
print(a[1])          # 20
print(a[5])          # mango
print(st[-1])        # mango
print(st[-5])        #20
```

ii. By using slice operator:-

Syntax:- [start:stop:step]

a	10	20	"apple"	30	40	"mango
Postive index	0	1	2	3	4	5
Negative index	-11	-10	-9	-8	-7	-6

*

```
a=(10,20,"apple",30,40,"mango")
print(a[1:5])                    #  (20, 'apple', 30, 40)

a=(10,20,"apple",30,40,"mango")
print(a[::2])                    # (10, 'apple', 40)

a=(10,20,"apple",30,40,"mango")
print(a[::-1])                   # ('mango', 40, 30, 'apple', 20, 10)
```

Accessing Nested Tuple Items:-

```
a=((10,20,30),[5,8,7],"Hello")
print(a[0][0])                    # 10
print(a[1][0])                    # 5
print(a[2][0])                    # H
```

Membership Test in Tuples:-

```
a=(10,20,30,40)
print(10 in a)                    # True
```

Common tuple operations/mathematical operator for tuple

i. Tuple Addition:- The concatenate operator (+) is used to join two tuples together.

```
a=(10,20,30,40)
b=(2,4,6,8)
c=a+b
print(c)                    # (10, 20, 30, 40, 2, 4, 6, 8)
```

```
a=(10,20,30,40)
c=a+(5,)
print(c)                              # (10, 20, 30, 40, 5)
```

ii. Tuple Multiplication:- Multiplication operator (*) is used to repeat tuple elements.

```
a=(10,20,30)
c=a*2
print(c)        # (10, 20, 30, 10, 20, 30)
```

Tuple Functions:-

The following functions are used in Tuple.

1. cmp()
2. count()
3. len()
4. max() and min()
5. sorted()
6. index()
7. sum()
8. tuple()

cmp( ):-

It is not found in Python 3. In Python 2, it is used to compare tuples. In this, if both the tuples are equal then its result is 0. If the first tuple is bigger than the second tuple then its result is 1, otherwise -1.

```
tup1=(11,22,33)

tup2=(50,60,70)

tup3=(11,22,33)


cmp(tup1,tup2)          # -1

cmp(tup1,tup3)          # 0

cmp(tup2,tup1)          # 1
```

count():-

This method is used to find out how many times a value occurs in a tuple.

```
a=(1,2,2,3,4,5,5,5,5)

print(a.count(2))                    # 2
```

len():-

This function is used to find the length of a tuple.

```
a=(1,2,2,3,4,5,5,5,5)
print(len(a))                    # 9
```

max() & min():-

This method is used to return the largest and lowest item from the tuple.

```
a=(1,2,2,3,4,5,5,5,5)
print(max(a))                    # 5

a=(1,2,2,3,4,5,5,5,5)
print(min(a))                    # 1
```

sorted():-

This method is used to sort the items in a tuple. Through this the tuple can be sorted in ascending order.

```
a=(20,30,40,10,50,80,60,70)
print(sorted(a))                 # [10, 20, 30, 40, 50, 60, 70, 80]
```

index() :-

This method is used to find the index number of an object/element.

```
a=(20,30,40,10,50,80,60,70)
print(a.index(10))                    # 3
```

sum():-

This method is used to add all the elements of a tuple.

```
a=(20,30,40,10)
print(sum(a))                    # 100
```

tuple():-

It is used to create tuples from different types of values.

* a=("akhilesh")

```
print(tuple(a))          #  ('a', 'k', 'h', 'i', 'l', 'e', 's', 'h')
```

* a=([2,4,6,8])

```
print(tuple(a))                  # (2, 4, 6, 8)
```

Packing and Unpacking Tuples:-

When we create a tuple, we normally assign values to it. This is called "packing" a tuple.

```
name = ("aman", "raman", "atul")
print(name)                                  # ('aman', 'raman', 'atul')
```

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

```
name = ("aman", "raman", "atul")
(a, b, c) = name
print(a)                    # aman
print(b)                    # raman
print(c)                    # atul
```

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list.

