Python Tuples

Tuple is an **order collection** of **immutable** (unchangeable) data structures of (heterogeneous) **different data types**.

- It is an **ordered collection**, so it preserves the order of elements in which they were defined.
- A tuple is **immutable** in Python, thus data cannot be changed once it's assigned.
- The data that can be stored in a tuple are heterogeneous in nature i.e. we can store multiple data of different data types like String, Integers, and objects as well in a single tuple.

Tuple vs List

1. The elements of a **list** are **mutable** whereas the elements of a **tuple** are **immutable**.

2. **When we do not want to change the data over time**, the **tuple** is a preferred data type whereas when **we need to change the data in future**, **list** would be a wise option.

3. Iterating over the elements of a **tuple** is faster compared to iterating over a **list**.

4. Elements of a **tuple** are enclosed in parenthesis whereas the elements of **list** are enclosed in square brackets.

Creating a Tuple

Tuples are defined by assigning comma separated values optionally enclosing elements in parentheses (). The following declares a tuple type variable.

- The **empty tuple** is written as two parentheses containing nothing –

```
>>> tup1 = ();
>>> type(tup1)
Output: <class 'tuple'>
```

- To create a **tuple containing a single value** you have to include a comma, even though there is only one value. But at the end is not required if two or more elements are present.

```
>>> tup1=(12)    # (,) not present; tup1 is not a tuple
>>> tup1
Output: 12
>>> tup1=(12,) or >>> tup1= 12,    #comma is there;so tuple
>>> tup1
Output: (12,)
>>> type(tup1)
Output: <class 'tuple'>
>>> tup1=(12,21)    or >>> tup1=12,21
>>> tup1
Output: (12, 21)
>>> data1=(1,2.8,"Hello World") # tuple of int, float, string
>>> print(data1)
Output: (1, 2.8, 'Hello World')
>>> data2 = ("Book", [1, 2, 3]) # tuple of string and list
>>> print(data2)
Output: ('Book', [1, 2, 3])
>>> data3 = ((2, 3, 4), (1, 2, "hi")) #nested tuple
>>> print(my_data3)
Output: ((2, 3, 4), (1, 2, 'hi'))
```

tuple packing and unpacking

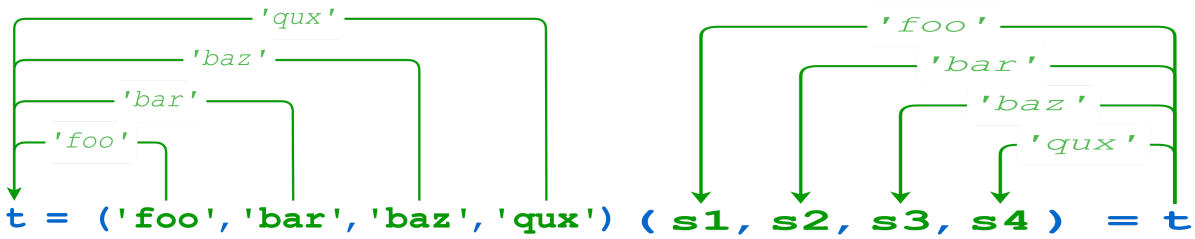
A literal tuple containing several items can be assigned to a single object:

```
>>> t = ('foo', 'bar', 'baz', 'qux')
```

When this occurs, it is as though the items in the tuple have been “packed” into the object.

```
>>> t[0]    #Output: ‘foo’
>>> t[2]    # Output: ,baz’
```

If that “packed” object is subsequently assigned to a new tuple, the individual items are “unpacked” into the objects in the tuple




```
>>> (s1, s2, s3, s4) = t
>>> s1    # Output: 'foo'
>>> s2    # Output: 'bar'
>>> my_tuple=3,4.6,"dog" or >>> my_tuple=(3,4.6,"dog")
>>> print(my_tuple)   # Output: (3, 4.6, 'dog')
>>> a, b, c = my_tuple
>>> print(a)      # Output: 3
>>> print(b)      # Output: 4.6
>>> print(c)      # Output: dog
>>> a = 1, 2, 3, 4
>>> _, x, y, _ = a
>>> print(x,y)    # Output: 2,3
```

Accessing Values of Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain values available at that index.

