Tuples in python

A tuple is a Python sequence which stores a group of elements or items. Tuples are similar to lists but the main difference is tuples are immutable whereas lists are mutable. Since tuples are immutable, once we create a tuple we cannot modify its elements. Hence we cannot perform operations like append(), extend(), insert(), remove(), pop() arid clear() on tuples. Tuples are generally used to store data which should not be modified and retrieve that data on demand

Creating Tuples

We can create a tuple by writing elements separated by commas inside parentheses (). The elements can be of the same datatype or different types. For example, to create an empty tuple, we can simply write empty parenthesis, as:
tup1 = ()

If we want to create a tuple with only one element, we can mention that element in parentheses and after that a comma is needed, as:
tup2=(10,)

Here is a tuple with different types of elements:
tup3= (10. 20.-30.1, 40.5, 'Hyderabad', 'New Delhi')

We can create a tuple with only one of type of elements also, like the following: tup4= (10, 20, 30)

If we do not mention any brackets and write the elements separating them by commas, then they are taken by default as a tuple. See the following example:

tup5 = 1, 2, 3, 4

The point to remember is that if you do not use any brackets, it will become a tuple and not a list or any other datatype.

It is also possible to create a tuple from a list. This is done by converting a list ints tuple using the tuple() function. Consider the following example:

list [1, 2, 3]

```
tpl=tuple(list)
print(tpl)
```

The tuple is shown below:

```
(1, 2, 3)
```

Another way to create a tuple is by using the range() function that returns a sequence. To create a tuple ‘tpl' that contains numbers from 4 to 8 in steps of 2, we can use the range() function along with tuple() function, as:

```
tpl= tuple(range(4, 9, 2))
print(tpl)
```

The preceding statements will give:

```
(4, 6. 8)
```

Accessing the Tuple Elements

Accessing the elements from a tuple can be done using indexing or slicing. This is the same as that of a list. For example, let's take a tuple by the name 'tup' as:

```
tup (50,60,70,80, 90, 100)
```

Indexing represents the position number of the element in the tuple. Now, tuple represents the element, tup[1] represents the 1" element and so on.

```
print (tup[0])
```

The preceding will give:

```
50
```

Now, if you write:

```
print (tup [5])
```

Then the following element appears:

```
100
```

Similarly, negative indexing is also possible. For example, tup[-1] indicates the last element and tup[-2] indicates the second element from the end and so on. Consider the following statement:

```
print(tup[-1])
```

The preceding statement will give:

```
100
```

If you write,

```
print (tup[-6])
```

Then the following output appears:

```
50
```

Slicing represents extracting a piece or part of the tuple. Slicing is done in the format. start stop: stepsize). Here, 'start' represents the position of the starting element and stop represents the position of the ending element and the stepsize indicates the incrementation. If the tuple contains 'n' elements, the default values will be 0 for start and n-1 for 'stop' and 1 for 'stepsize'. For example, to extract all the elements from the tuple, we can write:

```
print(tup[:])
```

The elements of tuple appear:

```
(50, 60, 70, 80, 90, 100)
```

For example, to extract the elements from 1 to 4, we can write:

```
print(tup [1:4])
```

The elements of tuple appear:

```
(60, 70, 80)
```

Similarly, to extract every other element, i.e. alternate elements, we can write:

```
print(tup[::2])
```

The following output appears:

```
(50, 70, 90)
```

Negative values can be given in the slicing. If the 'step size' is negative, the elements are extracted in reverse order as:

```
print (tup[::-2])
```

The elements appear in the reverse order:

```
(100, 80. 60)
```

When the step size is not negative, the elements are extracted from left to right. In the following example, starting position -4 indicates the 4th element from the last. Ending position -1 indicates the last element. Hence, the elements from 4th to one element before the ending element (left to right) will be extracted.

```
print(tup[-4:-1])
```