```
name = ("aman", "raman", "atul", "vimal", "vishal", "Anoop")
(a, b, *c) = name
print(a)                # aman
print(b)                # raman
print(c)                # ['atul', 'vimal', 'vishal', 'Anoop']
```

Dictionary:-

Dictionary in Python is one of those 4 built in data types (list, tuple, set, dictionary) which is used to store data collection or can be said to store multiple values. Dictionary in Python It is used to store data of **key: value** type.

Properties of dictionary:-

1. Elements in dictionary are in pairs of key and value.
2. In dictionary, key and value are separated by colon (:) while the pair of key and value is separated by comma (,).
3. Create mutable object through dictionary
4. Dictionary allows heterogeneous objects in key and value.
5. Duplicate keys are not allowed in Dictionary. But duplicate value is allowed.
6. To access a value from the dictionary, the key of that value is used.
7. { } bracket is used to create dictionary object.

Important Note:-

1. Dict is a class
2. Dict is mutable
3. Dict is not a sequence
4. Dict is iterable
5. Indexing is not applicable to dict object
6. Dict cannot have duplicate keys (not data value)
7. Dict element are pair of key-value and data-value
8. Slicing operator is not applicable

Example:-

```
dict1 = {"name1":"Rakesh", "name2":"Ramesh", "name3":"Kamalesh"}

print(dict1)          # {'name1': 'Rakesh', 'name2': 'Ramesh', 'name3': 'Kamalesh'}
```

Creating Dictionary

Syntax:- **d={key1:value1,key2:value2,.....}**

Empty dictionary:-

```
d={}
print(d)                    # {}
print(type(d))              # <class 'dict'>
```

Adding element in empty dictionary:-

```
d={}
d[10]='akhilesh'
d[11]='monu'
d[12]='sonu'
d[13]='ankit'
print(d)                    # {10: 'akhilesh', 11: 'monu', 12: 'sonu', 13: 'ankit'}
```

Note:- Value can also be assigned directly to the dictionary.

```
a={50:'Aman', 51:'Raman', 52:'Suman'}
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Suman'}
print(type(a))              #<class 'dict'>
```

Accessing Value from Dictionary

To access the value in List and Tuple, their 'index' is used. Similarly, if we want to access the values of the dictionary, then 'keys' are used in their square brackets ([]).

```
d={10:'Atul',11:'Vimal',12:'vishal',13:'Vikash'}
print(d[10])                    # Atul
print(d[11])                    # Vimal
print(d[12])                    # vishal
print(d[13])                    # Vikash
```

Note:- To access the element from dictionary, you can use get keyword.

```
a={50:'Aman', 51:'Raman', 52:'Suman'}
print(a)                        # {50: 'Aman', 51: 'Raman', 52: 'Suman'}
print(a.get(51))                #Raman
```

Note:- If the specified key is not available, 'keyError' message appears.

Traversing a Dictionary:-

Dictionary से प्रत्येक value को access करने के लिए for in loop का use कर सकते हैं।

You can use for in loop to access each value from the dictionary

```
d={10:'Atul',11:'Vimal',12:'vishal',13:'Vikash'}

for x in d:

   print(d[x])
```

Output:

```
Atul

Vimal

vishal

Vikash
```

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

```
d={10:'Atul',11:'Vimal',12:'vishal',13:'Vikash'}
for x in d:
    print(x)
Output:
10
11
12
13
```

```
d={10:'Atul',11:'Vimal',12:'vishal',13:'Vikash'}
for x in d:
    print(x,":",d[x])
Output:
10 : Atul
11 : Vimal
12 : vishal
13 : Vikash
```

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

Add Dictionary Element in Python

To add an element to the dictionary, the unique key name and its value are given in square brackets ([]).

```
a={50:'Aman', 51:'Raman', 52:'Suman'}
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Suman'}
print(a[51])                # Raman
print(type(a))              <class 'dict'>
```

Deleting (Removing) Element from Dictionary:-

To delete an element from the dictionary, a unique key is given in square brackets ([]) along with the del keyword.

```
a={50:'Aman', 51:'Raman', 52:'Suman'}
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Suman'}
del a[51]
print(a)                    # {50: 'Aman', 52: 'Suman'}
```

Deleting Dictionary:-

del keyword is used to delete the dictionary.

