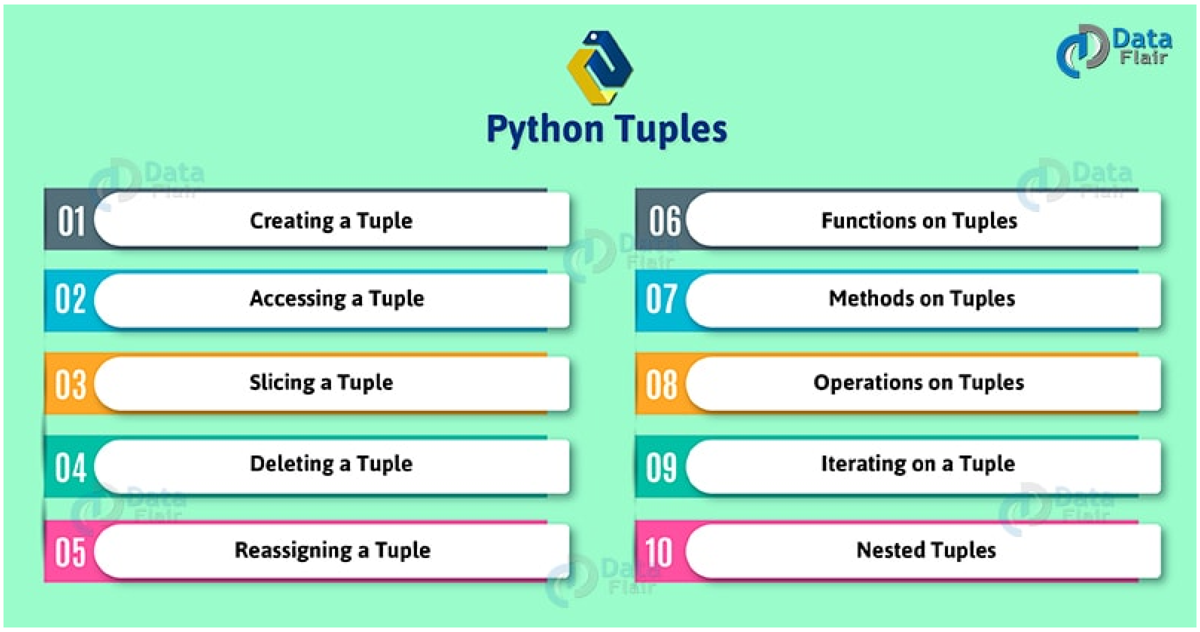
Python Tuples with Syntax and Examples

data-flair.training/blogs/python-tuples-syntax-examples

Contents

1. Python Tuples

Python provides a range of constructs to deal with items. These include **python lists**, dictionaries, sets, tuples, and many more. It also supports in-built functions and methods that we can apply on these constructs. In this tutorial, we will rather take a deeper look at **Python tuples**. First, let's look at what a Python tuple is and then we will understand what is Python list of tuples and other topics.



Python Tuples

2. An Introduction to Python Tuples

Python Tuples are like like a list. It can hold a sequence of items. The difference is that it is, however, immutable. Let's learn the syntax to create a tuple.

3. Creating a Python Tuple

To declare a tuple, you must type a list of items separated by commas, inside parentheses. Then assign it to a variable.

```
>>> percentages=(90,95,89)
```

You should use a tuple when you don't want to change just an item in future.

a. Python Tuples Packing

You can also create a Python tuple without parentheses. This is called tuple packing.

```
>>> b=1, 2.0, 'three'
```

b. Python Tuples Unpacking

Python tuples unpacking is when you assign values from a tuple to a sequence of **variables in python**.

```
>>> percentages=(99,95,90,89,93,96)
>>> a,b,c,d,e,f=percentages
```

3. >>> c

90

You can do the same to a list.

c. Creating a tuple with a single item

Until now, we have seen how easy it is to declare a tuple. But when you do so with just one element, it may create some problems. Let’s take a look at it.

1. >>> a=(1)
2. >>>type(a)

<class ‘int’>

Wasn’t the type() method supposed to return class ‘tuple’?

To get around this, we add a comma after the item.

1. >>> a=(1,)
2. >>>type(a)

<class ‘tuple’>

Problem solved. And as we saw in tuple packing, we can skip the parentheses here.

1. >>> a=1,
2. >>>type(a)

<class ‘tuple’>

Also, like a list, a tuple may contain items of different types.

>>> a=(1,2.0,'three')

4. Accessing Python Tuples

a. Accessing the entire tuple

To access a tuple in python, just type its name.

>>> percentages

(90, 95, 89)

Or, pass it to the print statement.

