TUPLE- DATA Type :

- Tuple is used to represent a set of homogeneous or heterogeneous elements into a single entity.
- Tuple objects are immutable that means once if we create a tuple object later we cannot modify those tuple elements.
- All elements are separated by commas (,) and enclosed by parentheses. Parentheses are optional. ()
- Tuple allows duplicate elements.
- Every element in the tuple has its own index number
- Tuple supports both forward indexing and also backward indexing, forward indexing starts from 0 and backward indexing starts from -1.
- If we take **only one element** in the tuple then we should use **comma (,)** after that single element.

  t = (10,) --->> tuple type

  t1 = (10) --->> int type
- Tuples can be used as keys to the dictionary.
- We can create a tuple in different ways, like with tuple(), with () or without () also.
- The main difference between lists and tuples is- Lists are enclosed in square brackets like [] and their elements and size can be changed, while tuples are enclosed in parentheses like () and their elements and size cannot be updated.

Creating a tuple with tuple() :

```
>>> tup  =  tuple([10,20,30,True,'Python'])
>>> print(tup)                    (10, 20, 30, True, 'Python')
>>> type(tup)                     <class 'tuple'>
>>> id(tup)                       52059760
```

Creating an empty tuple:

Example:

```
>>> tup  =  ()                    #creating empty tuple
>>> print(tup)                    ()
>>> type(tup)                     <class 'tuple'>
>>> id(tup)                              23134256
```

Creating a tuple with ()

Example:

```
>>> tup2  =  (10,20,30,40,50)            #creating homogeneous tuple
>>> print(tup2)                          (10, 20, 30, 40, 50)
>>> type(tup2)                           <class 'tuple'>
>>> id(tup2)                             63484864
```

Creating a tuple without ()

Example:

```
>>> tup = 10,20,True,'Py'                #creating tuple without parenthesis
>>> print(tup)                           (10, 20, True, 'Py')
>>> type(tup)                            <class 'tuple'>
>>> id(tup)                              67086688
```

Creating a tuple with heterogeneous elements

Example:

```
>>> tup1 = (10,20,30,True,"Python",10.5,3+5j)      #creating heterogeneous tuple
>>> print(tup1)
(10, 20, 30, True, 'Python', 10.5, (3+5j))
```

Creating a tuple with homogeneous elements

Example:

```
>>> t = (10,20,30,40)   # creating homogeneous tuple
>>> print(t)           # (10,20,30,40)
```

NOTE : **tuple with Single value**

- Creating a tuple with a single element is tricky, if we take only one element then the type of that tuple will be based on specified element type.

```
>>> t2 = (1)
>>> t2                   1
>>> type(t2)             <type 'int'>
>>> t2 = (True)
>>> print(t2)            True
>>> type(t2)             <type 'bool'>
>>> t2 = ('a')
>>> print(t2)            'a'
>>> type(t2)             <type 'str'>
```

Solution :

So to solve the above problem we should use comma (,) after the element in the tuple if tuple contains single element.

For example:

```
>>> t2=(1,)
>>> print(t2)           (1,)
>>> type(t2)            <type 'tuple'>
>>> t2=(False,)
>>> print(t2)           (False,)
>>> type(t2)            <type 'tuple'>
>>> t2 = ('a')
>>> print(t2)           ('a',)
>>> type(t2)            <type 'tuple'>
```

Tuple Indexing:

Tuple indexing is nothing but fetching a specific element from the existing tuple by using its index value.

Tuple Slicing:

Tuple slicing is nothing but fetching a sequence of elements from the existing tuple by using their index values.

Example:

```
>>>tup = (10,20,30,True,"Python",10.5,3+5j,10)
>>> print(tup)               # (10, 20, 30, True, 'Python', 10.5, (3+5j), 10)
>>> type(tup)                # <class 'tuple'>

            0     1     2      3        4          5       6      7
tup   =  ( 10,   20,   30,   True,  "Python",    10.5,   3+5j,   10 )
           -8    -7    -6     -5       -4          -3      -2     -1

>>> tup[0:4]                               (10, 20, 30, True)
>>> tup[0:0]                               ()
>>> tup[0:1]                               (10,)
>>> tup[0:5]                               (10, 20, 30, True, 'Python')
>>> tup[3:5]                               (True, 'Python')
>>> tup[2:-2]                              (30, True, 'Python', 10.5)
>>> tup[-5:-2]                              (True, 'Python', 10.5)
>>> tup[-5:]                                (True, 'Python', 10.5, (3+5j), 10)
>>> tup[6:]                             ((3+5j), 10)
```

Tuple concatenation :

- We can concatenate two or more tuples in python.