```
a={50:'Aman', 51:'Raman', 52:'Suman'}
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Suman'}
del a
print(a)                    # NameError: name 'a' is not defined
```

Changing (updating) Element of Dictionary:-

```
a={50:'Aman', 51:'Raman', 52:'Suman'}
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Suman'}
a[52]="Ankit"
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Ankit'}
```

Appending Element in Dictionary:-

```
a={50:'Aman', 51:'Raman', 52:'Suman'}
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Suman'}
a[53]="Ankit"
print(a)                    # {50: 'Aman', 51: 'Raman', 52: 'Suman', 53: 'Ankit'}
```

Functions of Dictionary:-

Sr. No.	Function	Sr. No.	Function
1.	dict()	10.	values()
2.	len()	11.	items()
3.	clear()	12.	updates()
4.	copy()	13.	fromkeys()
5.	get()	14.	zip()
6.	pop()	15.	has_key()
7.	popitem()	16.	sorted()
8.	setdefault()		
9.	keys()		

Dict():-

dict() function is used to create a dictionary.

```
dict1 = dict({"one":'Ramesh', "two":'suresh', "three":'rajesh'})
print(dict1)                    # {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
```

len():-

The len() function is used to find the length of the dictionary.

```
dict = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(len(dict))                # 3
```

clear():-

clear() method is used to clear the dictionary.

```
dict={"one":'Ramesh', "two":'suresh', "three":'rajesh'}
dict.clear()
print(dict)                     # {}
```

copy():-

copy() method is used to copy the dictionary.

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
dict2 = dict1.copy()
print(dict1)                    # {'one': 'Ramesh', 'two': 'suresh', 'three': 'rajesh'}
print(dict2)                    # {'one': 'Ramesh', 'two': 'suresh', 'three': 'rajesh'}
```

get():-

get() method is used to get the value of the given key.

Syntax:- dict.get(key, defaultVal)

* dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}

print(dict1.get("two")) # suresh

* dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}

print(dict1.get("four",'mahesh')) # mahesh

Note:- Here the key named 'four'; is not present in the dictionary and default value is set to 'mahesh' if the given key; get() method, then the value set on the parameter is returned.

* dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}

print(dict1.get("four")) # None

Note:- If the given key; get() method returns 'None' if it is not found and a default value is not set.

pop():-

In pop() method, the given key is removed and its value is returned. If the given key is not found then the set default value is returned.

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print("Before Removing item :")
print(dict1)        # {'one': 'Ramesh', 'two': 'suresh', 'three': 'rajesh'}
dict1.pop("one")
print("After Removing item :")
print(dict1)                # {'two': 'suresh', 'three': 'rajesh'}
```

```
*       dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(dict1)                        # {'one': 'Ramesh', 'two': 'suresh', 'three': 'rajesh'}
print(dict1.pop("four", "mahesh"))                      # Mahesh
```

```
*       dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(dict1)                        # {'one': 'Ramesh', 'two': 'suresh', 'three': 'rajesh'}
print(dict1.pop("four"))            # KeyError: 'four'
```

Note:- If the key is not found and the default value is not set then 'keyError' exception occurs.

popitem():-

popitem() method removes the last item of the given dictionary and returns (key, value)

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print("Before Removing last item : ")
print(dict1)                    # {'one': 'Ramesh', 'two': 'suresh', 'three': 'rajesh'}
print("Removed item :",dict1.popitem())          # Removed item : ('three', 'rajesh')
print("After Removing last item : ")
print(dict1)                    # {'one': 'Ramesh', 'two': 'suresh'}
```

setdefault():-

If the key given in setdefault() method exists then its value is returned. If there is no key then the set default value is returned.

Syntax:- dict.setdefault(key, default)

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(dict1.setdefault("one"))                  # Ramesh
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(dict1.setdefault("four"))                 # None
```

Note:- If the key is not in the dictionary and no default parameter is used, 'None' is returned.

keys():-

Only keys of dictionary can be returned from keys() method.

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(dict1.keys())                 # dict_keys(['one', 'two', 'three'])
```

values():-

With the values() method, only the values of the keys of the dictionary can be returned.

