

# Python - Tuple Methods

## Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
<a href="#">count()</a>	Returns the number of times a specified value occurs in a tuple
<a href="#">index()</a>	Searches the tuple for a specified value and returns the position of where it was found

# Python Tuples

```
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)
```

Output:

```
('apple', 'banana', 'cherry')
```

## 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 tuple are indexed, tuples can have items with the same value:

### Example

Tuples allow duplicate values:

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

Output:

```
('apple', 'banana', 'cherry', 'apple', 'cherry')
```

## 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))
```

Output:

3

## 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))
```

Output:

<class 'tuple'>

<class 'str'>

## 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))
```

Output:

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

# 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)
```

Output:

```
('apple', 'banana', 'cherry')
```

# 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 and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered and changeable. No duplicate members.

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])
```

Output:

banana

**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])
```

Output:

cherry

## 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])
```

Output:

('cherry', 'orange', 'kiwi')

**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])
```

Output:

```
('apple', 'banana', 'cherry', 'orange')
```

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

## Example

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

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

Output:

```
('cherry', 'orange', 'kiwi', 'melon', 'mango')
```

# Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

## 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])
```

Output:

```
('orange', 'kiwi', 'melon')
```

# 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")
```

Output:

```
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)
```

Output:

```
('apple', 'kiwi', 'cherry')
```

## Add Items

Once a tuple is created, you cannot add items to it.

### Example

You cannot add items to a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple.append("orange") # This will raise an error
print(thistuple)
```

Output:

Error

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)
```

# Remove Items

**Note:** You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

## Example

Convert the tuple into a list, remove "apple", and convert it back into a tuple:

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

Or you can delete the tuple completely:

## Example

The `del` keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Output:

Error



# Python - Unpack Tuples

## Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

### Example

Packing a tuple:

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

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

### Example

Unpacking a tuple:

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

```
(green, yellow, red) = fruits
```

```
print(green)
print(yellow)
print(red)
```

Output:

apple

banana

cherry

**Note:** The number of variables must match the number of values in the tuple, if not, you must use an asterix to collect the remaining values as a list.

## Using Asterix\*

If the number of variables is less than the number of values, you can add an \* to the variable name and the values will be assigned to the variable as a list:

### Example

Assign the rest of the values as a list called "red":

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
```

```
(green, yellow, *red) = fruits
```

```
print(green)
print(yellow)
print(red)
```

Output:

apple

banana

['cherry', 'strawberry', 'raspberry']

If the asterix is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

## Example

Add a list of values the "tropic" variable:

```
fruits = ("apple", "mango", "papaya", "pineapple", "cherry")
```

```
(green, *tropic, red) = fruits
```

```
print(green)
print(tropic)
print(red)
```

Output:

apple

banana

['cherry', 'strawberry', 'raspberry']

# Python - Loop Tuples

## Loop Through a Tuple

You can loop through the tuple items by using a `for` loop.

### Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
    print(x)
```

Output:

apple

banana

cherry

Learn more about `for` loops in our [Python For Loops](#) Chapter.

## Loop Through the Index Numbers

You can also loop through the tuple items by referring to their index number.

Use the `range()` and `len()` functions to create a suitable iterable.

### Example

Print all items by referring to their index number:

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
    print(thistuple[i])
```

Output:

apple

banana

cherry

## Using a While Loop

You can loop through the list items by using a `while` loop.

Use the `len()` function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

## Example

Print all items, using a `while` loop to go through all the index numbers:

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
    print(thistuple[i])
    i = i + 1
```

Output:

apple

banana

cherry

Learn more about `while` loops in our [Python While Loops](#) Chapter.

# Python - Join Tuples

## Join Two Tuples

To join two or more tuples you can use the `+` operator:

### Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)
```

```
tuple3 = tuple1 + tuple2  
print(tuple3)
```

Output:

```
('a', 'b', 'c', 1, 2, 3)
```

## Multiply Tuples

If you want to multiply the content of a tuple a given number of times, you can use the `*` operator:

### Example

Multiply the fruits tuple by 2:

```
fruits = ("apple", "banana", "cherry")  
mytuple = fruits * 2
```

```
print(mytuple)
```

Output:

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

## Exercise 1:

Use the correct syntax to print the first item in the `fruits` tuple.

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

## Exercise 2:

Use the correct syntax to print the number of items in the `fruits` tuple.

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

## Exercise 3:

Use negative indexing to print the last item in the tuple.

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

## Exercise 4:

Use a range of indexes to print the third, fourth, and fifth item in the tuple.

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