Example:

```
>>> tup1=(1,2,3,'a',True)    #creating first tuple tup1
>>> print(tup1)              (1, 2, 3, 'a', True)
>>> type(tup)                <class 'tuple'>
>>> tup2=(10,20,False,'b')   #creating second tuple tup2
>> print(tup2)               (10, 20, False, 'b')
>>> type(tup2)               <class 'tuple'>
>>> tup3  =  tup1+tup2           #concatenating tup1 and tup2 as tup3
>>> print(tup3)              (1, 2, 3, 'a', True, 10, 20, False, 'b')
>>> type(tup3)               <class 'tuple'>
```

Tuple multiplication or repetition :

- We can multiply or repeat a tuple n number of times.

```
>>> tup1=(1,2,3,'a',True)
>>> print(tup1)              (1, 2, 3, 'a', True)
>>> type(tup1)               <class 'tuple'>
>>> tup1*3
(1, 2, 3, 'a', True, 1, 2, 3, 'a', True, 1, 2, 3, 'a', True)
```

Tuple Data type Methods :

1. len():

- This function returns no.of elements in the tuple.

```
>>> tup = (1,2,3,4,'a',5.5)
>>> len(tup)                     6
```

2. count():

- This function counts the number of occurences of a specific elements. This function takes exactly one argument like element.

Example:

```
>>> tup = (1,10,20,True,0)
>>> tup.count(1)                 2
>>> tup.count(0)                 1
```

3. index(object, index_value,end_index):

- This function is used to find the index value of specific|given element.
- This function returns by default first occurence of given element index_value.
- It is also accepts the second parameter as index value, it is used for from where you want search the given index. By default index_value starts from zero.

Example:

```
>>> tup=(1,10,20,True,0)
>>> tup.index(0)            4
>>> tup.index(10)                1
>>> tup.index(20)                2
```

4. max():

- This function returns maximum value from the tuple elements.

Example:

```
>>> tup=(1,3,2,55,3,5,23)
>>> max(tup)                     55
```

5. min():

- This function returns minimum value from the tuple elements.

Example:

```
>>> tup=(1,3,2,55,3,5,23)
>>> min(tup)                     1
```

6. sum():

- this function returns sum of all the elements.

Example:

```
>>> tup=[1,9,5,11,2]
>>> sum(tup)                     28
```

7. sorted(object):

- sorted() is going to take the elements from given object and arranging all the elements by default in a assending order.
- after arranging all the elements in assending order then resoult store in a new   variable.
- sorted() method is not doing any changes in a original object and the result store in a new object.
- sorted() method returns result in a list format by defalt.

- if you want to get in tuple format then use tuple() method

Example:

```
>>> tup = (1,3,2,55,3,5,23)
>>> sorted(tup)                 [1, 2, 3, 3, 5, 23, 55]
```

Note: by default this function sorts the data in ascending order. We can also get in descending order by setting True for reverse.

Example:

```
>>> tup=(1,3,2,55,3,5,23)
>>> sorted(tup,reverse=True)              [55, 23, 5, 3, 3, 2, 1]
```

Or

```
>>> t1 = tuple([1,2,3,7,4])
>>> t1                              (1, 2, 3, 7, 4)
>>> t2 = reversed(t1)
>>> tuple(t2)                       (4, 7, 3, 2, 1)
```

8. reversed():

- reversed() is going to take the elements from given object and arranging all the elements by default in a reversing order.
- after arranging all the elements in reversing order then resoult store in a new variable.
- reversed() method is not doing any changes in a original object and the result store in a new object.
- reversed() method returns result in a <reversed object at 0x03EFFC30> format by defalt. Internally elements are reversed.
- if you want to get in tuple format then use tuple() method

```
>>> t  =  (10, 40, 60, 20)
>>> t2  =  reversed(t)
>>> t2
<reversed object at 0x03EFFC30>
>>> tuple(t2)    #  (20, 60, 40, 10)
```

Note:

- tuple object is not supporting both sort() and reverse() and copy() and clear() also.

```
>>> t = (10, 40, 60, 20)
>>> t2 = sort(t)
```

```
NameError: name 'sort' is not defined
>>> t3 = reverse(t)
NameError: name 'reverse' is not defined
>>> t = (10, 40, 60, 20)
>>> id(t)              # 65890704
>>> t2 = t
>>> print(t2)            # (10, 40, 60, 20)
>>> id(t8)              # 65890704
>>> t2 = t.copy()
AttributeError: 'tuple' object has no attribute 'copy'
```

DEL Command :

We cannot delete the elements of existing tuple but we can delete the entire tuple object by using del command.

