Tuples In Python

A Tuple is a standard data type of Python that can store a sequence of values belonging to any type. The Tuples are depicted through parentheses i.e. round brackets, e.g. following are some tuples in Python:

```
()                      #tuple with no member, empty tuple
(1, 2, 3)               # tuple of integers
(1, 2.5. 3.7, 9)        # tuple of number(Int + float)
('a', 'b', 'c')         # tuple of characters
('a', 1, 'b', 3.5, 'zero')   # tuple of mixed values types
```

Creating Tuples

Creating a tuple is similar to list creation, but here you need to put a number of expression in parentheses. That is, use round bracket to indicate start and end of the tuple, and separate the items by commas.

T = (value,……)

T = ()

1. The Empty Tuple

The empty tuple is (). It is the tuple equivalent of 0 or "". You can also create an empty tuple as:

```
T = tuple()
```

It will generate an empty tuple and name that tuple as T.

2. Single Element Tuple

Making a tuple with a single element is tricky because if you just give a single element in round brackets, Python consider it is a value only, e.g.

```
>>>t = (1)
>>>t
1
```

(1) was treated as an integer expression, hence t stores and integer 1, not a tuple. To construct a tuple with one element just add a comma after the single element as shown below

```
>>> t = 3,
>>>t
(3)
>>>t2 = (4, )
```

```
>>>t2
4
```

3. Long Tuple

If a tuple contains many elements, then to enter such long tuples, you can split it across several lines.

```
sqrs = ( 0, 1, 4, 9, 16, 25,
         36, 49, 64, 81, 100)
```

Creating tuples from existing sequences

You can also use the built-in tuple object(tuple()) to create tuples from sequence as per the syntax given below:

T = tuple(<sequence>)

where <sequence> can be any kind of sequence object including strings, lists and tuples.

```
>>> t1 = tuple('Hello')
>>> t1
('H', 'e', 'l', 'l', 'o')
```

Tuple of single element can be crated with sequence having single element

```
>>> t1 = tuple('h')
>>> t1
('h',)
```

Tuple t2 is created from another sequence as list L

```
>>> L = ['w', 'e', 'r', 't', 'y']
>>> t2 = tuple(L)
>>> t2
('w', 'e', 'r', 't', 'y')
```

Accessing Tuples

Like lists, tuple elements are also indexed, i.e. its individual item can access with its index. It also has both forward index as 0, 1, 2, 3, 4 …. and backward indexing as -1, -2, -3, -4……

t = (‘a’, ‘e’, ‘I’, ‘o’, ‘u’)

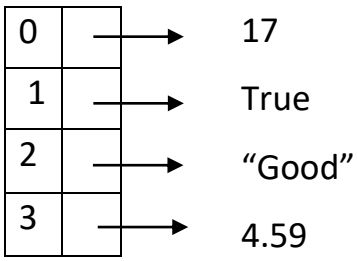
0	1	2	3	4
a	e	i	o	u
-5	-4	-3	-2	-1

```
>>>t[0]
a
>>>t[4]
>>>’u’
```

Memory Representation of Tuples

Tuples are stored in the memory exactly like string except that because some of their objects are larger than other, they store reference of each index instead of single character as in string.

Each of the individual items of tuple are stored somewhere else in the memory



Difference from Lists

Although tuples are similar to lists in many ways, yet there is an important difference is mutability of the two. Tuples are not mutable, while lists are. You cannot change individual elements of a tuple in place, but lists allow you to do so.

That is, following statement is fully valid for lists BUT not for tuples.

L[i] = element

is VALID for list BUT

T[i] = element

is INVALID for Tuple, as you cannot perform item-assignment in immutable type.

Traversing a Tuple

The for loop makes it easy to traverse or loop over the items in a tuple, as per the following syntax

```
for <item> in <Tuple>:
        process each item
```

For e.g.

```
T = (1, 2, 3, 4, 5)
>>> for i in T:
        print(i)
1
2
3
4
5
```

How it works

The loop variable i in the above loop will be assigned the Tuple element, one at a time. So i assigned 1 in first iteration and loop body process with this of i; in second iteration 2 will be assigned to i and loop body will be execute for this value of i; and so on.

If you only need to use indexes of elements to access them, you can use function range() and len() as per the following syntax:

```
        for index in range(len(T)):
                process T[index]
```

```
>>> for j in range(len(T)):
        print(T[j])
1
2
3
4
5
```

Tuple Operations

The most common operations that you perform with tuple include joining tuples and slicing tuples.

Joining Tuples

The '+' operator can also be use to join or concatenate two tuples.

```
>>> tp1 = (1,3, 5)
>>> tp2 = (6, 7, 8)
>>> t = tp1 + tp2
>>> t
(1, 3, 5, 6, 7, 8)
```

The '+' operator concatenates two tuples and create new tuple.

The + operator when used with tuples require that both the operands must be of tuple types. You cannot add a number or any other value to a tuple.

```
>>> tp1 + 2
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    tp1 + 2
TypeError: can only concatenate tuple (not "int") to tuple
```

Repeating and Replicating Tuples

Like strings and lists, you can use * operator to replicate a tuple specified number of times. e.g.

```
>>> tp1 * 3
(1, 3, 5, 1, 3, 5, 1, 3, 5)
```

Slicing the Tuples

Tuple slices, like list-slices or string slices are the sub part or the tuple extracted out. You can use indexes of tuple elements to create tuple slice as per following syntax

```
seq = T[start : stop]
>>> tp1 = (10, 12, 14, 20, 22, 24, 30, 32, 34)
>>> seq = tp1[3:-3]
>>> seq
(20, 22, 24)
```

Tuple also support slice steps too. That is, if you want to extract, not consecutive but every other element of the tuple, there is way out- the slice step.

```
seq = T[start : stop : step]
>>> tp1[0:10:2]
(10, 14, 22, 30, 34)
```

Unpacking Tuples

Creating a tuple from a set of values is called packing and its reverse, i.e. creating individual values from a tuple's elements is called unpacking.

```
<var1>, <var2>,  =  t
>>> t = (1, 2, 'A', 'B')
>>> w, x, y, z = t
```

Python will now assign each of the elements of tuple t to the variable on the left side of assignment operator.

Tuples Functions and Methods

Python also offers many built-in functions and methods for tuple manipulation.

1. The len() method

   This method returns length of tuple i.e. the count of elements in the tuple.

   len(<syntax>)

   ```
   >>> employee = ('John', 10000, 24, 'Sales')
   >>> len(employee)
   4
   ```

2. The max() method

   This method returns the element from the tuple having maximum value.

   max(tuple)

   - Takes an argument and returns an object(the element with maximum value.)

```
>>> tp1 = (10, 12, 14, 20, 22, 24, 30, 32, 34)
```

```
>>> max(tp1)
34
>>> tp2 = ("Karan", "Zubin", "Zara", "Ana")
>>> max(tp2)
'Zubin'
```

if a tuple contain elements of different types than the elements are compared as per the following rules:

- Sequence are always treated as higher than simple data types like integers and float.
- sequence are treated as lists < strings < tuples. That is a tuple is considered highest type of sequence.

So, if we compute maximum from a tuple with mixed type of elements then the result depends on these rules e.g.

```
>>> tp3 = ("1", "Rustam", (1, 2), [1, 2], ["1", "Rustam"])
>>> max(tp3)
(1, 2)
```

3. The min() method

   This method returns the element from the tuple having minimum value.

   min(<tuple>)

```
>>> tp1 = (10, 12, 14, 20, 22, 24, 30, 32, 34)
>>>min(tp1)
10
```