The following elements appear

```
(70, 80, 90)
```

In most of the cases, the extracted elements from the tuple should be stored in separate variables for further use. In the following example, we are extracting the first two elements from the student' tuple and storing them into two variables.

```
student (10, 'Vinay kumar’, 50,60,65,61,70)
rno, name student [0:2]
```

Now, the variable ‘rno’ represents 10 and 'name' represents Vinaykumar. Consider the following statement:

```
print (rno)
```

The preceding statement

```
10
```

Now, if you write:

```
print(name)
```

The preceding statement will provide the name of the student:

```
Vinay kumar
```

If we want to retrieve the marks of the student from 'student' tuple, we can do it as:

```
marks=student [2:7]
for i in marks:
print (i)
```

The preceding statements will give the following output:

```
50
60
65
61
70
```

Basic Operations on Tuples

The 5 basic operations: finding length, concatenation, repetition, membership and iteration operations can be performed on any sequence, be it a string, list, tuple or a dictionary.

To find the length of a tuple, we can use len() function. This returns the number of elements in the tuple.
Consider the following example:

```
student (10, 'vinaykumar', 50,60,65,61,70)
len(student)
```

The preceding statement will give the following output:

```
7
```

We can concatenate or join two tuples and store the result in a new tuple. For example, the student paid a fees of Rs. 25,000.00 every year for 4 terms, then we can create a 'fees' tuple as:

```
fees=(25000.00,)*4
print (fees)
```

The preceding statement will give the following output:

```
(25000.0, 25000.0, 25000.0, 25000.0)
```

Now, we can concatenate the ‘student’ and ‘fees' tuples together to form a new tuple ‘student1' as:

```
student1=student+fees
print(student1)
```

The preceding statement will give:

```
(10. vinay kumar, 50, 60, 65, 61, 70, 25000.0, 25000.0, 25000.0,25000.0)
```

Searching whether an element is a member of the tuple or not can be done using in' and not in operators. The in' operator returns True if the element is a member. The 'not in operator returns True if the element is not a member. Consider the following statements:

```
name=’Vinay kumar’
name in student1
```

The preceding statements will give:

```
True
```

Suppose if you write:

```
name not in student1
```

Then the following output will appear:

False

The repetition operator repeats the tuple elements. For example, we take a tuple "tpl' and repeat its elements for 4 times as:

```
tpl= (10, 11, 12)
tpl1 = tp1*3
print (tpl1)
```

The preceding statements will give the following output:
(10, 11, 12, 10, 11, 12, 10, 11, 12).

Functions to Process Tuples

There are a few functions provided in Python to perform some important operations on tuples. Any function is called directly with its name. In this table, count() and index() are not functions. They are methods. Hence, they are called in the format: object.method().

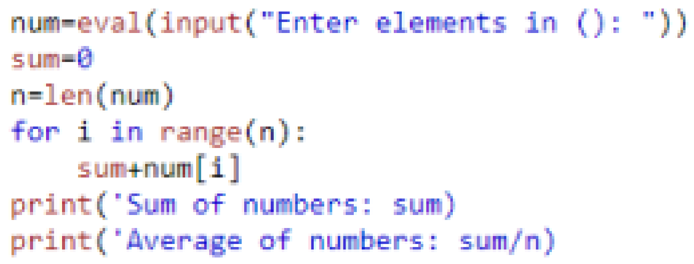
Function	Example	Description
len()	len(tpl)	Returns the number of elements in the tuple.
min()	min(tpl)	Returns the smallest element in the tuple.
max()	max(tpl)	Returns the biggest element in the tuple.
count()	tpl.count(x)	Returns how many times the element x' is found in tpl
index()	tpl.index(x)	Returns the first occurrence of the element 'x' in tpl. Raises ValueError if 'x' is not found in the tuple.
sorted()	sorted(tpl)	Sorts the elements of the tuple into ascending order. sorted(tpl, reverse=True) will sort in reverse order.

