Python Tuple Manipulation

Tuple

A collection of elements which are surrounded by parenthesis ( ) are known as Tuple in Python. The sequence of elements may include int, float, String, List, tuple itself or any other type.

Note: Tuple are Ordered and Immutable (Unchangeable)

```
Tup=('hello', 'Pin 333001', 'abc.kvs@gov.in')

print(Tup)
Output: ('hello', 'Pin 333001', 'abc.kvs@gov.in')
```

Creating Tuple

A Tuple can be create as similar to other variable creation in Python. For creation of Tuple we just assign a Tuple values to a variable name followed by an equal sign.

```
Tup=(20,)                              Tup=(10,20,30,40,50)

Tup=(10,20.5,30.8,40.4,50,60)
     Tup=("Apple","Mango","Orange")

Tup=('A','E','I','O','U'])
     Tup=("Apple",100,"Banana",30)

Tup=(10,[2,4,6],"KVS",5.8,'A')
```

Creating an Empty Tuple

An empty List can be created by assigning ( ) to a variable.

Tup=( )

Here Tup is a tuple type Variable. # Check type(Tup)

```
Print(Tup)                    Output: ( )
```

1. An empty tuple can be created by tuple() constructor also.
   tup=tuple()
   ```
   Print(tup)                    Output: ( )
   ```

2. Creation of tuple with one element only.
   Syntax:
   ```
   Tuple_variable=(1-element,) # See the comma
   Tp=(10,)      # put a comma after element in ( )
   Print(type(Tp))   Output: <class 'tuple'>
   Tp=(10)       # No comma after element in ( )
   Print(type(Tp))   Output: <class 'int'>
   ```
   Here Tp(10) is similar to Tp=10 of integer type

3. Creation of tuple with more than one element
   Syntax:
   Tuple_variable=(elements separated by comma)

   ```
   Tup=(10,20,30,40,50)
   Tup=(10,20.5,30.8,40.4,50,60)
   Tup=("Apple","Mango","Orange")
   Tup=('A','E','I','O','U'])
   Tup=("Apple",100,"Banana",30)
   Tup=(10,[2,4,6],"KVS",5.8,'A')
   ```

Creating a Tuple from Existing Sequence

Creating a Tuple using tuple() Constructor

A Tuple can be created from other sequence or by using tuple() constructor.

Example:

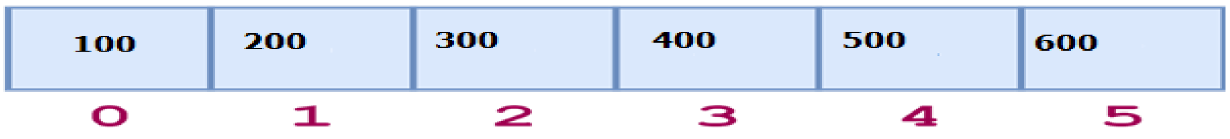
```
t=tuple("KVS")
Print(t)              Output: ('K', 'V', 'S')
t=tuple([20,40,50])
Print(t)              Output: (20, 40, 50)
L=[10,20]
S="KV"
T=(100,200)
Tup=tuple((L,S,T))     # tuple(), Requires 1 parameter so
put (L,S,T)
Print(Tup)                Output: ([10, 20], 'KV', (100, 200))
```

Indexing a Tuple

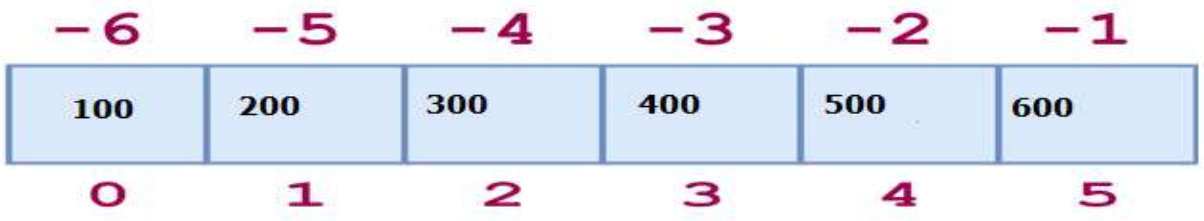
In Python, tuples are ordered sequences of element, and thus can be indexed in this way. Individual elements in a Tuple can be accessed by specifying the Tuple name followed by an index number in square brackets ([]).