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(dict1.values())               # dict_values(['Ramesh', 'suresh', 'rajesh'])
```

items():-

items() method returns all key-value pairs of the dictionary in a tuple (representing list of tuple)

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print(dict1.items())          # dict_items([('one', 'Ramesh'), ('two', 'suresh'), ('three', 'rajesh')])
```

update():-

update() method is used to update the old dictionary.

```
dict1 = {"one":'Ramesh', "two":'suresh', "three":'rajesh'}
print("Before Updating or Adding Keys and Values :")
print(dict1)          # {'one': 'Ramesh', 'two': 'suresh', 'three': 'rajesh'}
dict2 = {"one":'akhilesh', "four":'rakesh', "five":'mahesh'}
print("After Updating or Adding Keys and Values:")
dict1.update(dict2)
print(dict1)          # {'one': 'akhilesh', 'two': 'suresh', 'three': 'rajesh', 'four': 'rakesh', 'five': 'mahesh'}
```

fromkeys():-

fromkeys() method returns the items of the given sequence as keys of the dictionary.

Syntax:- dict.fromkeys(seq, val)

Parameter :

seq : Here sequence(list, tuple, string) is given. These are 'keys' for the new dictionary to be returned.

val : Optional. These are the values of the keys. If not given then 'None' is returned.

```
list = ["H", "e", "l", "l", "o"]
print(dict.fromkeys(list))          # {'H': None, 'e': None, 'l': None, 'o': None}
```

Note:- If second parameter 'seq' is not given then 'None' is returned.

```
list = ["H", "e", "l", "l", "o"]
print(dict.fromkeys(list, "uma"))          # {'H': 'uma', 'e': 'uma', 'l': 'uma', 'o': 'uma'}
```

zip():-

zip() method is used to represent two lists as a key and value.

```
reg=[1491514,1491515,1491516]
name=['ankit','sunil','mahesh']
st=dict(zip(reg,name))
print(st)                    # {1491514: 'ankit', 1491515: 'sunil', 1491516: 'mahesh'}
```

has_key():-

The given key is used using the has_key() method; Whether it is on the dictionary or not is returned as a boolean value if the key given in the has_key() method; If it is in the dictionary then 'True' is returned and if it is not then 'False' is returned.

Note:- has_key() method; Does not support Python 3.x versions

sorted():-

Sorts the given dictionary and returns its keys.

```
dict1 = { 'num6': 6, 'num3': 3, 'num2': 2, 'num4': 4, 'num1': 1, 'num5': 5}
sortedDict = sorted(dict1)
print(sortedDict)            #['num1', 'num2', 'num3', 'num4', 'num5', 'num6']
```

Nested Dictionary:-

Syntax:- {key:value, key:value, key:{key:value, key:value.....}}

```
student={'reg':14591,'name':'ankit',2:{'reg':14590,'name':'sunil'}}

print(student)     # {'reg': 14591, 'name': 'ankit', 2: {'reg': 14590, 'name': 'sunil'}}

print(student['reg'])                    # 14591

print(student['name'])                   # ankit

print(student[2])                        # {'reg': 14590, 'name': 'sunil'}

print(student[2]['reg'])                 # 14590

print(student[2]['name'])                #sunil
```

Sequence Data Types:-

Sequence is a collection of items indexed by a positive integer. It is a combination of mutable and immutable variables. There are three types of sequences available in Python

1. String
2. List
3. Tuples

String:-

A sequence of characters is called a string. The string is represented in single quotes (' ') or double quotes (" ").

Note:- 1. To represent string data in a single line, you can use ' ' or " ".

2. """ """ is used to represent string data in multi lines.

Defining String Data:-

st= "ak it solution" print(st) print(type(st))	num= "10" print(num) print(type(num))	num1= "20.5" print(num1) print(type(num1))

Accessing character of string:-

String character is accessed in two ways-

1. By using index number
2. By using slice operator

1. **By using index number:-**

st	P	Y	T	H	O	N
Postive index	0	1	2	3	4	5
Negative index	-6	-5	-4	-3	-2	-1

Note:- Python supports both positive & negative indexing.

* **Positive index means left to right which is also called forward direction.**

```
print(st[0])        # P
print(st[3])        # H
```

* **Negative index means right to left.**

```
print(st[-6])       # P
print(st[-5])       #Y
```

2. By using slice operator:-

Syntax:- [start:stop:step]

st	H	E	L	L	O		W	O	R	L	D
Postive index	0	1	2	3	4	5	6	7	8	9	10
Negative index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Note: 1. [ ] It is known as slice operator and is used to access the sub-string of a string.

