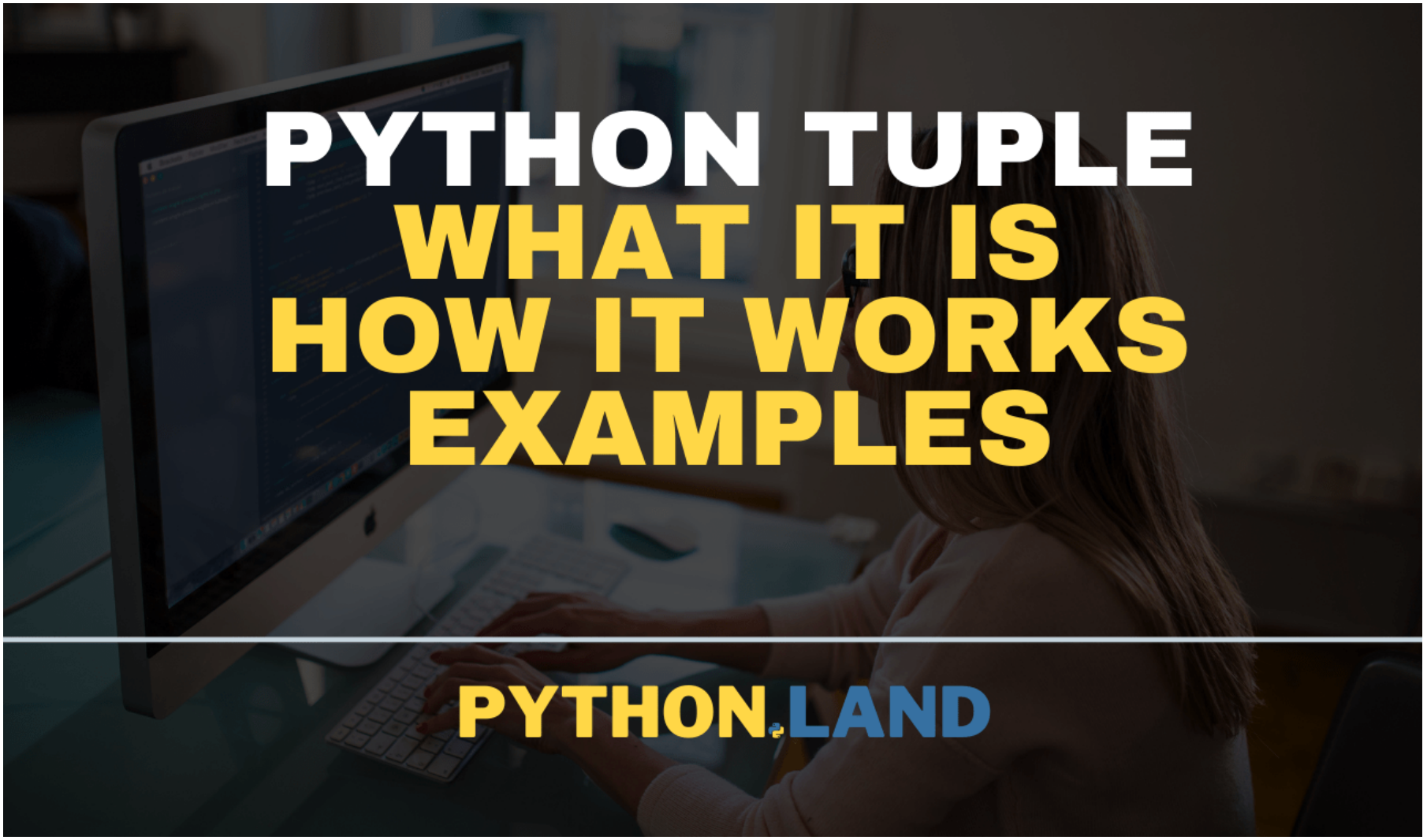
Previous: Python Float

Next: Python List ›

Home » Python Data Types » Python Tuple: How to Create, Use, and Convert



Python Tuple: How to Create, Use, and Convert

April 23, 2022

A Python tuple is one of Python's three built-in sequence data types, the others being lists and range objects. A Python tuple shares a lot of properties with the more commonly known Python list:

- It can hold multiple values in a single variable
- It's ordered: the order of items is preserved
- A tuple can have duplicate values
- It's indexed: you can access items numerically
- A tuple can have an arbitrary length

But there are significant differences:

- A tuple is immutable; it can not be changed once you have defined it.
- A tuple is defined using *optional* parentheses () instead of square brackets []
- Since a tuple is immutable, it can be hashed, and thus it can act as the key in a dictionary

Table of Contents [hide]

Beginners Python Course (2023) **Modules, Packages, And Virtual Environments (2023)**

Creating a Python tuple

We create tuples from individual values using optional parentheses (round brackets) like this:

```
>>> my_numbers = (1, 2, 3)
>>> the_same = 1, 2, 3
>>> my_strings = ('Hello', 'World')
>>> my_mixed_tuple = ('Hello', 123, True)
```

Like everything in Python, tuples are objects and have a class that defines them. We can also create a tuple by using the `tuple()` constructor from that class. It allows any Python iterable type as an argument. In the following example, we create a tuple from a list:

```
>>> tuple([0,1,2])
(0, 1, 2)
>>>
```

Now you know how to convert a Python list to a tuple as well!

