Python Tuple

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses(), separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, string, etc.).

```
my_tuple = ()

print(my_tuple)

my_tuple = (1, 2, 3)

print(my_tuple)

my_tuple = (1, "Hello", 3.4)

print(my_tuple)

my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

print(my_tuple)
```

Access Tuple Elements

There are various ways in which we can access the elements of a tuple.

1. Indexing

We can use the index operator [ ] to access an item in a tuple, where the index starts from 0.So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range(6,7,... in this example) will raise an IndexError

The index must be an integer, so we cannot use float or other types. This will result in TypeError.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.

```
my_tuple = ('p','e','r','m','i','t')

print(my_tuple[0])   # 'p'

print(my_tuple[5])   # 't'

n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index

print(n_tuple[0][3])          # 's'

print(n_tuple[1][1])          # 4
```

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')

print(my_tuple[-1])      #  't'

print(my_tuple[-6])     #  'p'
```

In-built Functions for Tuple:

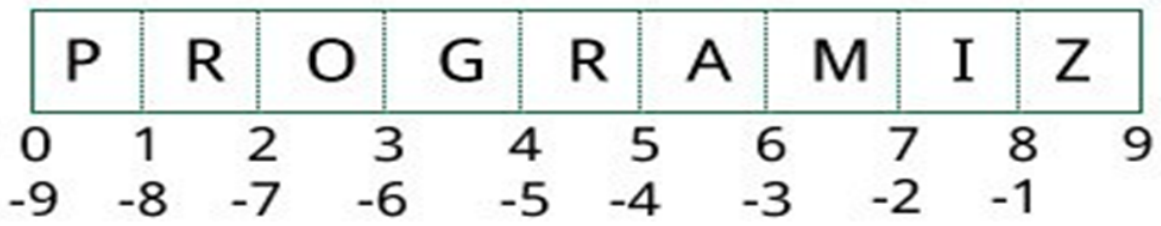
Function	**Description**
cmp(tuple1,tuple2)	Compares elements of two different tuples
len(tuple)	Returns the length of the tuple
max(tuple)	Returns the element with max value from the tuple
min(tuple)	Returns the element with min value from the tuple
tuple(seq)	Returns a tuple, by converting a list to a tuple. Takes a list in the parameter

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')

print(my_tuple[1:4])

print(my_tuple[:-7])

print(my_tuple[7:])

print(my_tuple[:])
```

Slicing can be best visualized by considering the index to be between the elements as shown below. So if we want to access a range, we need the index that will slice the portion from the tuple.



Changing a Tuple

Unlike lists, tuples are immutable.

This means that elements of a tuple cannot be changed once they have been assigned. But, if the element is itself a mutable data type like a list, its nested items can be changed.

We can also assign a tuple to different values (reassignment).

```
my_tuple = (4, 2, 3, [6, 5])

my_tuple[3][0] = 9          # Output: (4, 2, 3, [9, 5])

print(my_tuple)

my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

print(my_tuple)
```

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

We can also repeat the elements in a tuple for a given number of times using the * operator.

Both + and * operations result in a new tuple.

```
# Concatenation

# Output: (1, 2, 3, 4, 5, 6)

print((1, 2, 3) + (4, 5, 6))

# Repeat

# Output: ('Repeat', 'Repeat', 'Repeat')

print(("Repeat",) * 3)
```

Deleting a Tuple

As discussed above, we cannot change the elements in a tuple. It means that we cannot delete or remove items from a tuple.

Deleting a tuple entirely, however, is possible using the keyword del.

```
# Deleting tuples

my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# can't delete items

# TypeError: 'tuple' object doesn't support item deletion

# del my_tuple[3]

# Can delete an entire tuple

del my_tuple

# NameError: name 'my_tuple' is not defined

print(my_tuple)
```

Tuple Methods

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

Some examples of Python tuple methods:

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p'))  # Output: 2

print(my_tuple.index('l'))  # Output: 3
```

Other Tuple Operations

1. Tuple Membership Test

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

```
# Membership test in tuple

my_tuple = ('a', 'p', 'p', 'l', 'e',)

# In operation

print('a' in my_tuple)

print('b' in my_tuple)

# Not in operation

print('g' not in my_tuple)
```

2. Iterating Through a Tuple

We can use a for loop to iterate through each item in a tuple.

```
# Using a for loop to iterate through a tuple

for name in ('John', 'Kate'):

    print("Hello", name)
```

Advantages of Tuple

- Tuples being immutable, turns out to be a write-protected collection. Tuples can be of advantage when we want to store some secure read-only data that we cannot afford to be changed throughout our code.
- Tuples can store data of multiple data types, which makes them a heterogeneous collection.
- Tuple being a read only collection, has a faster iteration. (As they are stored in a single block of memory, and don’t have extra space for storing objects, they have a constant set of values)

Disadvantages of Tuple of Tuple

- Tuple’s being write protected, is an advantage but also a disadvantage as it cannot be used when we want to add or delete a specific element. Hence has a limited use case.
- Syntactically less readable as tuples can be created by either adding parentheses or by not providing them in case we have more than one element. But not using parentheses in the case of one element, will not create a tuple and hence a trailing comma in such case is required. This can make code readability a bit complex for some.
- As tuple is a class, it's stored on the heap and is overhead on the garbage collector.