Tuple indexing in Python is zero-based: the first element in the Tuple has index 0, the next has index 1, and so on. The index of the last element will be the length of the Tuple minus one.

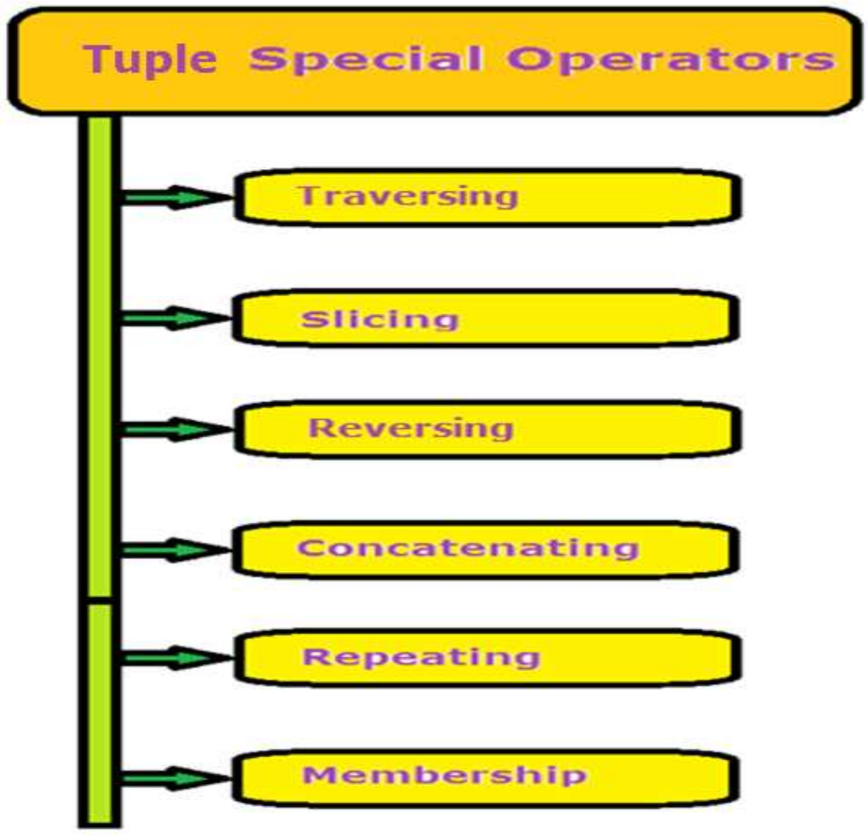
For example, a schematic diagram of the indices of the Tuple (100,200,300,400,500,600) would look like this:



Tuple indices can also be specified with negative numbers, in which case indexing occurs from the end of the Tuple backward: -1 refers to the last element, -2 the second-to-last element, and so on. Here is the same diagram showing both the positive and negative indices into the Tuple (100,200,300,400,500,600):



Special Tuple Operators



Traversing a Tuple

Square brackets can be used to access elements of the Tuple.

Get the element at position 1 (remember that the first element has the position 0):

```
Tup=(“Apple”,100,”Banana”,30)
Print( Tup[1] )              # Output:  100
Print( Tup[-2] )             # Output: Banana
```

Display All List elements using loop

```
Tup=(10,[2,4,6],"KVS",5.8,'A')
for val in Tup:
       print(val, end=”,”)
# Output:   10,[2, 4, 6],KVS,5.8,A
```

Display All List elements using range()

```
Size=len(Tup)

for index in range(Size):

     print(Tup[index], end=” ”)

# Output:   10,[2, 4, 6],KVS,5.8,A
```

Tuple Slicing

Python also allows a form of indexing syntax that extracts sub tuple from a Tuple, known as Tuple slicing.

