Python Tuple

- A tuple is a sequence of immutable objects, therefore tuple cannot be changed.
- The objects are enclosed within parenthesis and separated by comma.
- Tuple is similar to list. Only the difference is that list is enclosed between square bracket, tuple between parenthesis and List have mutable objects whereas Tuple have immutable objects.
- If Parenthesis is not given with a sequence, it is by default treated as Tuple.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

```
my_tuple = ()                # empty tuple
my_tuple = (1, 2, 3)          # tuple having integers
my_tuple = (1, "Hello", 3.4)  # tuple with mixed datatypes

my_tuple = 3, 4.6, "dog"   # tuple can be created without
                          parentheses, also called tuple packing
a, b, c = my_tuple                      # tuple unpacking is also
possible
```

Creating a tuple with one element is a bit tricky.
Having one element within parentheses is not enough. We will need a trailing comma to indicate that it is in fact a tuple.

```
>>> my_tuple = ("hello")   # only parentheses is not enough
>>> type(my_tuple)
<class 'str'>

>>> my_tuple = ("hello",)  # need a comma at the end
>>> type(my_tuple)
<class 'tuple'>

>>> my_tuple = "hello",    # parentheses is optional
>>> type(my_tuple)
<class 'tuple'>

>>>tupl1='a','mahesh',10.56
>>>tupl2=tupl1,(10,20,30)   #nested tuple

>>>tupl1
('a', 'mahesh', 10.56)
>>>tupl2
(('a', 'mahesh', 10.56), (10, 20, 30))
```

Accessing Tuple

Tuple can be accessed in the same way as List.

1. **Indexing**

```
>>> my_tuple = ['p','e','r','m','i','t']
>>> my_tuple[0]
'p'
>>> my_tuple[5]
't'
>>> my_tuple[6]   # index must be in range
...
IndexError: list index out of range
>>> my_tuple[2.0] # index must be an integer
...
TypeError: list indices must be integers, not float
>>> n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
>>> n_tuple[0][3]  # nested index
's'
>>> n_tuple[1][1]  # nested index
4
>>> n_tuple[2][0]  # nested index
1
>>> n_tuple = ([8, 4, 6], (1, 2, 3),"mouse")
>>> n_tuple[2][0]
'm'
```

2) Negative Indexing

Python allows negative indexing for its sequences.
The index of -1 refers to the last item, -2 to the second last item and so on.

```
>>> my_tuple = ['p','e','r','m','i','t']
>>> my_tuple[-1]
't'

>>> my_tuple[-6]
'p'
```

3) Slicing

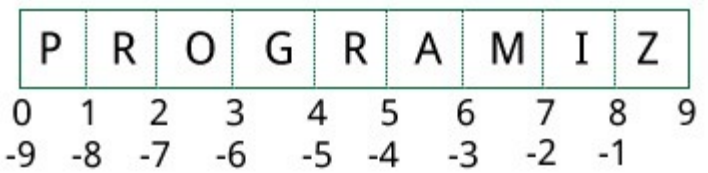
We can access a range of items in a tuple by using the slicing operator (colon).

```
>>> my_tuple = ('p','r','o','g','r','a','m','i','z')
>>> my_tuple[1:4]  # elements 2nd to 4th
('r', 'o', 'g')


>>> my_tuple[:-7]  # elements beginning to 2nd
('p', 'r')


>>> my_tuple[7:]   # elements 8th to end
('i', 'z')

>>> my_tuple[:]    # elements beginning to end
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```



Tuple Operations

a) Adding Tuple:

Tuple can be added by using the concatenation operator(+) to join two tuples.

```
    data1=(1,2,3,4)
    data2=('x','y','z')
    data3=data1+data2
    print data1
    print data2
    print data3
```

Output:

```
>>>
(1, 2, 3, 4)
('x', 'y', 'z')
(1, 2, 3, 4, 'x', 'y', 'z')
>>>
```

Note: The new sequence formed is a new Tuple.

b) Replicating Tuple:

Replicating means repeating. It can be performed by using '*' operator by a specific number of time.
Eg:

```
tuple1=(10,20,30);
tuple2=(40,50,60);
print tuple1*2
print tuple2*3
```

Output:

```
>>>
(10, 20, 30, 10, 20, 30)
(40, 50, 60, 40, 50, 60, 40, 50, 60)
>>>
```

c) Updating elements in a List:

Elements of the Tuple cannot be updated.
This is due to the fact that Tuples are immutable.
Whereas the Tuple can be used to form a new Tuple.
Eg:

```
data=(10,20,30)
data[0]=100
print data
```

Output:

```
>>>
Traceback (most recent call last):
          File "C:/Python27/t.py", line 2, in
        data[0]=100
TypeError: 'tuple' object does not support item assignment
>>>
```

d) Deleting elements from Tuple:

Deleting individual element from a tuple is not supported. However the whole of the tuple can be deleted using the del statement.
Eg:

```
data=(10,20,'rahul',40.6,'z')
print data
del data[3]                    # will show an error
del data                        #will delete the tuple data
print data

        #will show an error since tuple data is already deleted
```

e) Tuple Membership Test

We can test if an item exists in a tuple or not, using the keyword in.

```
>>> my_tuple = ('a','p','p','l','e',)
>>> 'a' in my_tuple
True


>>> 'b' in my_tuple
False


>>> 'g' not in my_tuple
True
```

Functions of Tuple:

There are following in-built Type Functions:

Function	Description
min(tuple)	Returns the minimum value from a tuple.
max(tuple)	Returns the maximum value from the tuple.
len(tuple)	Gives the length of a tuple
cmp(tuple1,tuple2)	Compares the two Tuples.
tuple(sequence)	Converts the sequence into tuple.

Few more Function Examples:

```
>>> tuple=(1,2,3,4)
>>> sum(tuple)                  # use of sum() function
10

>>> tuple=5,2,7,4
>>> tuple
(5, 2, 7, 4)
>>> sorted(tuple)               # use of sorted() function
[2, 4, 5, 7]
>>> tuple
(5, 2, 7, 4)

>>> sorted(tuple,reverse=True)
[7, 5, 4, 2]

>>> tuple=1,2,2,3,4,5
>>> tuple.count(2)              # use of count() function
2
>>> tuple.count(4)
1

>>> tuple.index(5)              # use of index() function
5                               # Element 5 is at index 5
```

Advantages of Tuple over List

1. Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.

2. Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.

3. If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.