**10 Examples Of Python Tuples**

Data structures are crucial parts of any programming language. In order to create robust and well-performing products, one must know the data structures very well.

In this post, we will work on an important data structure of Python programming language and that is *tuple*.

Tuple is a collection of values separated by comma and enclosed in parenthesis. Unlike lists, tuples are immutable. The immutability can be considered as the identifying feature of tuples.

I will explain the features of tuples and operations on them with examples.

**1. Creating tuples**

Tuples consist of values in parenthesis and separated by comma.

a = (3, 4)print(type(a))  
<class 'tuple'>

Tuples can store values of different data types and duplicate values.

a = (3, 3, 'x', [1,2])print(a)  
(3, 3, 'x', [1, 2])print(type(a))  
<class 'tuple'>

We can also create tuples without using the parenthesis. A sequence of values separated by comma will create a tuple.

a = 3, 4, 5, 6print(type(a))  
<class 'tuple'>

**2. Creating a tuple with 0 or 1 element**

A tuple with zero element is just an empty one and created as follows:

a = ()print(type(a))  
<class 'tuple'>

However, there is a little trick to create a tuple with one element. You need to put a comma after the element. Otherwise, you will create a variable of the type of the value.

a = (3)  
print(type(a))  
<class 'int'>b = ([1,2])  
print(type(b))  
<class 'list'>

Let’s do the same but with using a comma after the values:

a = (3,)  
print(type(a))  
<class 'tuple'>b = ([1,2],)  
print(type(b))  
<class 'tuple'>

**3. Tuples are iterable**

Just like with lists, you can iterate over a tuple.

a = (1, 2, 3)  
for i in a:  
 print(i\*\*2)  
1  
4  
9

**4. Indexing and slicing**

Indexing and slicing tuples are the same as with lists.

a = (3, 5, 'x', 5)  
print(f'The first element of tuple a is {a[0]}')  
print(f'The last element of tuple a is {a[-1]}')The first element of tuple a is 3  
The last element of tuple a is 5

And slicing:

a = (1, 2, 5, 6, 7)print(a[-2:])  
(6, 7)print(a[:3])  
(1, 2, 5)

**5. Tuples are immutable but can have mutable elements**

The immutability might be the most identifying feature of tuples. We cannot assign to the individual items of a tuple.

a = (3, 5, 'x', 5)  
a[0] = 7 #error

However, tuples can contain mutable elements such as lists.

a = ([1,2], ['x', 'y'])a[0][0] = 99  
a[1][0] = 't'print(a)  
([99, 2], ['t', 'y'])

**6. Sort vs sorted**

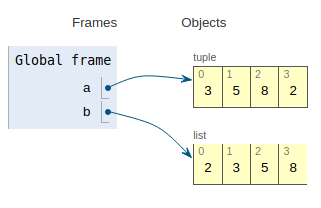
Since tuples are immutable, we cannot sort them:

a = (3, 0, 2)a.sort() #error  
'tuple' object has no attribute 'sort'

However, the sorted function can take a tuple as argument and returns a sorted list of the values in the tuple. Please note that it does not return a sorted tuple. The type of the return variable is list.

a = (3, 5, 8, 2)  
b = sorted(a)print(b)  
[2, 3, 5, 8]print(type(b))  
<class 'list'>

https://miro.medium.com/max/60/1*UMy9rZf6sOy08iUF1aMa0w.png?q=20



**7. Length of a tuple**

The len function can be used to get the length of a tuple.

a = (3, 0, 2)  
len(a)  
3

**8. Count and index methods**

Tuples have count and index methods. The count method returns the number of occurrences of a value in the tuple.

a = (1, 'x', 1, 1, 'x')print(a.count('x'))  
2print(a.count(1))  
3

The index method returns the index of a value in the tuple.

a = (1, 'x', 3, 5, 'x')print(a.index('x'))  
1print(a.index(1))  
0

**Note**: If a value exists multiple times in a tuple, the index method returns the index of the very first occurrence.

**9. Adding tuples together**

The ‘+’ operator can be used to add tuples together.

a = (1, 2)  
b = ('x', 'y')  
c = a + bprint(c)  
(1, 2, 'x', 'y')

**10. Functions that return multiple values**

One of the most common use cases of tuples is with functions that return multiple values. The following function takes an array and returns the sum and the count of items in it.

def count\_sum(arr):  
 count = len(arr)  
 sum = arr.sum()  
 return count, sum

What this function returns is a tuple with 2 items:

import numpy as np  
arr = np.random.randint(10, size=8)a = count\_sum(arr)print(a)  
(8, 39)print(type(a))  
<class 'tuple'>

**Bonus**

Both tuples and lists are collection of values. The most significant difference is the immutability.

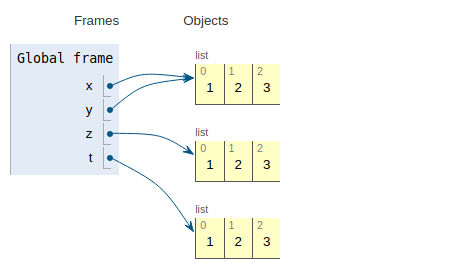
Due to the immutability, copying tuples and lists are different. We need to more careful when copying lists since they are mutable.

We have three ways to copy a list and assign to a new variable:

x = [1, 2, 3]y = x  
z = x[:]  
t = x.copy()

The lists y, z, and t contain the same values as x. However, y points to the values of x whereas z and t are totally different lists.

https://miro.medium.com/max/60/1*KpNVTgKfSYw1zg6XSvxv0A.png?q=20



Thus, any change in x will cause a change in y as well.

x.append(4)print(y)  
[1, 2, 3, 4]print(t)  
[1, 2, 3]

We need to be very careful when copying lists. However, we should not have the same concern with tuples because they are immutable.

When you copy a tuple and assign it to a new variable, they all point to the same values in the memory.

a = (1, 2, 3)b = a  
c = a[:]

https://miro.medium.com/max/60/1*mESVanmd77EPq9Z3iY_RTA.png?q=20