Format of Tuple slicing:
Tuple_Variable [Start Index: End Index: Step Value]
Where start, end index and step value separated with colon (:)
Start index is by default Zero (0)
End Index is by default end of Tuple.
Step Value is by default 1

Note:

1. Start index always include in result
2. End index (if given) always exclude in result
3. End index (if not given) then last index of Tuple include

Get the elements from position 2 to position 5 (not included):

T=(100,200,300,400,500,600)
print(T[2:5])
	Output:
	(300, 400, 500)

Use negative indexes to start the slice from the end of the Tuple:

Get the elements from position 5 to position 1, starting the count from the end of the Tuple:

Tup=(100,200,300,400,500,600)
print(Tup[-5:-2])
	(200, 300, 400)

print(Tup[2:8:2])	(300, 500)
print(Tup[2:8:-1])	()
print(Tup[::2])	(100, 300, 500)
print(Tup[5:0:-1])	(600, 500, 400, 300, 200)
print(Tup[10:0:-2])	(600, 400, 200)
print(Tup[::-1])	(600, 500, 400, 300, 200, 100)
print(Tup[::-2])	(600, 400, 200)

Reversing Tuple

A Tuple in Python can be reversed by using following ways.

1. By using Traversing (Negative index Traversing)
2. By using loop

```
Tup=(100,”KVS”,20.5,’A’,500)
print(Tup[-1::-1])
    Output: (500, 'A', 20.5, 'KVS', 100)
```

By Using Loop:

```
    Tup=(100,”KVS”,20.5,’A’,500)
    size=len(Tup)
    size=Size-1
    While (size>=0):
        print(Tup[size], end=””)
        size=size-1
Output: 500, 'A', 20.5, 'KVS', 100
```

Concatenating Tuple

Concatenating Tuple mean joining two or more strings with the help of + operator and make a new Tuple.

Example:

```
	t1=(1,2,3)
	t2=("Jaipur","Ajmer")
	t3=t1+t2	print(t3)
	Output: (1, 2, 3, 'Jaipur', 'Ajmer')
	t3=t1+(300,)
	Print(t3)			# Output: (1, 2, 3, 300)
	t3=(100,200)+("abc","xyz")
	Print(t3)
	Output: (100, 200, 'abc', 'xyz')
	t3=t1+300	Print(t3)
Output: TypeError: can only concatenate tuple (not "int") to tuple
```

Replicating Tuple

The replicating operator * is used to repeating multiple times of a Tuple to create multiple copies of given Tuple.

Example:

```
T=(100,200)

Print(T*3)

	Output: (100, 200, 100, 200, 100, 200)

T = 2 * ("Jaipur","Ajmer")

Print(T)

	Output: ('Jaipur', 'Ajmer', 'Jaipur', 'Ajmer')
```

Membership Operator in Tuple

Membership operator is Boolean type operator. It used to check whether a sub Tuple is present in main Tuple or not.. There are two membership operators are used in Tuple.

1. in operator: If sub Tuple is present in main Tuple then it will return True otherwise it will return False result.
2. not in operator: If sub Tuple is not present in main Tuple then it will return True otherwise it will return False result.

Example:

```
T=(100, 200, 100, 200, 100, 200)

B=200 in T

Print(B)                    # Output: True

Print( 300 in T)            # Output: False

Print([100,200] in T)       # Output: False

Print(300 not in T)         # Output: True

Print(100 not in T)         # Output: False
```

Relational / Comparison Operator

The relational operators of Python can also apply on strings also. The relational operators are also the Boolean operators and produce either True or False result.

Tuple comparison is different from numeric values comparison. Python compares the Tuple using their ASCII (American Standard Code for Information Interchange). Python compares the ASCII values of first letter of each Tuple, if they are equal then next elements of Tuple compare and so on, until it finds differ ASCII Values.

Letter	ASCII Value
A	65
B	66
Z	90
a	97
a	98
z	122

In Python by ord('element') function, the ASCII value of any element can be find. Similarly chr(ASCII Value) can used to find Character of given ASCII value.

Example:

T1=(100,200,300)

T2=(100,200,300)

Print(T1 ==T2) Output: True

Print(T1 < T2) Output: False

Print(T1 >=T2) Output: True

Print(T1 > T2) Output: False

Print(T1 != T2) Output: False

Tuple are Immutable

Tuple are immutable in nature in Python. Its mean the content of Tuple cannot be changed once it is created. In other words, the assignment operator does not work to change the content of Tuple.