2. [:] This is known as the range slice operator.

*

```
a="hellow world"
print(a[0:5:1])          #      hello
```

*

```
a="hellow world"
print(a[0:5])        #      hello
```

*

```
a="hellow world"
print(a[:5])             #      hello
```

*

```
a="hellow world"
print(a[5:])             # w world
```

*

```
a="hellow world"
print(a[0::2])           # hlo ol
```

String Operators:-

4 operators work in String-

1. Concatenation (+) Operator

2. Repetition (*) operator

3. Membership (in & not in) operator

4. Comparison (>,<, ==, !=) operator

Concatenation (+) Operator:-

This operator is used to join two strings.

st1= "ram" st2= "sita" print(st1+st2) #ramsita

Note:- This operator is used to combine operands of the same type.

Repetition (*) operator:-

This operator is used to multiply the string.

a= "ram" print(a*2) # ramram

a="ram" print(a*'2') #error

String Function:-

String has some built-in functions which are as follows-

len()	capitalize()	find()	isalnum()	isalpha()	isdigit()	lower()
islower()	isupper()	upper()	istrip()	rstrip()	isspace()	istitle()
replace(old,new)	split()	join()	swapcase()	count()	str()	index()

1. len():- This function is used to find the number of characters in a string.

Example:- a= "apple" print(len(a)) # 5

2. capitalize():- This function is used to capitalize the first character of the string.

Example:- a="ak it solution" print(a.capitalize()) # Ak it solution

3. find():-This function is used to find any sub string from a string. This function assigns the index number to the found string. If nothing is found in the string then it returns —

syntax:- find(find_string, starting position, end position)

a="ak it solution" print(a.find("ti")) # 10

4. isalnum():-This function is used to check letters and digits in the string. Its result comes in true & false.

a='akhilesh123' print(a.isalnum()) # True

5. isalpha():-This function is used only to check letters.

a='akhilesh' print(a.isalpha()) # True

String Function

6. isdigit():-This function is used only to check the digit.

a='123' print(a.isdigit()) # True

7. lower():-This function is used to convert upper case string to lower case string.

a='AKHILESH' print(a.lower()) # akhilesh

8. islower():-This function is used to check lower case string.

a='AKHILESH' print(a.islower()) # False

9. isupper():-This function is used to check upper case string.

a='AKHILESH' print(a.isupper()) # True

10. upper():- This function is used to convert lower case string into upper case string.

a='akhilesh' print(a.upper()) # AKHILESH

11. lstrip():-This function is used to remove the character from the left side (with space) from a string.

a='AKHILESH' print(a.lstrip('A')) # KHILESH

12. rstrip():-This function is used to remove the character from the right side of a string.

a='AKHILESH' print(a.rstrip('H')) #AKHILES

13. isspace():- This function is used to check space. If there is even one character in the string then its result will be false.

a=' ' print(a.isspace()) # True

String Function

14. istitle():- This function is used to check title case in a string.

```
a='Ak It Solution'                print(a.istitle())          # True
```

15. replace(old,new):- This function is used to replace the substring of a string.

```
a='Ak It Solution'      print(a.replace('Ak','uma'))    # uma It Solution
```

16. split():- This function is used to split a string and return it in a substring list.

```
a='Ak It Solution'      print(a.split())        # ['Ak', 'It', 'Solution']
```

17. join():- This function is used to combine different words into a single string.

```
a=('ram','ramesh','suresh')
b="/"
print(b.join(a))                        #ram/ramesh/suresh
```

18. swapcase():-This function is used to convert from lowercase letter to uppercase letter and from uppercase letter to lowercase letter.

```
a="akhilesh"          print(a.swapcase())              # AKHILESH
```

19. count():-This function is used to count the number of substrings in a string.

```
a="ak it solution"      print(a.count('ak'))          # 1
```

20. index():-The index() method finds the first occurrence of the specified value. The index() method raises an exception if the value is not found.

