CLASS – XI

Chapter- Tuples

STUDY NOTE

Period -01

Introduction:

The Python tuples are sequences that are used to store a tuple of values of any type. You have learnt in earlier chapters that Python tuples are immutable i.e, you cannot change the elements of a tuple in place; Python will create a fresh tuple when you make changes to an element of tuple. Tuple is a type of sequence like strings and lists but it differs from them in the way that lists are mutable but strings and tuples are immutable. This chapter is dedicated to basic tuple manipulation in Python. We shall be taking about creating and accessing tuples, various tuple operations and tuple manipulations through some built-in functions.

Creating and Accessing Tuples:-

A tuple is 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 typles in Python:

```
( )                           # tuple with no member, empty tuple
(1, 2, 3)                     # tuple of integers
(1, 2, 5, 3, 7, 9)            # tuple of numbers (integers and floating point)
('a', 'b', 'c')               # tuple of characters
('a', 1, 'b', 3.5, 'zero')    # tuple of mixed value types
('One', 'Two', 'Three')       # tuple of strings
```

Before we proceed and discuss how to create tuples, one thing that must be clear is that Tuples are immutable (i.e, non-modifiable) i.e, you cannot change elements of a tuple in place.

Note:- Tuples are immutable sequences of Python i.e, you cannot change elements of a tuple in place.

Creating Tuples:-

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

```
(2, 4, 6)
('abc", 'def')
(1, 2.0, 3, 4.0)
( )
```

Thus to create a tuple you can write in the form given below:

```
T = ( )
```

T = (value, ...)

This construct is known as a tuple display construct. Consider some more examples:

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 considers it a value only, e.g.

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

   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 Tuples:- If a tu;e contains many elements, then to enter such long tuples, you can split it across several lines, as given below:

   sqrs = (0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625)

   Notice the opening parenthesis and closing parenthesis appear just in the beginning and end of the tuple.

4. Nested Tuples:- If a tuple contains an element which is a tuple itself then it is called nested tuple e.g, following is a nested tuple:

   t1 = (1, 2, (3, 4)

   The tuple t1 has three elements in it: 1, 2 and (3, 4). The third element of tuple t1 is a tuple itself; hence, t1 is a nested tuple.

Note: - Tuples are formed by placing a comma-separated tuple of expressions in parentheses.

Quick Interesting Facts:

- Tuples are immutable sequences of Python i.e., you cannot change elements of a tuple in place.

PERIOD-02

Creating Tuples from Existing Sequences:-

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

```
T = tuple (<sequence>)
```

where <sequence> can be any kind of sequence object including strings, lists and tuples.
Python creates the individual elements of the tuple from the individual elements of passed sequence. If you pass in another tuple, the tuple function makes a copy.
Consider following examples:-

```
>>> t1 = tuple ('hello')
>>> t1
('h', 'e', '1', '1', 'o')
>>> L = ['w', 'e', 'r', 't', 'y']
>>> t2 = tuple (L)
>>> t2
('w', 'e', 'r', 't', 'y')
```

You can use this method of creating tuples of single characters or single digits via keyboard input.
Consider the code below:

```
t1 = tuple (input('Enter tuple elements:'.))
Enter tuple elements : 234567
>>> t1
('2', '3', '4', '5', '6', '7')
```

See, with tuple () around input (), even if you not put parenthesis, it will create a tuple using individual characters as elements. But most commonly used method to input tuples is eval (input()) as shown below:

```
tuple = eval (input ("Enter tuple to be added:"))
print ("Tuple you entered:", tuple)
```

When you execute it, it will work somewhat like:

```
Enter tuple to be added : (2, 4, "a", "hjkjl", [3, 4])
Tuple you entered : (2, 4, "a", "hjkjl",[3, 4])
```

If you are inputting a tuple with eval (), then make sure to enclose the tuple elements in parenthesis. Please note sometimes (not always) eval () does not work in Python shell. At that time, you can run it through a script too.

Accessing Tuples:-

Tuples are immutable (non-editable) sequences having a progression of elements. Thus, like lists, you can access its individual elements. Before we talk about that, let us learn about how elements are indexed in tuples.

Similarity with Lists:-

Tuples are very much similar to lists except for the mutability. In other words, Tuples are immutable counter parts of lists. Thus, like lists, tuple elements are also indexed, i.e, forward indexing as 0, 1, 2, 3, ... and backward indexing as -1, -2, -3, ... see the figure.