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**.

```
# Accessing tuple elements using indexing
     >>> my_tuple = ('p','e','r','m','i','t')
     >>> print(my_tuple[0])   #Output:  p
     >>> my_tuple[5]          #Output: 't'

# nested tuple
     n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
# nested index
     >>> print(n_tuple[0][3])       # Output: 's'
     >>> print(n_tuple[1][1])       # Output: 4
     >>> print(n_tuple[2][2])       # Output: 3
```

2. Negative Indexing

Like list and string 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.

```
# Negative indexing for accessing tuple elements
     >>> my_tuple = ('p', 'e', 'r', 'm', 'i', 't')
     >> print(my_tuple[-1])      # Output: 't'
     >>> print(my_tuple[-6])      # Output: 'p'
# Negative indexing for nested tuple
     >>> n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
     >>> print(n_tuple[-1][-2])  # Output: 2
     >>> print(n_tuple[-2][-3])  # Output: 8
     >>> n_tuple[-3][-4]         # Output: 'o'
```

3. Slicing

We can access a range of items in a tuple by using the slicing operator colon **:**.

```
# Accessing tuple elements using slicing
     >>> my_tuple = ('p','r','o','g','r','a','m','i','z')
# elements 2nd to 4th
     >>> print(my_tuple[1:4])      # Output: ('r', 'o', 'g')


# elements beginning to 2nd
     >>> print(my_tuple[:-7])     # Output: ('p', 'r')
# elements 8th to end
     >>> print(my_tuple[7:])     # Output: ('i', 'z')
# elements beginning to end
     >>> print(my_tuple[:])    # Output: ('p','r','o','g','r','a','m','i','z')
     >>> my_tuple[-4:-1]       # Output: ('a', 'm', 'i')
     >>> my_tuple[-7:-2:2]     # Output: ('o', 'r', 'm')
```

Changing/Deleting a Tuple

Unlike lists, the tuple items cannot be changed or deleted by using the **del** keyword as tuples are immutable. To delete an entire tuple, we can use the del keyword with the tuple name.

- **a new value can't be assigned in tuple**

```
>>> n_t=12,34,'abcd',89
>>> n_t    # Output: (12, 34, 'abcd', 89)
>>> n_t[1]=100 # Traceback (most recent call last):
                File "<stdin>", line 1, in <module>
      TypeError:'tuple' object does not support item assignment
```

- **an individual element/value can't be deleted in tuple**

```
>>> tuple1 = (1, 2, 3, 4, 5, 6)
>>> print(tuple1)    # Output: (1, 2, 3, 4, 5, 6)
>>> del tuple[0]   # Traceback (most recent call last):
                     File "<stdin>", line 1, in <module>
          TypeError: 'tuple' object doesn't support item deletion
```

As we know, Tuples are immutable in Python, hence data cannot be changed, but lists are mutable. Thus **lists present inside tuples (nested tuple) can be changed**.

- **Assigning value in a list that is element of tuple**

```
>>> Tuple = (3, 4.5, [4, 5, 6])
>>> print("Original Tuple is:", Tuple)
>>> Tuple[2][0] = 200
>>> print("Updated Tuple is:", Tuple)
```

Output:

Original Tuple is: (3, 4.5, [4, 5, 6])
Updated Tuple is: (3, 4.5, [200, 5, 6])

❖ **Deleting value from a list that is element of tuple**

```
>>> Tuple = (3, 4.5, [4, 5, 6])
>>> Tuple          Output:(3, 4.5, [4, 5, 6])
>>> del Tuple[2][1]
>>> Tuple     Output: (3, 4.5, [4, 6])
```

Reassigning value to the entire tuple or deleting the entire tuple is possible.

❖ **Assigning value to entire tuples**

```
>>> my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i',
'z')
>>> print(my_tuple)
Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
>>> my_tuple=("CEMK",1998,"Kolaghat",2020)
>>> print(my_tuple)
Output: ('CEMK', 1998, 'Kolaghat', 2020)
```

❖ **Deleting entire tuples**

```
>>> my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i',
'z')
>>> print(my_tuple)
Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
>>> del my_tuple    # Can delete an entire tuple
>>> print(my_tuple)
 Output: NameError: name 'my_tuple' is not defined
```

We **cannot use append(), extend(), remove () or pop ()** function in the Tuple as tuples are immutable.

Tuple Operations

Like string, tuple objects are also a sequence. Hence, the operators used with strings are also available for the tuple.

Operator +

The + operator returns a tuple containing all the elements of the first and the second tuple object.

```
>>> t1=(1,2,3)
>>> t2=(4,5,6)
>>> t1+t2
Output: (1, 2, 3, 4, 5, 6)
>>> t2+(7,)
Output: (4, 5, 6, 7)
```

Operator *

The * operator Concatenates multiple copies of the same tuple.

```
>>> t1=(1,2,3)
>>> t1*4
Output: (1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Operator ==

Compare operator (==) returns 1 if the two tuple are equal. It returns -1 if two tuples are not equal.