>>>print(percentages)

(90, 95, 89)

b. Accessing a single item

To get a single item from a tuple, use its index in square brackets. Indexing begins at 0.

```
>>> percentages[1]
```

```
95
```

5. Slicing a Tuple

If you want a part(slice) of a tuple, use the slicing operator [].

```
>>> percentages=(99,95,90,89,93,96)
```

a. Positive Indices

When using positive indices, we traverse the list from the left.

```
>>> percentages[2:4]
```

```
(90, 89)
```

This prints out items from index 2 to index 3 (4-1) (items third to fourth).

```
>>> percentages[:4]
```

```
(99, 95, 90, 89)
```

This prints out items from the beginning to the item at index 3.

```
>>> percentages[4:]
```

```
(93, 96)
```

This prints out items from index 4 to the end of the list.

```
>>> percentages[2:2]
```

```
()
```

However, this returns an empty tuple.

b. Negative indexing

Now, let’s look at negative indexing. Unlike positive indexing, it begins traversing from the right.

```
>>> percentages[:-2]
```

```
(99, 95, 90, 89)
```

This prints out the items from the tuple’s beginning to two items from the end.

```
>>> percentages[-2:]
```

```
(93, 96)
```

This prints out items from two items from the end to the end.

```
>>> percentages[2:-2]

(90, 89)
```

This prints out items from index 2 to two items from the end.

```
>>> percentages[-2:2]

()
```

This last piece of code, however, returns an empty tuple. This is because the start(-2) is behind the end(2) in this case.

Lastly, when you provide no indices, it prints the whole tuple.

```
>>> percentages[:]

(99, 95, 90, 89, 93, 96)
```

6. Deleting a Python Tuple

As we discussed above, a tuple is immutable. This also means that you can’t delete just a part of it. You must delete an entire tuple, if you may.

1. >>> del percentages[4]
2. Traceback(most recent call last):
3. File "<pyshell#19>", line 1, in <module>
4. del percentages[4]

TypeError: ‘tuple’ object doesn’t support item deletion

So, deleting a single element didn’t work. Let’s try deleting a slice.

1. >>> del percentages[2:4]
2. Traceback(most recent call last):
3. File "<pyshell#20>", line 1, in <module>
4. del percentages[2:4]

TypeError: ‘tuple’ object does not support item deletion

As you can see, that didn’t work either. Now, let’s try deleting the entire tuple.

1. >>> del percentages
2. >>> percentages
3. Traceback(most recent call last):
4. File "<pyshell#40>", line 1, in <module>
5. percentages

NameError: name ‘percentages’ is not defined

We see that the tuple has successfully been deleted.

7. Reassigning Tuples in Python

As we discussed, a tuple is immutable. So let’s try changing a value. But before that, let’s take a new tuple with a list as an item in it.

>>> my_tuple=(1,2,3,[4,5])

Now, let’s try changing the list [4,5]. Its index is 3.

1. >>> my_tuple[3]=6
2. Traceback(most recent call last):
3. File "<pyshell#43>", line 1, in <module>
4. my_tuple[3]=6

TypeError: ‘tuple’ object does not support item assignment

See, that failed. Now how about changing an element from the same list]?

1. >>> my_tuple[3][0]=6
2. >>> my_tuple

(1, 2, 3, [6, 5])

This worked without a flaw. So we can see that while tuples are immutable, a mutable item that it holds may be reassigned.

8. Functions on Tuples

A lot of functions that work on lists work on tuples too. A function applies on a construct, and returns a result. It does not modify the construct. Let’s see what we can do.

a. len()

Like a list, a tuple is of a certain length. The len() function returns its length.

>>> my_tuple

(1, 2, 3, [6, 5])

>>>len(my_tuple)

4

It returned 4, not 5, because the list counts as 1.

b. max()

It returns the item from the tuple with the highest value.

We can’t apply this function on the tuple my_tuple, because ints cannot be compared to a list. So let’s take yet another tuple.

1. >>> a=(3,1,2,5,4,6)

2. >>>max(a)

6

Let’s try that on strings.

>>>max(('Hi','hi','Hello'))

‘hi’

‘hi’ is the greatest out of these, because h has the highest ASCII value among h and H.

But you can’t compare an int and a string.

1. >>>max(('Hi',9))
2. Traceback(most recent call last):
3. File "<pyshell#59>", line 1, in <module>
4. max(('Hi',9))

TypeError: ‘>’ not supported between instances of ‘int’ and ‘str’

c. min()

Like the max() function, the min() returns the item with the lowest values.

>>>min(a)

1

As you can see, 1 is the smallest item in the tuple.

d. sum()

This function returns the arithmetic sum of all the items in the tuple.

>>>sum(a)

21

However, you can’t apply this function on a tuple with strings.