Thus, you can access the tuple elements just like you access a list's or a string's elements e.g, Tuple [i] will give you elements at ith index; Tuple [a:b] will give you elements between indexes a to b-1 and so on. Put in other words, tuples are similar to lists in following ways:

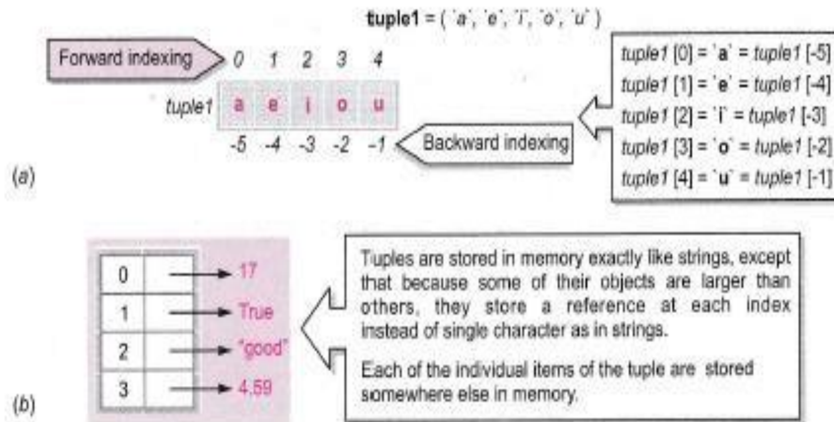

Figure 12.1 (a) Tuple Elements' two way indexing (b) How tuples are internally organized

- Length. Function len (T) returns the number of items (count) in the tuple T.
- Indexing and Slicing: - T[i] returns the item at index i (the first item has index 0).
  T[i : j] returns a new tuple, containing the objects between i and j excluding index j,
  T [i : j : n] returns a new tuple containing every nth item from index i to j, excluding index j. Just like lists.
- Membership operators: - Both 'in' and 'not in' operators work on Tuples just like they work for other sequences. That is, in tells if an element is present in the tuple or not and not in does the opposite.
- Concatenation and Replication operations + and *. The + operator adds one tuple to the end of another. The * operator repeats a tuple. We shall be taking about these two operations in a later section 12.3 - Tuple Operations.

Accessing Individual Elements:-

As mentioned, the individual elements of a tuple are accessed through their indexes given in square brackets. Consider the following examples:

```
>>> vowels = ('a', 'e', 'i', 'o', 'u')
>>> vowels [0]
'a'
>>> vowels [4]
'u'
>>> vowels [-1]
'u'
```

```
>>> vowels [-5]
'a'
```

Recall that like strings, if you pass in a negative index, Python adds the length of the tuple to the index to get its forward index. That is, for a 6-element tuple T, T[-5] will be internally computed as:

T[-5 + 6] = T[1], and so on.

Note:- While accessing tuple elements, if you pass in a negative index, Python adds the length of the tuple to the index to get element's forward index.

Difference from Lists:-

Although tuples are similar to lists in many ways, yet there is an important difference in 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). That is, if we have a list L and a tuple T, then

L [i] = element

is VALID for Lists. BUT

T [i] = element

is INVALID as you cannot perform item-assignment in immutable types.

Examples:

01. Find the output generated by following code fragments:

(a) plane = ("passengers", "Luggage")

plane [1] = "Snakes"

(b) (a, b, c) = (1, 2, 3)

(c) (a, b, c, d) = (1, 2, 3)

(d) a, b, c, d = (1, 2, 3)

(e) a, b, c, d, e = (p, q, r, s, t) = t1

(f) What will be the values and types of variables a, b, c, d, e, f, q, q, r, s, t after executing part e above if t1 contains (1, 20, 3, 4.0, 5) ?

