

# Python from Scratch

## Python Tuples

### Lesson 11

- **Python Tuples**
  - Tuple
  - Tuple Items
  - Ordered
  - Unchangeable
  - Allow Duplicates
  - Tuple Length
  - Create Tuple With One Item
  - Tuple Items - Data Types
  - type()
  - The tuple() Constructor
  - Python Collections (Arrays)
- **Access Tuples**
  - Access Tuple Items
  - Negative Indexing
  - Range of Indexes
  - Range of Negative Indexes
  - Check if Item Exists
- **Update Tuples**
  - Change Tuple Values
  - Add Items
  - Remove Items
- **Unpack Tuples**
  - Unpacking a Tuple
  - Using Asterisk\*
- **Loop Tuples**
  - Loop Through a Tuple
  - Loop Through the Index Numbers
  - Using a While Loop
- **Join Tuples**
  - Join Two Tuples
  - Multiply Tuples
- **Python - Tuple Methods**
- **Python - Tuple Exercises**

# 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)
```

---

## 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 list:

### Example



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

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

## 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])
```

---

## 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 build-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)
```



2. **Add tuple to a tuple.** You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

### Example



Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("apple", "banana", "cherry")  
y = ("orange",)  
thistuple += y  
  
print(thistuple)
```

**Note:** When creating a tuple with only one item, remember to include a comma after the item, otherwise it will not be identified as a tuple.

## 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
```

## 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)
```

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

## Using Asterisk\*

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)
```

If the asterisk 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)
```

## 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)
```

---

### 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])
```

## Using a While Loop

You can loop through the tuple 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 referring 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
```

## 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)
```

---

### 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)
```



## Python - Tuple Methods

### Tuple Methods

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

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

## Python - Tuple Exercises

### Test Yourself With Exercises

Now you have learned a lot about tuples, and how to use them in Python.

Are you ready for a test?

Try to insert the missing part to make the code work as expected:

#### Exercise:

Print the first item in the **fruits** tuple.

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