We will write a program to accept elements of a tuple from the keyboard and find their sum and average. In this program, we are using the following statement to accept elements from the keyboard directly in the format of a tuple (i.e. inside parentheses).

```
num=eval(input("Enter elements in (): "))
```

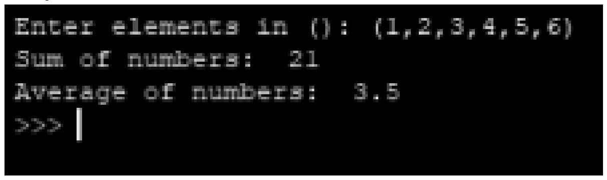
The eval() function is useful to evaluate whether the typed elements are depending upon the format of brackets given while typing the elements. If we type the elements inside the square braces as: [1,2,3,4,5] then they are considered as elements of a list. If we enter the elements inside parentheses as: (1,2,3,4,5), then they are considered as elements of a tuple. Even if you
do not type any brackets, then by default they are taken as elements of

tuple. Ex:



```
num=eval(input("Enter elements in (): "))
sum=0
n=len(num)
for i in range(n):
    sum+num[i]
print('Sum of numbers: sum)
print('Average of numbers: sum/n)
```

Output:



```
Enter elements in (): (1,2,3,4,5,6)
Sum of numbers:  21
Average of numbers:  3.5
>>> 
```

When the above program asks the user for entering the elements, the user can type the elements inside the parentheses as: (1,2,3,4,5,6) or without any parentheses 1,2,3,4,5,6. When a group of elements are entered with commas (,), then they are default taken as a tuple in Python.

In the program below, first we enter the elements separated by commas. These elements are stored into a string 'str' as:

str=input('Enter elements separated by commas: ').split(',')

Then each element of this string is converted into integer and stored into a list ‘lst'

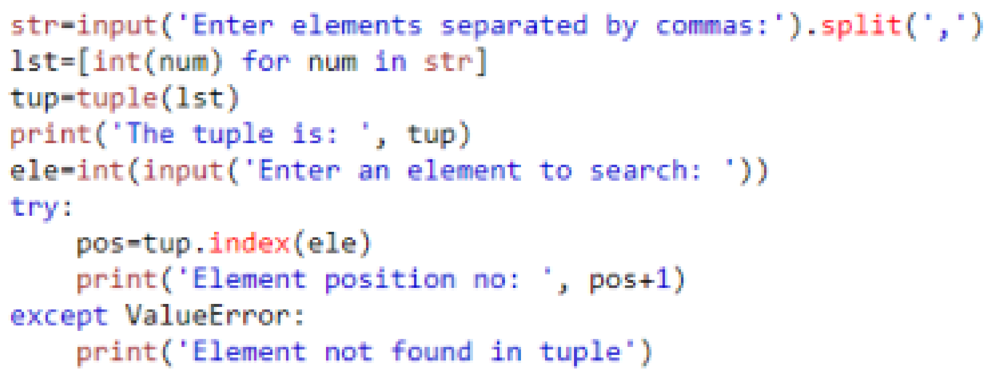
as: lst= [int (num) for num in str]

This list can be converted into a tuple using tuple() function as:

tup tuple(lst)

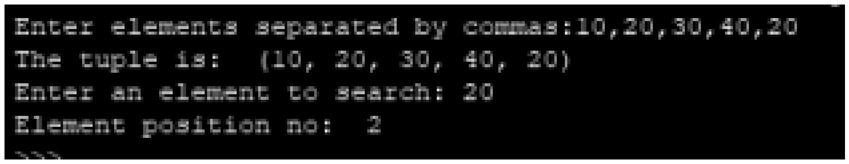
Later, index() method is used to find the first occurrence of the element 'ele' as: pos=tup.index(ele)

If the 'ele' is not found in the tuple, then there will be an error by the name ValueError' which should be handled using try and except blocks which is shown in the program



```
str=input('Enter elements separated by commas:').split(',')
lst=[int(num) for num in str]
tup=tuple(lst)
print('The tuple is: ', tup)
ele=int(input('Enter an element to search: '))
try:
    pos=tup.index(ele)
    print('Element position no: ', pos+1)
except ValueError:
    print('Element not found in tuple')
```