(g) t2 = ('a')

type (T5)

(h) t3 = ('a',)

type (t3)

(i) T4 = (17)

type (T4)

(j) T5 = (17,)

type (T5)

(k) tuple = ('a', 'b', 'c', 'd', 'e')

```
tuple = ('A',) + tuple [1 : ]
print (tuple)
```

(l) t2 = (4, 5, 6)

```
t3 = (6, 7)
t4 = (t3 + t2
t5 = t2 + t3
print (t4)
print (t5)
```

(m) t3 = (6, 7)

```
t4 = t3 * 3
t5 = (t3 * (3)
print (t4)
print (t5)
```

(n) t1 = (3, 4)

```
t2 = ('3', '4')
print (t1 + t2)
```

02. Carefully read the given code fragments and figure out the errors that the code may produce.

(a) t = ('a', 'b', 'c', 'd', 'e')

```
print (t[5])
```

(b) t = ('a', 'b', 'c', 'd', 'e')

```
t [0] = 'A'
```

(c) t1 = (3)

```
t2 = (4, 5, 6)
t3 = t1 + t2
print (t3)
```

(d) t1 = (3,)

```
t2 = (4, 5, 6)
t3 = t1 + t2
print (t3)
```

(e) t2 = (4, 5, 6)

```
t3 = (6, 7)
```

ODM EDUCATIONAL GROUP Changing your Tomorrow

```
print (t3 - t2)
```

Quick Interesting Facts:

- To create a tuple, put a number of comma-separated expressions in round brackets.
- The empty round brackets i.e () indicate an empty tuple.

PERIOD-03

Tuples operations

Traversing a Tuple:-

Recall that traversal of a sequence means accessing and processing each element of it. Thus traversing a tuple also means the same and same is the tool for it, i.e, the Python loops. The for loop makes it easy to traverse or loop over the items in a tuple, as per following syntax:

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

For example, following loop shows each item of a tuple T in separate lines:

```
T = ('p', 'y', 't', 'h', 'o', 'n')
for a in T :
    print (T[a])
```

The above loop will produce result as :

```
p
y
t
h
o
n
```

How it Works:-

The loop variable a in above loop will be assigned the Tuple elements, one at a time. So, loop-variable a will be assigned 'P' in first iteration and hence 'P' will be printed; in second iteration, a will get element 'Y' will be printed; and so on.

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

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

Consider program below that traverses through a tuple using above format and prints each item of a tuple L in separate lines along with its index.

Program:

Program to print elements of a tuple ('Hello', "Isn't", 'Python', 'fun', '?') in separate lines along with element's both indexes (positive and negative)

```
T = ('Hello', 'Isn't', 'Python', 'fun', '?')
length = len (T)
for a in range (length) :
    print ('At indexes', a, 'and ', (a - length), 'element :', T[a])
```

```
At indexes 0 and -5 element: Hello
At indexes 1 and -4 element: Isn't
At indexes 2 and -3 element: Python
At indexes 3 and -2 element: fun
At indexes 4 and -1 element: ?
```

Tuple Operations:-

The most common operations that you perform with tuple include joining tuples and slicing tuples. In this section, we are going to talk about the same.

Joining Tuples:-

Joining two tuples is very easy just like you perform addition, literally ;). the + operator, the concatenation operator, when used with two tuples, joins two tuples.

Consider the example given below:

```
>>> tpl1 = (1, 3, 5)
>>> tpl2 = (6, 7, 8)
>>> tpl1 + tpl2
(1, 3, 5, 6, 7, 8)
```

As you can see that the resultant tuple has firstly elements of first tuple lst1 and followed by elements of second tuple lst2. You can also join two or more tuples to form a new tuple, e.g,

```
>>> tpl1 = (10, 12, 14)
>>> tpl2 = (20, 22, 24)
>>> tpl3 = (30, 32, 34)
>>> tpl = tpl1 + tpl2 + tpl3
>>> tpl
(10, 12. 14, 20, 22, 24, 30, 32, 34)
```

The + operator when used with tuples requires that both the operands must be of tuple types. You cannot add a number or any other value to a tuple. For example, following expressions will result into error:

tuple + number
tuple + complex - number
tuple + string
tuple + list

Consider the following examples:

```
>>> tpl1 = (10, 12, 14)
>>> tpl1 + 2
:
TypeError : can only concatenate tuple (not "int") to ltuple
>>> tpl1 + "abc"
:
TypeError: can only concatenate tuple (not "str") to tuple
```

Important:-

Sometimes you need to concatenate a tuple (say tpl) with another tuple containing only one element. In that case, if you write statement like:

```
>>> tpl + (3)
```

Python will return an error like :

```
:
TypeError : can only concatenate tuple (not "int") to tuple
```