Syntax:- *string*.index(*value, start, end*)

```
a="akhilesh"              b=a.index('h')          print(b)          # 2
```

Strings are Immutable:-

String is immutable, which means that the content of the string variable cannot be changed after it is created.

```
a="apple"

a[0]='c'

print(a)                    #TypeError:
```

Note:- You can change the character by using the following method-

```
a="apple"

b='c'+a[1:]

print(b)            # cpple
```

```
#Traversal String using for loop
st="PYTHON"
for i in st:
    print(i)
```

```
#Traversal String using while loop
st="PYTHON"
i=0
while i<len(st):
    print(st[i])
    i=i+1
```

Traversing a String:-

Traversing a string means accessing all the elements of the string one after the other using subscripts or indexes. A string can be traversed character by character using a for loop or while loop.

```
name="abhishek"
letter=name[1]
print(letter)
```

```
name="abhishek"
letter=name[1.5]
print(letter)
```

```
# Iterate Python String
Str = 'SCHOOL'
For word in str:
 Print ( "Letters are:",word)
```

```
# Iterate Python String
Str = "SCHOOL"
For word in range (len (str)):
Print ("Letters at index {0} is = {1
}". Format(word,str[word]))
```

```
Print ('black'+'white')
Print (('h') in 'elephant')
```

```
Print ("HELLO" *3 )
Print (( 'i' ) in ' elephant')
```

```
>>> word = "language"
>>> word[1:3]
>>>word[5:]
>>>word[-1]
>>>word[-1:]
>>>word[::-1]
```

```
>>> s="ten"
>>> rev_str =s [len(s)::-1]
>>> print(rev_str)
```

```
a = ['apple','banana','pomegranate']
for x in a:
    print(x, len(x))
```

```
Str1 ='Hello World'
   len(str1)
```

```
Str1 = 'Apple is
Healthy . Apple is also
Ggod for health.'
Print()
Print(str1.
Count("Apple"))
Print()
```

```
Str = "elephant"
Count = 0
For i in str:
    Count = count+1
Print(count)
```

```
Str= " work is worship"
Print ('work' in str.split())
```

Set Data Type:-

In Python, set data type contains a set of elements. Every element in Python's set is new and these new elements cannot be changed. The elements given in a set are immutable. But the elements of the set can be added or removed with any method. Python Set is an unordered collection. { } is used to represent elements of Set type. And each element is separated by (,).

Note:-

1. Set is a class
2. Set is mutable
3. Set is not a sequence
4. Set is iterable
5. Indexing is not applicable to set object
6. Set cannot have duplicate values
7. Set does not guarantee to store values in the order of insertion
8. Slicing operator is not applicable
9. Concatenation (+) and Repetition (*) operator not supported in set
10. Comparison operator (>, <, <=, <=, ==,!=) is supported in set. (Two set objects are equal if their elements are same, doesn't matter the order of elements.)

Types of Set:-

There are two types of sets:

1. Set
2. Frozenset

How to declare Python Set

- Python Set is declared with elements between { }

```
a = {1, 2, 3}
print(a)                    # {1, 2, 3}
print(type(a))              # <class 'set'>
```

- Set can also be declared using set() function.

```
a = set([1, 2, 3])
print(a)                    # {1, 2, 3}
print(type(a))              # <class 'set'>
```

Creating Set

The elements of a set are written inside curly braces ({}) and each element is separated with a comma (,).

```
set1 = {"H", "e", "l", "l", "o"}
#elements in curly braces
set2 = set("Hello")
#passing string to set function
set3 = set(["H", "e", "l", "l", "o"])
#passing list to set function
set4 = set(("H", "e", "l", "l", "o"))
#passing tuple to set function
set5 = set()
#empty set
```

Mixed Data Type Set

- set does not support mutable items. This is a mutable item in the set [1, 2].

```
set = {1, "H", 2.6, [1,2]}
```

Output

```
set = {1, "H", 2.6, [1,2]}
TypeError: unhashable type: 'list'
```

- Tuple can be used in set. Because both set and tuple are immutable.

```
set = {1, "H", 2.6, (1,2)}
```

Iterating Over a Set

A 'for_in' loop is used to iterate through each element and print it.