Output:



```
Enter elements separated by commas:10,20,30,40,20
The tuple is:  (10, 20, 30, 40, 20)
Enter an element to search: 20
Element position no:  2
>>>
```

Nested Tuples

A tuple inserted inside another tuple is called a nested tuple. For example, tup(50.60,70,80,90,(200, 201))

Observe the last parentheses in the tuple ‘tup', ie. (200, 201). These parentheses represent that it is a tuple with 2 elements inserted into the tuple ‘tup'. The tuple (200, 201) is called a nested tuple as it is inside another tuple.

The nested tuple with the elements (200, 201) is treated as an element along with other elements in the tuple ‘tup'. To retrieve the nested tuple, we can access it as an ordinary element as tup[5] as its index is 5. Now, consider the following statement:
print('Nested tuple=’, tup[6])

The preceding statement will give the following output:
Nested tuple= (200, 201).

Every nested tuple can represent a specific data record. For example, to store 4 employees data in 'emp' tuple, we can write:

emp=((10, "Vijay”, 9000.90), (20, "Nihaar" 5500.75), (30, “vanaja”,8900.00). (40, “Kapoor", 5000.50))
Here, ‘emp' is the tuple name. It contains 4 nested tuples each of which represents the data of an employee. Every employee's identification number, name and salary are stored as nested tuples.

Sorting Nested Tuples

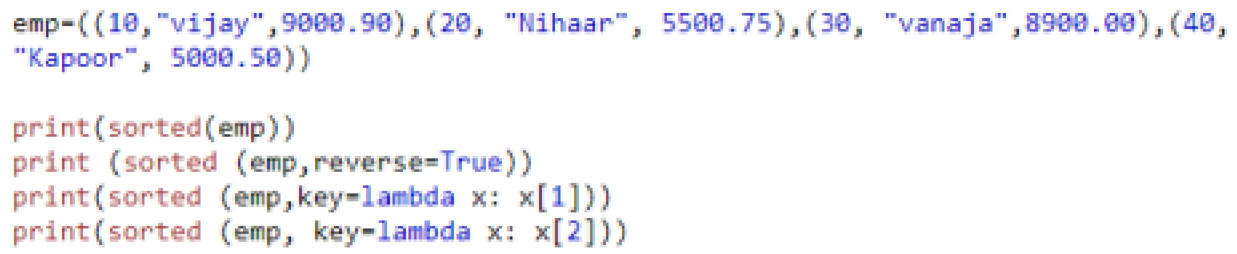
To sort a tuple, we can use a sorted() function. This function sorts by default into ascending order. For example,

print (sorted (emp))

will sort the tuple emp' in ascending order of the 0th element in the nested tuples, i.e identification number. If we want to sort the tuple based on employee name, which is the 1st element in the nested tuples, we can use a lambda expression as:

print (sorted (emp, key=lambda x: x[1]))

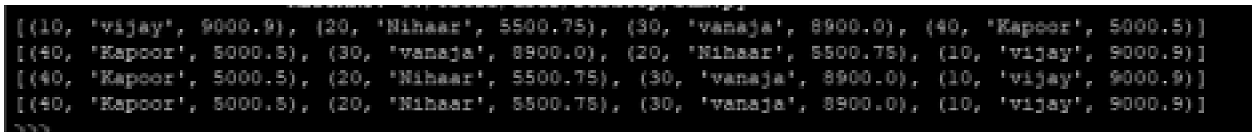
Here, key indicates the key for the sorted() function that tells on which element sorting should be done. The lambda function: lambda x: x[1] indicates that x[1] should be taken as the key that is nothing but the 1st element. If we want to sort the tuple based on salary, we can use the lambda function as: lambda x: x[2].



```
emp=((10,"vijay",9000.90),(20, "Nihaar", 5500.75),(30, "vanaja",8900.00),(40,
"Kapoor", 5000.50))

print(sorted(emp))
print (sorted (emp,reverse=True))
print(sorted (emp,key=lambda x: x[1]))
print(sorted (emp, key=lambda x: x[2]))
```

