

Python Tuple

```
mytuple = ("apple", "banana", "cherry")
```

Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](#), [Set](#), and [Dictionary](#), all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

Example:

Create a Tuple:

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple)
```

[Try it Yourself »](#)

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value:

Example:

Tuples allow duplicate values:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
print(thistuple)
```

Tuple Length

To determine how many items a tuple has, use the len() function:

Example

Print the number of items in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Create Tuple With One Item:

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example:

One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))
```

#NOT a tuple

```
thistuple = ("apple")
print(type(thistuple))
```

Tuple Items - Data Types

Tuple items can be of any data type:

Example

String, int and boolean data types:

```
tuple1 = ("apple", "banana", "cherry")
```

```
tuple2 = (1, 5, 7, 9, 3)
```

```
tuple3 = (True, False, False)
```

A tuple can contain different data types:

Example

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")
```

type()

From Python's perspective, tuples are defined as objects with the data type 'tuple':

```
<class 'tuple'>
```

Example

What is the data type of a tuple?

```
mytuple = ("apple", "banana", "cherry")
```

```
print(type(mytuple))
```

The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

Example

Using the tuple() method to make a tuple:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
```

```
print(thistuple)
```

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.

- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered** and changeable. No duplicate members.

*Set *items* are unchangeable, but you can remove and/or add items whenever you like.

**As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Python - Access Tuple Items

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Note: The first item has index 0.

Negative Indexing

Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the tuple:

```
thistuple = ("apple", "banana", "cherry")  
print(thistuple[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT included, "kiwi":

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[:4])
```

By leaving out the end value, the range will go on to the end of the tuple:

Example

This example returns the items from "cherry" and to the end:

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:])
```

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[-4:-1])
```

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

Example

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
    print("Yes, 'apple' is in the fruits tuple")
```

Python - Update Tuples

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

But there are some workarounds.

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)
```

```
print(x)
```

Add Items

Since tuples are immutable, they do not have a built-in `append()` method, but there are other ways to add items to a tuple.

1. **Convert into a list:** Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

Example

Convert the tuple into a list, add "orange", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")  
y = list(thistuple)  
y.append("orange")  
thistuple = tuple(y)
```