For Example,

```
set = {1, 2, 5, 3, 4}
for i in set:
    print(i)
```

Note:-

- **Empty Set:-**

The elements of a set are written in curly braces({}) but if empty curly braces({}) are used then it becomes 'dictionary' type. For Empty Set empty set() function has to be given.

```
set1 = {}
print(type(set1))                    # <class 'dict'>
set2 = set()
print(type(set2))                    # <class 'set'>
```

- Duplicate data is not allowed in Set data type, it removes other duplicate data.

```
s1=set([1,4,3,2,5,1,4,3,5])
print(s1)                    # {1, 2, 3,4, 5}
```

- Insertion order is not preserved in Set data type but sorting of elements can be done.

```
s1={10,20,30,40,50}
print(s1)                    # {50, 20, 40, 10, 30}
```

- Indexing and slicing is not allowed in Set data type.

```
s1={10,20,30,40,50}
print(s1[2])
print(s2[1:4])               # TypeError: 'set' object is not subscriptable
```

- Set data type allow in heterogeneous element

```
s1={10,15.02,'ak it solution'}
print(s1)                    # {10, 'ak it solution', 15.02}
```

Set Functions in Python

Set Function	Description
all()	If all the elements of the sequence are True then it returns True.
any()	If one or all elements of the sequence are True then it returns True.
len()	Returns the length of the set.
max()	Returns the max value from the set.
min()	Returns the min value from the set.
set()	Creates a set or converts a sequence into a set in a different way.
sorted()	Sorts and returns the given sequence.
sum()	Adds the elements of a given sequence or collection and returns their sum.

1. s1={10,15.02,'ak it solution'}

```
print(all(s1))                                  # True
```

2. s1={10,15.02,'ak it solution',False}

```
print(any(s1))                                  # True
```

3. s1={10,15.02,'ak it solution'}

```
print(len(s1))                                  # 3
```

4. s1={10,50,60,40,20,30}

```
print(max(s1))                                  # 60
```

5. s1={10,50,60,40,20,30}

```
print(min(s1))                                  # 10
```

6. s1=(10,50,60,40,20,30)

```
print(set(s1))                          # {40, 10, 50, 20, 60, 30}
```

7. s1={10,50,60,40,20,30}

```
print(sorted(s1))                       # [10, 20, 30, 40, 50, 60]
```

8. s1={10,50,60}

```
print(sum(s1))                                  # 120
```

Set Method in Python

Sr. No	Set Method	Description
1	add()	This method is used to add an element to the set.
2	clear()	Clears the given set.
3	copy()	Copies the given set.
4	difference()	Updates and returns the difference of the two given sets.
5	discard()	Removes the given element from the given set.
6	intersection()	Returns the identical element(s) from the given set.
7	issubset()	Returns True if one set is a subset of another set.
8	issuperset()	If given main set; Returns 'True' if it is a superset of the given set.
9	pop()	Removes and returns a random element.
10	remove()	This method is used to remove an element from the set.
11	union()	The set is returned by collecting the given sets.
12	update()	This method is used to update the set.

1. a={10,20,30} a.add(50) print(a) **Output** {10, 20, 50, 30}	a={10,20,30} a.add((10,20)) print(a) **Output** {10, 20, 30, (10, 20)}	2. a={10,20,30} a.clear() print(a) **Output** set()
3. a={10,20,30} b=a.copy() print(a) print(b) **Output** {10, 20, 30} {10, 20, 30}	4. a={10,20,30} b={10,30,60} print(a.difference(b)) **Output** {20}	5. a={10,20,30} a.discard(10) print(a) **Output** {20, 30}
6. a={10,20,30} b={10,30,60} print(a.intersection(b)) **Output** {10, 30}	7. a={10,30} b={10,30,60} print(a.issubset(b)) **Output** True	8. a={10,30} b={10,30,60} print(b.issuperset(a)) **Output** True
9. a={10,20,30,40,50} b=a.pop() print(a) print(b) **Output** {20, 40, 10, 30}	10. a={10,20,30,40,50} a.remove(10) print(a) **Output** {50, 20, 40, 30}	11. x={10,20,30} y={40,50,60} z=x.union(y) print(z) **Output** {50, 20, 40, 10, 60, 30}
12. a={10,20,30} a.update([50]) print(a) **Output** {10, 20, 50, 30}	a={10,20,30} a.update((40,50,60)) print(a) **Output** {40, 10, 50, 20, 60, 30}	a={10,20,30} a.update(50) print(a) **Output** TypeError: 'int' object is not iterable