1. >>>sum(('1','2','3'))
2. Traceback(most recent call last):
3. File "<pyshell#57>", line 1, in <module>
4. sum(('1','2','3'))

TypeError: unsupported operand type(s) for +: ‘int’ and ‘str’

e. any()

If even one item in the tuple has a Boolean value of True, then this function returns True. Otherwise, it returns False.

>>>any(('','0',''))

True

The string ‘0’ does have a Boolean value of True. If it was rather the integer 0, it would’ve returned False.

>>>any(('',0,''))

False

f. all()

Unlike any(), all() returns True only if all items have a Boolean value of True. Otherwise, it returns False.

>>>all(('1',1,True,''))

False

g. sorted()

This function returns a sorted version of the tuple. The sorting is in ascending order, and it doesn’t modify the original tuple.

>>>sorted(a)

[1, 2, 3, 4, 5, 6]

h. tuple()

This function converts another construct into a tuple. Let’s look at some of those.

1. >>>list1=[1,2,3]
2. >>>tuple(list1)

(1, 2, 3)

1. >>> string1="string"
2. >>>tuple(string1)

(‘s’, ‘t’, ‘r’, ‘i’, ‘n’, ‘g’)

How well would it work with sets?

1. >>> set1={2,1,3}
2. >>>tuple(set1)

(1, 2, 3)

>>> set1

{1, 2, 3}

As we can see, when we declared a set as 2,1,3, it automatically reordered itself to 1,2,3. Furthermore, creating a tuple from it returned the new tuple in the new order, that is, ascending order.

9. Methods on Python Tuples

A method is a sequence of instructions to perform on something. Unlike a function, it does modify the construct on which it is called. You call a method using the dot **operator in python.** Let's learn about the two in-built methods of Python.

a. index()

This method takes one argument and returns the index of the first appearance of an item in a tuple. Let's take a new tuple.

1. >>> a=(1,2,3,2,4,5,2)
2. >>> a.index(2)

1

As you can see, we have 2s at indices 1, 3, and 6. But it returns only the first index.

b. count()

This method takes one argument, and returns the number of times an item appears in the tuple.

```
>>> a.count(2)
```

```
3
```

10. Operations on Tuples in Python

Now, we will look at the operations that we can perform on tuples.

a. Membership

We can apply the 'in' and 'not in' operators on items. This tells us whether they belong to the tuple.

```
>>>'a' in tuple("string")
```

```
False
```

```
>>>'x' not in tuple("string")
```

```
True
```

b. Concatenation

Like we've previously discussed on several occasions, concatenation is the act of joining. We can join two tuples using the concatenation operator '+'.

```
>>>(1,2,3)+(4,5,6)
```

```
(1, 2, 3, 4, 5, 6)
```

Other arithmetic operations do not apply on a tuple.

c. Logical

All the logical operators (like >,>=,..) can be applied on a tuple.

```
>>>(1,2,3)>(4,5,6)
```

```
False
```

```
>>>(1,2)==('1','2')
```

```
False
```

As is obvious, the ints 1 and aren’t equal to the strings ‘1’ and ‘2’. Likewise, it returns False.

d. Identity

Remember the ‘is’ and ‘is not’ operators we discussed about in our tutorial on Python Operators? Let’s try that on tuples.

```
1. >>> a=(1,2)
2. >>>(1,2) is a
```

```
False
```

That did not make sense, did it? So what really happened? Well, in Python, two tuples or lists do not have the same identity. In other words, they are two different tuples or lists. As a result, it returns False.

11. Iterating on a Python Tuple

You can iterate on a tuple using a for loop like you would iterate on a list.

```
1. >>>for i in(1,3,2):
2. print(i)
```

```
1

3

2
```

12. Nested Tuples

Finally, we will learn about nesting tuples. You may remember how we can nest lists. Due to the similarities of a tuple to a list, we do the same with tuples.

```
>>> a=((1,2,3),(4,(5,6)))
```

Suppose we want to access the item 6. For that, since we use indices, we write the following code.

```
>>> a[1][1][1]

6
```

A tuple may also contain other constructs, especially, lists. After all, it is a collection of items, and items can be anything.

```
>>>(1,2,[3,4])

(1, 2, [3, 4])
```

This was all on Python Tuples Tutorial.

13. Conclusion: Python Tuples

Summing up, we now know quite a bit about Python tuples. Today, we learned about creating tuples. In that, we looked at tuple packing and unpacking, and how to create a tuple with just one item. Then we talked about how to access, slice, delete, and reassign a tuple. After that, we looked at the in-built functions and methods that we can call on a tuple. Lastly, we learned about the operations we can perform on a Python tuple, how to iterate on it, and nested tuples. Try them in the shell and leave your honest comments.Hope you like the Python Tuples tutorial.