Output:



```
[(10, 'vijay', 9000.9), (20, 'Nihaar', 5500.75), (30, 'vanaja', 8900.0), (40, 'Kapoor', 5000.5)]
[(40, 'Kapoor', 5000.5), (30, 'vanaja', 8900.0), (20, 'Nihaar', 5500.75), (10, 'vijay', 9000.9)]
[(40, 'Kapoor', 5000.5), (20, 'Nihaar', 5500.75), (30, 'vanaja', 8900.0), (10, 'vijay', 9000.9)]
[(40, 'Kapoor', 5000.5), (20, 'Nihaar', 5500.75), (30, 'vanaja', 8900.0), (10, 'vijay', 9000.9)]
```

Inserting Elements in a Tuple

Since tuples are immutable, we cannot modify the elements of the tuple once it is created. Now, let's see how to insert a new element into an existing tuple. Let's Take as an existing tuple. Since x cannot be modified, we have to create a new tuple y with the newly inserted element. The following logic can be used:

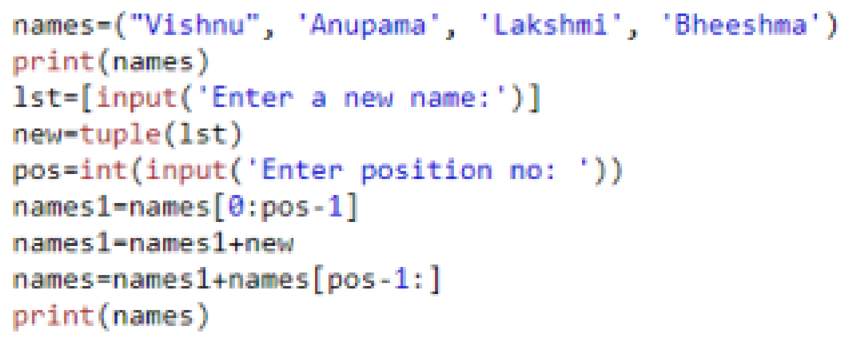
First of all, copy the elements of ‘x' from 0th position to pos-2 position into ‘y’ as:
y=x[0:pos-1]

Concatenate the new element to the new tuple ‘y’.

y=y+new

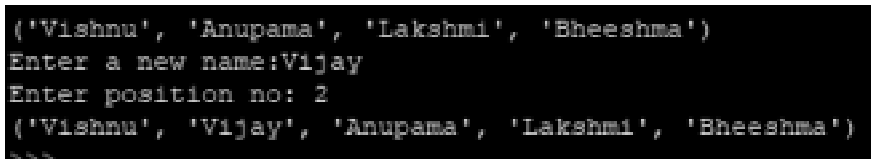
Concatenate the remaining elements (from pos-1 till end) of x to the new tuple y. The total tuple can be stored again with the old name ‘x’.

x=y+x[pos-1:]



```
names=("Vishnu", 'Anupama', 'Lakshmi', 'Bheeshma')
print(names)
lst=[input('Enter a new name:')]
new=tuple(lst)
pos=int(input('Enter position no: '))
names1=names[0:pos-1]
names1=names1+new
names=names1+names[pos-1:]
print(names)
```

Output:



```
('Vishnu', 'Anupama', 'Lakshmi', 'Bheeshma')
Enter a new name:Vijay
Enter position no: 2
('Vishnu', 'Vijay', 'Anupama', 'Lakshmi', 'Bheeshma')
```

This program works well with strings. Of course the same program can be used with integers also, if we change the input statement that accepts an integer number as:

lst= [int (input("Enter a new integer:”))]

Modifying Elements of a Tuple

It is not possible to modify or update an element of a tuple since tuples are immutable. If we want to modify the element of a tuple, we have to create a new tuple with a new value in the position of the modified element. The logic used in the previous section holds good with a slight difference in the last step. Let's take the existing tuple and y' is the new tuple.