```
>>> tuple1 = ('a', 'b', 'c', 'd', 'e')
>>> tuple2 = ('1','2','3')
>>> tuple3 = ('a', 'b', 'c', 'e', 'd')
>>> tuple1==tuple3
Output: False
>>> tuple1==tuple1
Output:True
>>> tuple2==tuple1
Output:False
```

Operator [] (index operator)

The [] operator Returns the item at the given index. A negative index counts the position from the right side.

```
>>> t1=(1,2,3,4,5,6)
>>> t1[3]      # Output : 4
>>> t1[-2]     # Output : 5
```

Operator [:] (Slicing Operator)

The [:] operator returns the items in the range specified by three index operands separated by the : symbol.
If the first operand is omitted, the range starts from zero. If the second operand is omitted, the range goes up to the end of the tuple. If the 3rd operand is omitted then default stride is taken as 1.

```
>>> t1=(1,2,3,4,5,6)
>>> t1[1:3]      # Output : (2, 3)
>>> t1[3:]       # Output : (4, 5, 6)
>>> t1[:3]       # Output : (1, 2, 3)
>>> t1[:6:2]     # Output : (1, 3, 5)
```

Iterating a tuple

```
# tuple of fruits
    >>> my_tuple = ("Apple", "Orange", "Grapes", "Banana")
# iterating over tuple elements
    >>> for fruit in my_tuple:
        ...   print(fruit)
```

Output:

```
Apple
```

```
Orange
Grapes
Banana
```

Membership Test in Tuples

in: Checks whether an element exists in the specified tuple.

not in: Checks whether an element does not exist in the specified tuple.

```
>>> my_data = (11, 22, 33, 44, 55, 66, 77, 88, 99)
>>> print(my_data)
Output: (11, 22, 33, 44, 55, 66, 77, 88, 99)
>>> print(22 in my_data) or 22 in my_data
Output: True
>>> print(2 in my_data)
Output: False
>>> print(88 not in my_data)
Output: False
>>> print(101 not in my_data)
Output: True
```

Tuple Functions

any(): Returns True if any element present in a tuple and returns False if the tuple is empty

```
>>>Tuple = (3, 1, 4.5)
>>>print("Elements present in Tuple:", any(Tuple))
>>>Tuple1 = ()
>>>print("Elements present in Tuple1:", any(Tuple1))
```

Output: Elements present in Tuple: True
Elements present in Tuple1: False

min(): Returns smallest element (Integer) of the Tuple

```
>>> Tuple = (3, 5.6, 5, 8)
>>> print("Smallest element in the tuples is:", min(Tuple))
Output: Smallest element in the tuples is: 3
>>> Tuple1=('e','i','o','a','u')
>>> min(Tuple1)
Output: 'a'
```

max(): Returns largest element (Integer) of the Tuple

```
>>> Tuple = (3, 5.6, 5, 8)
>>> print("Largest element in the tuples is:", max(Tuple))
Output: Largest element in the tuples is: 8
>>> Tuple1=('e','i','o','a','u')
>>> max(Tuple1)
Output: 'u'
```

len(): Returns the length of the Tuple

```
>>> Tuple = (3, 5.6, 5, 8)
>>> print("Length of the tuple is:", len(Tuple))
Output:
Length of the tuple is: 4
```

sorted(): Used to sort all the elements of the Tuple and returns a list

```
>>> Tuple = (2, 3.5, 1, 6, 4)
>>> A=sorted(Tuple)
>>> A
Output: [1, 2, 3.5, 4, 6]
>>> Tuple1=('e','i','o','a','u')
>>> A=sorted(Tuple1)
```

```
>>> print(“Sorted character is:”,A)
Output: Sorted character is: ['a', 'e', 'i', 'o', 'u']
```

sum(): Returns sum of all elements (numbers) of the Tuples

```
>>> Num = (3, 5.1, 2, 9, 3.5)
>>> print("Sum of numbers in the tuples is:", sum(Num))
Output: Sum of numbers in the tuples is: 22.6
```

tuple(): Python tuple() converts a list of items or a sequence of items into tuples.

```
>>> aList = [123, 'xyz', 'zara', 'abc']
>>> aTuple = tuple(aList)
>>> print "Tuple elements : ", aTuple
Output: Tuple elements :  (123, 'xyz', 'zara', 'abc')
>>> btuple=tuple(“CEMK)
>>> print(btuple)
```

Tuple Methods

Methods that add items or remove items are not available with tuple. Only the following two methods (count and index) are available.

count() method returns the occurrence of an item within the tuple

index() method returns the first index of the item within the tuple

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

```
>>> print(my_tuple.count('p'))
Output: 2
>>> print(my_tuple.index('l'))
Output: 3
>>> my_tuple.index('p')
Output: 1
```