Which method is best?

It's not always easy for Python to infer if you're using regular parentheses or if you're trying to create a tuple. To demonstrate, let's define a tuple holding only one item:

```
>>> t = (1)
1
>>> t = (1, )
(1,)
```

Python sees the number one, surrounded by useless parentheses on the first try, so Python strips down the expression to the number 1. However, we added a comma in the second try, explicitly signaling to Python that we are creating a tuple with just one element.

A tuple with just one item is useless for most use cases, but it demonstrates how Python recognizes a tuple: because of the comma.

If we can use `tuple()`, why is there a second method as well? The other notation is more concise, but it also has its value because you can use it to unpack multiple lists into a tuple in this way concisely:

```
>>> l1 = [1, 2, 3]
>>> l2 = [4, 5, 6]
>>> t = (*l1, *l2)
>>> t
(1, 2, 3, 4, 5, 6)
```

The leading * operator unpacks the lists into individual elements. It's as if you would have typed them individually at that spot. This unpacking trick works for all iterable types if you were wondering!

Multiple assignment using a Python tuple

You've seen something called tuple unpacking in the previous topic. There's another way to unpack a tuple, called multiple assignment. It's something that you see used a lot, especially when returning data from a function, so it's worth taking a look at this.

Multiple assignment works like this:

```
>>> person = ('Erik', 38, True)
>>> name, age, registered = person
>>> name
'Erik'
>>> _
```

Like using the *, this type of unpacking works for all iterable types in Python, including lists and strings.

As I explained in the Python trick on returning multiple values from a Python function, unpacking tuples works great in conjunction with a function that returns multiple values. It's a neat way of returning more than one value without having to resort to data classes or dictionaries:

```
def get_user_by_id(userid):
    # fetch user from database
    # ....
    return name, age

name, age = get_user_by_id(4)
```

Indexed access

We can access a tuple using index numbers like `[0]` and `[1]`:

```
>>> my_mixed_tuple = 'Hello', 123, True
>>> my_mixed_tuple[0]
'Hello'
>>> my_mixed_tuple[2]
```

```
True
>>> _
```

Append to a Python Tuple

Because a tuple is immutable, **you can not append data to a tuple after creating it**. For the same reason, you can't remove data from a tuple either. You can, of course, create a new tuple from the old one and append the extra item(s) to it this way:

```
>>> t1 = (1, 2, 3)
>>> t = (*t1, 'Extra', 'Items')
>>> t
(1, 2, 3, 'Extra', 'Items')
```

What we did was unpack `t1`, create a new tuple with the unpacked values and two different strings and assign the result to `t` again.

Get tuple length

The `len()` function works on Python tuples just like it works on all other iterable types like lists and strings:

```
>>> t = 1, 2, 3
>>> len(t)
3
```

Python Tuple vs List

The most significant difference between a Python tuple and a Python list is that a List is mutable, while a tuple is not. After defining a tuple, you can not add or remove values. In contrast, a list allows you to add or remove values at will. This property can be an advantage; you can see it as write protection. If a piece of data is not meant to change, using a tuple can prevent errors. After all, six months from now, you might have forgotten that you should not change the data. Using a tuple prevents mistakes.

Another advantage is that tuples are faster, or at least that is what people say. I have not seen proof, but it makes sense. Since it's an immutable data type, a tuple's internal implementation can be simpler than lists. After all, they don't need ways to grow larger or insert elements at random positions, which usually is implemented as a linked list. From what I understand, a tuple uses a simple array-like structure in the CPython implementation.

Python Tuple vs Set

The most significant difference between tuples and Python sets is that a tuple can have duplicates while a set can't. The entire purpose of a set is its inability to contain duplicates. It's an excellent tool for deduplicating your data.

Converting Python tuples

Convert Tuple to List

Python lists are mutable, while tuples are not. If you need to, you can convert a tuple to a list with one of the following methods.

The cleanest and most readable way is to use the `list()` constructor:

```
>>> t = 1, 2, 3
>>> list(t)
[1, 2, 3]
```

A more concise but less readable method is to use unpacking. This unpacking can sometimes come in handy because it allows you to unpack multiple tuples into one list or add some extra values otherwise:

```
>>> t = 1, 2, 3
>>> l = [*t]
>>> l
[1, 2, 3]
>>> l = [*t, 4, 5, 6]
>>> l
[1, 2, 3, 4, 5, 6]
```

Convert tuple to set

Analogous to the conversion to a list, we can use `set()` to convert a tuple to a set:

```
>>> t = (1, 2, 3)
>>> s = set(t)
>>> s
```

```
{1, 2, 3}
```

Here, too, we can use unpacking:

```
>>> s = {*t}
>>> s
{1, 2, 3}
```

Convert tuple to string

Like most objects in Python, a tuple has a so-called dunder method, called `__str__`, which converts the tuple into a string. When you want to print a tuple, you don't need to do so explicitly. Python's print function will call this method on any object that is not a string. In other cases, you can use the `str()` constructor to get the string representation of a tuple:

```
>>> t = (1, 2, 3)
>>> print(t)
(1, 2, 3)
>>> str(t)
'(1, 2, 3)'
```


‹ Previous: Python Float

Next: Python List