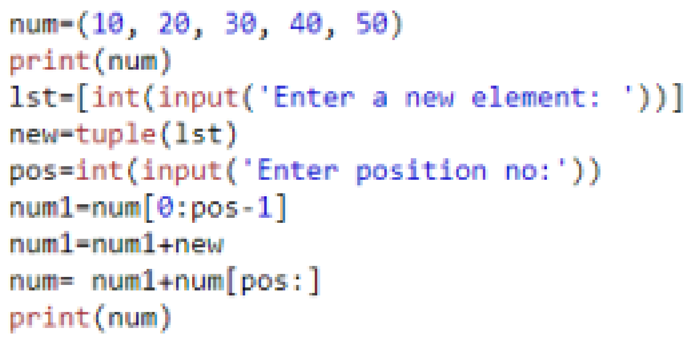
1. First of all, copy the elements of ‘x' from 0th position to pos-2 position into ‘y'

as: y=x[0:pos-1]

2. Concatenate the new element to the new tuple ‘y’ .Thus the new element is stored in the position of the element being modified.
y=y+new

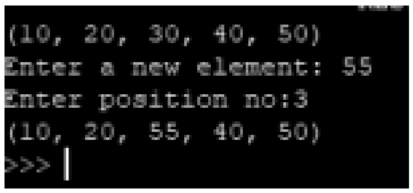
3. Now concatenate the remaining elements from x' by eliminating the element which is at ‘pos-1’. It means we should concatenate elements from ‘pos' till the end. The total tuple can be assigned again to the old name ‘x’.

x=y+x[pos:]



```
num=(10, 20, 30, 40, 50)
print(num)
lst=[int(input('Enter a new element: '))]
new=tuple(lst)
pos=int(input('Enter position no:'))
num1=num[0:pos-1]
num1=num1+new
num= num1+num[pos:]
print(num)
```

Output:



```
(10, 20, 30, 40, 50)
Enter a new element: 55
Enter position no:3
(10, 20, 55, 40, 50)
>>> |
```

Deleting Elements from a Tuple

The simplest way to delete an element from a particular position in the tuple is to copy all the elements into a new tuple except the element which is to be deleted. Let's assume that the user enters the position of the element to be deleted as 'pos'. The corresponding position will be 'pos-1' in the tuple as the elements start from 0th position. Now the log will be:

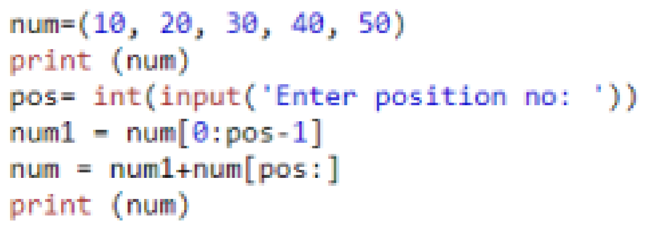
1. First of all, copy the elements of x' from 0th position to pos-2 position into y'

as: y = x[0:pos-1]

2. Now concatenate the remaining elements from 'x' by eliminating the element which is at

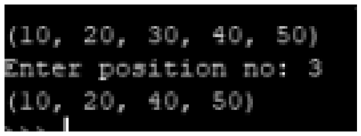
‘pos-1'. It means we should concatenate elements from 'pos' till the end. The tota tuple can be assigned again to the old name 'x'.

x=y+x[pos:]
Ex:



```
num=(10, 20, 30, 40, 50)
print (num)
pos= int(input('Enter position no: '))
num1 = num[0:pos-1]
num = num1+num[pos:]
print (num)
```

Output:



```
(10, 20, 30, 40, 50)
Enter position no: 3
(10, 20, 40, 50)
```