The reason for above error is that a number enclosed in () is considered number only. To make it a tuple with just one element, just add a comma after the only element, i.e, make it (3,). Now Python won't return any error and successfully concatenate the two tuples.

```
>>> tpl = (10, 12, 14, 20, 22, 24, 30, 32, 34)
>>> tpl + (3, )
(10, 12, 14, 20, 22, 24, 30, 32, 34, 3)
```

Repeating or Replicating Tuples

Like strings and lists, you can use * operator to replicate a tuple specified number of times, e.g, if tpl1 is (1, 3, 5), then

```
>>> tpl1 * 3
(1, 3, 5, 1, 3, 5, 1, 3, 5)
```

Example:

01. Carefully read the given code fragments and figure out the errors that the code may produce.

(a) t = ('a', 'b', 'c', 'd', 'e')
    print (t[5])

(b) t = ('a', 'b', 'c', 'd', 'e')
    t [0] = 'A'

(c) t1 = (3)

```
    t2 = (4, 5, 6)
    t3 = t1 + t2
    print (t3)
(d) t1 = (3,)
    t2 = (4, 5, 6)
    t3 = t1 + t2
    print (t3)
(e) t2 = (4, 5, 6)
    t3 = (6, 7)
    print (t3 - t2)
```

Quick Interesting Facts:

- Tuples index their elements just like strings or lists, i.e two way indexing.
- Tuples are stored in memory exactly like strings, except that because some of their objects are larger than others, they store a reference at each index instead of single character as in strings.

PERIOD-04

Slicing the Tuples

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

```
seq = T [start : stop]
```

The above statement will create a tuple slice namely seq having elements of tuple T on indexes start, start +1, start +2, ... , stop - 1. Recall that index on last limit is not included in the tuple slice. The tuple slice is a tuple in itself that is you can perform all operations on it just like you perform on tuples. Consider the following example:

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

For normal indexing, if the resulting index is outside the tuple, Python raises an IndexError exception. Slices are treated as boundaries instead, and the result will simply contain all items between the boundaries. For the start and stop given beyond tuple limits in a tuple slice, Python simply returns the elements that fall between specified boundaries, if any.

For example, consider the following:

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

```
>>> tpl = [3 : 30]
(20, 22, 24, 30, 32, 34)
>>> tpl [-15 : 7]
(10, 12, 14, 20, 22, 24, 30)
```

Tuples also support slice steps too. That is, if you want to extract, not consecutive but every other element of the tuple, there is a way out - the slice steps. The slice steps are used as per following format:

```
seq = T[start : stop : step]
```

Consider some examples to understand this.

```
>>> tpl
(10, 12, 14, 20, 22, 24, 30, 32, 34)
>>> tpl [0 : 2]
(10, 14, 22, 30, 32)
>>> tpl [2 : 10 : 3]
(14, 24, 34)
>>> tpl [: : 3]
(10, 20, 30)
```

Consider some more examples:

```
seq = T [: : 2]        # get every other item, starting with the first
seq = T [5 : : 2]      # get every other item, starting with the
                       # sixth element, i.e, index 5
```

You can use the + and * operators with tuple slices too. For example, if a tuple namely TP has values as (2, 4, 5, 7, 8, 9, 11, 12, 34), then.

```
>>> Tp [2 : 5] * 3
(5, 7, 8, 5, 7, 8, 5, 7, 8)
>>> Tp [2 : 5] + (3, 4)
(5, 7, 8, 3, 4)
```

ODM EDUCATIONAL GROUP Changing your Tomorrow

Example:

(1) What will be stored in variables a, b, c, d, e, f, g, h after following statements?

perc = (88, 85, 80, 88, 83, 86)

i. a = perc [2:2]
ii. b = perc [2:]
iii. c = perc [:2]
iv. d = perc[:-2]
v. e = perc [-2 :]
vi. f = perc [2 : -2]
vii. g = perc [-2 : 2]
viii. h = perc [:]

02. What does each of the following expressions evaluate to? Suppose that T is the tuple containing:

("These", ["are", "a", "few", "words"], "that", "we", "will", "use")

(a) T[1] [0 : :2]

(b) "a" in T [1] [0]

(c) T [:1] + t[1]

(d) T [2 : 2]

(e) T[2] [2] in T [1]

Quick Interesting Facts:

- Tuples are similar to strings in many ways like indexing, slicing and accessing individual elements and they are immutable just like strings are