Example:

```
t=(100,200,300)

t[1]=500
```

Output: TypeError: 'tuple' object does not support item assignment

Tuple Built in Functions

Tuple is immutable in Python so it do not support methods such as append(), extend(), insert(), remove() and pop().

Note: All Tuple methods returns new values. They do not change the original Tuple.

1. len():
   The len() function returns the length of a Tuple or number of elements available in tuple:
   Syntax:

   ```
   len(Tuple_variable)
   ```

   Example:

   ```
   T=(10,"KVS",[2,3,4],100)
   print(len(T))
   ```
   Output: 4

2. count():

The count() used to return the count of element repeated in tuple. That mean it finds that how many times an element occurs in tuple. This function requires single parameter as element that to be counted in tuple.

Syntax:

```
tuple_variable.count(Element)
```

Example:

```
T=(10,20,30,10,30,10)
Print(T.count(10))
Output: 3
Print(T.count(100))
Output: 0
```

3. index():

This function Searches in the Tuple for a specified value and returns the index of first occurrence where it was found. If element or index not found in tuple then generate
"ValueError: tuple.index(x): x not in tuple"

Syntax:

Tuple_variable.index(element, occurrence number)

By default occurrence number will be 1.
If occurrence provided as 0 then output will be 0.

Example:

```
T=(10,20,30,10,30,10)
x = T.index(10)
print(x)
Output: 0
Print(T.index(10,2))        Output: 3
Print(T.index(10,3))        Output: 5
```

4. sorted():

The sorted() function sort the elements of tuple and return a list after sorting.

Syntax:

```
sorted(Tuple_variable)
```

Example:

```
Tup=(10,40,20,5,25,35,23,5,2)
S=Sorted(Tup)
Print(S)
```

Output: [2, 5, 5, 10, 20, 23, 25, 35, 40]

5. any():

This function returns Boolean value. If tuple is empty then it will return False and if tuple contains at least 1 element then it will return True result.

Syntax:

```
any(Tuple_variable)
```

Example:

```
    T=()
Print(any(T))      Output: False
tup=(10,)
Print(any(tup))         Output: True
Tup=(10,20,30)
Print(any(Tup))        Output: True
```

6. max()

This function will return maximum value from the tuple.

Syntax:

```
max(tuple _Variable)  OR    max(tuple [range])
```

Example:

```
tup=(10,40,20,15,26,38)
Print(max(tup))                         # Output:40
Print(max(tup[2:5]) )                   # Output:26
T=("KVS","JJN","JPR")
Print(max(T))                           # Output:'KVS'
```

7. min()

This function will return maximum value from the tuple.

Syntax:

```
min(tuple _Variable)  OR    min(tuple [range])
```

Example:

```
tup=(10,40,20,15,26,38)
Print(min(tup))                         # Output:10
Print(min(tup[2:5]) )                   # Output:15
T=("KVS","JJN","JPR")
Print(min(T))                           # Output:'JJN'
```

Deletion Operation on List

1. del keyword

   The del keyword is used to delete any variable / object in Python from memory.

   When we provide a tuple variable without range then del keyword will delete the complete tuple variable from memory and it will not further allowed to use.

   (Deletion / Updation of particular element or range of elements are not allowed in tuple)

   Syntax:

   del tuple _variable

Example:

```
Tup=[10,30,20,50,40]
del Tup[2]
# TypeError: 'tuple' object does not support item deletion
del Tup[1:3]
# TypeError: 'tuple' object does not support item deletion
del Tup
# It will remove tuple variable from memory
print(tup)
NameError: name 'Tup' is not defined
```

Nut Shell

- Tuple can’t support to add element because of its immutable property.
- There is no append(), extend(), insert() methods for tuple.
- Tuple has no remove(), pop() and clear() method due to its immutable property.

Assignments:

1. Write a program to find reverse of List.
2. WAP to find Odd number in List.
3. WAP to find Even number in List.
4. WAP to find Sum of odd index elements in List.
5. WAP to find Sum of even index elements in List.
6. WAP to find maximum number in List.
7. WAP to find minimum number in List.
8. WAP to print elements in ascending order.
9. WAP to print elements in descending order.
10. Explain with suitable Python coding examples that tuple is immutable in nature.