Example:

```
>>> tup = (10,20,"Python",1.3)
>>> print(tup)                        # (10, 20, 'Python', 1.3)
>>> type(tup)                          # <class 'tuple'>
>>> del tup                            # deleting tuple by using del command.
>>> print(tup)                         # after deleting
NameError: name 'tup' is not defined
```

We can replace the elements of list but not tuple, like

```
>>> lst=[10,20,30,'Py',True]
>>> lst[4]=False                      # it is possible in list
>>> print(lst)                                                  [10, 20, 30, 'Py', False]
>>> tup = (10,20,30,'Py',True)
>>> tup[4]=False                              # it is not possible in tuple
TypeError: 'tuple' object does not support item assignment
```

Tuple packing:

- We can create a tuple by using existing variables, so its called tuple packing.

```
>>> a=10
>>> b=20
>>> c='Python'
>>> d=2+5j
>>> tup=(a,b,c,d)
>>> print(tup)                    (10, 20, 'Python', (2+5j))
>>> type(tup)                     <class 'tuple'>
>>> id(tup)              62673808
```

Tuple Unpacking

- Tuple unpacking allows to extract tuple elements automatically.
- Tuple unpacking is the list of variables on the left has the same number of elements   as the length of the tuple

```
>>> tup=(1,2,3,4)
>>> a,b,c,d=tup  # tuple unpacking
>>> print(a)             1
>>> print(b)             2
>>> print(c)             3
>>> print(d)             4
```

Nested tuple:

- Python supports nested tuple, means a tuple in another tuple.
- Tuple allows list as its element.

Example:

```
>>> t1=(1,'a',True)
>>> print(t1)                  (1, 'a', True)
>>> type(t1)                   <class 'tuple'>
>>> t2=(10,'b',False)
>>> print(t2)                  (10, 'b', False)
>>> type(t2)                   <class 'tuple'>
>>> t3=(t1,100,'Python',t2)    # creating a tuple with existing tuples t1 and t2
>>> print(t3)                  ((1, 'a', True), 100, 'Python', (10, 'b', False))
>>> type(t3)                   <class 'tuple'>
>>> print(t3[0])               (1, 'a', True)
>>> print(t3[1])               100
>>> print(t3[2])               Python
>>> print(t3[3])               (10, 'b', False)
>>> print(t3[3][0])            10
>>> print(t3[3][1])            b
>>> print(t3[3][2])            False
>>> print(t3[0][0])              1
>>> print(t3[0][1])              a
>>> print(t3[0][2])              True
>>> t3[0:2]                      ((1, 'a', True), 100)
>>> t3[2:4]                      ('Python', (10, 'b', False))
>>> t3[-2:4]                     ('Python', (10, 'b', False))
```

Note:

- ❖ We can't modify any element of the above tuples because tuples are immutable.
- ❖ If the tuple contains a list as a element then we can modify the elements of the list as it a mutable object.

Example:

```
>>> tup = (1,2,[10,12,'a'],(100,200,300),3,'Srinivas')
>>> print(tup)             # (1, 2, [10, 12, 'a'], (100, 200, 300), 3, 'Srinivas')
>>> tup[0]          1
>>> tup[1]         2
>>> tup[2]          [10, 12, 'a']
>>> tup[3]          (100, 200, 300)
>>> tup[4]          3
>>> tup[5]          'Srinivas'
>>> tup[0]=50
# trying to replace element 1 with 50, interpreter throws error.
    TypeError: 'tuple' object does not support item assignment
>>> tup[2][0]=50
 # trying to replace element of list 10  with 50, interpreter accepts.
>>> print(tup)             (1, 2, [50, 12, 'a'], (100, 200, 300), 3, 'Srinivas')
```

Converting tuple to list :

```
>>> tup=(1,2,4,9,8)          #creating a tuple
>>> print(tup)          (1, 2, 4, 9, 8)
>>> type(tup)           <class 'tuple'>
>>> lst = list(tup)          # converting tuple to list by using list()
>>> print(lst)               [1, 2, 4, 9, 8]
>>> type(lst)                <class 'list'>
```

Converting list to tuple:

```
>>> lst=[10,20,30,40,'a']      #creating a list
>>> print(lst)                 [10, 20, 30, 40, 'a']
>>> type(lst)                  <class 'list'>
>>> tup=tuple(lst)             #converting list to tuple by using tuple()
>>> print(tup)                 (10, 20, 30, 40, 'a')
>>> type(tup)                  <class 'tuple'>
```

Converting tuple to string:

```
>>> tup=('a','b','c')          #creating tuple
>>> print(tup)                 ('a', 'b', 'c')
>>> type(tup)                  <class 'tuple'>
>>> str1=''.join(tup)          #converting tuple to string by using join method
>>> print(str1)                abc
>>> type(str1)                 <class 'str'>
```

Converting string to tuple:

```
>>> str1="Python Srinivas"          #creating  a string
>>> print(str1)                     Python Srinivas
>>> type(str1)                      <class 'str'>
>>> tup=tuple(str1)                 #converting a string by using tuple function.
>>> print(tup)
('P', 'y', 't', 'h', 'o', 'n', ' ', 'S', 'r', 'i', 'n', 'i', 'v', 'a', 's')
>>> type(tup)                       <class 'tuple'>
```

Note:

```
>>> t = ("a","b","c",10)
>>> ''.join(t)
```

TypeError: sequence item 3: expected str instance, int found

Advantages of Tuple over List:

- Generally we use tuple for heterogeneous elements and list for homogeneous elements.
- Iterating through tuple is faster than list because tuples are immutable, So there might be a slight performance boost.
- Tuples can be used as key for a dictionary. With list, this is not possible because list is a mutable object.
- If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.