A tuple in Python is similar to a [list](https://www.programiz.com/python-programming/list). The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

**Creating a Tuple**

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, [string](https://www.programiz.com/python-programming/string), etc.).

# Different types of tuples

# Empty tuple

my\_tuple = ()

print(my\_tuple)

# Tuple having integers

my\_tuple = (1, 2, 3)

print(my\_tuple)

# tuple with mixed datatypes

my\_tuple = (1, "Hello", 3.4)

print(my\_tuple)

# nested tuple

my\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

print(my\_tuple)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

()

(1, 2, 3)

(1, 'Hello', 3.4)

('mouse', [8, 4, 6], (1, 2, 3))

In the above example, we have created different types of tuples and stored different data items inside them.

As mentioned earlier, we can also create tuples without using parentheses:

my\_tuple = 1, 2, 3

my\_tuple = 1, "Hello", 3.4

**Create a Python Tuple With one Element**

In Python, creating a tuple with one element is a bit tricky. Having one element within parentheses is not enough.

We will need a trailing comma to indicate that it is a tuple,

var1 = ("Hello") # string

var2 = ("Hello",) # tuple

We can use the type() function to know which class a variable or a value belongs to.

var1 = ("hello")

print(type(var1)) # <class 'str'>

# Creating a tuple having one element

var2 = ("hello",)

print(type(var2)) # <class 'tuple'>

# Parentheses is optional

var3 = "hello",

print(type(var3)) # <class 'tuple'>

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Here,

* ("hello") is a string so type() returns str as class of var1 i.e. <class 'str'>
* ("hello",) and "hello", both are tuples so type() returns tuple as class of var1 i.e. <class 'tuple'>

**Access Python Tuple Elements**

Like a [list](https://www.programiz.com/python-programming/list), each element of a tuple is represented by index numbers **(0, 1, ...)** where the first element is at index **0**.

We use the index number to access tuple elements. For example,

**1. Indexing**

We can use the index operator [] to access an item in a tuple, where the index starts from 0.

So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range(6,7,... in this example) will raise an IndexError.

The index must be an integer, so we cannot use float or other types. This will result in TypeError.

Likewise, nested tuples are accessed using nested indexing, as shown in the example below.

# accessing tuple elements using indexing

letters = ("p", "r", "o", "g", "r", "a", "m", "i", "z")

print(letters[0]) # prints "p"

print(letters[5]) # prints "a"

[Run Code](https://www.programiz.com/python-programming/online-compiler)

In the above example,

* letters[0] - accesses the first element
* letters[5] - accesses the sixth element

**2. Negative Indexing**

Python allows negative indexing for its sequences.

The index of **-1** refers to the last item, **-2** to the second last item and so on. For example,

# accessing tuple elements using negative indexing

letters = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

print(letters[-1]) # prints 'z'

print(letters[-3]) # prints 'r'

[Run Code](https://www.programiz.com/python-programming/online-compiler)

In the above example,

* letters[-1] - accesses last element
* letters[-3] - accesses third last element

**3. Slicing**

We can access a range of items in a tuple by using the slicing operator colon :.

# accessing tuple elements using slicing

my\_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# elements 2nd to 4th index

print(my\_tuple[1:4]) # prints ('r', 'o', 'g')

# elements beginning to 2nd

print(my\_tuple[:-7]) # prints ('p', 'r')

# elements 8th to end

print(my\_tuple[7:]) # prints ('i', 'z')

# elements beginning to end

print(my\_tuple[:]) # Prints ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

('r', 'o', 'g')

('p', 'r')

('i', 'z')

('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

Here,

* my\_tuple[1:4] returns a tuple with elements from index **1** to index **3**.
* my\_tuple[:-7] returns a tuple with elements from beginning to index **2**.
* my\_tuple[7:] returns a tuple with elements from index **7** to the end.
* my\_tuple[:] returns all tuple items.

**Note**: When we slice lists, the start index is inclusive but the end index is exclusive.

**Python Tuple Methods**

In Python ,methods that add items or remove items are not available with tuple. Only the following two methods are available.

Some examples of Python tuple methods:

my\_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my\_tuple.count('p')) # prints 2

print(my\_tuple.index('l')) # prints 3

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Here,

* my\_tuple.count('p') - counts total number of 'p' in my\_tuple
* my\_tuple.index('l') - returns the first occurrence of 'l' in my\_tuple

**Iterating through a Tuple in Python**

We can use the [for loop](https://www.programiz.com/python-programming/for-loop) to iterate over the elements of a tuple. For example,

languages = ('Python', 'Swift', 'C++')

# iterating through the tuple

for lang in languages:

print(language)

[Run Code](https://www.programiz.com/python-programming/online-compiler)

**Output**

Python

Swift

C++

**Check if an Item Exists in the Python Tuple**

We use the in keyword to check if an item exists in the tuple or not. For example,

languages = ('Python', 'Swift', 'C++')

print('C' in languages) # False

print('Python' in languages) # True

[Run Code](https://www.programiz.com/python-programming/online-compiler)

Here,

* 'C' is not present in languages, 'C' in languages evaluates to False.
* 'Python' is present in languages, 'Python' in languages evaluates to True.

**Advantages of Tuple over List in Python**

Since tuples are quite similar to lists, both of them are used in similar situations.

However, there are certain advantages of implementing a tuple over a list:

* We generally use tuples for heterogeneous (different) data types and lists for homogeneous (similar) data types.
* Since tuples are immutable, iterating through a tuple is faster than with a list. So there is a slight performance boost.
* Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.
* If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

## Slicing tuples

Tuple slicing is a frequent practice in Python, and it is the most prevalent technique used by programmers to solve efficient problems. Consider a Python tuple. You must slice a tuple in order to access a range of elements in it. One method is to utilize the colon as a simple slicing operator (:).

The slice operator allows you to specify where to begin slicing, where to stop slicing, and what step to take. Tuple slicing creates a new tuple from an old one.

### Syntax

tuple[Start : Stop : Stride]

The above expression returns the portion of the tuple from index Start to index Stop, at a step size Stride.

### Example 1

In the following example we have used the slice operation to slice a tuple. We also use negative indexing method to slice a tuple.

tuple= ('a','b','c','d','e','f','g','h','i','j')

print(tuple[0:6])

print(tuple[1:9:2])

print(tuple[-1:-5:-2])

### Output

The above code produces the following results

('a', 'b', 'c', 'd', 'e', 'f')

('b', 'd', 'f', 'h')

('j', 'h')

### Example 2

Following is another example for this −

my\_tuple = ('t', 'u', 'r', 'i', 'a', 'l', 's', 'p','o', 'i', 'n', 't')

print(my\_tuple[1:]) #Print elements from index 1 to end

print(my\_tuple[:2]) #Print elements from start to index 2

print(my\_tuple[5:12]) #Print elements from index 1 to index 3

print(my\_tuple[::5]) #Print elements from start to end using step size

### Output

('u', 'r', 'i', 'a', 'l', 's', 'p', 'o', 'i', 'n', 't')

('t', 'u')

('l', 's', 'p', 'o', 'i', 'n', 't')

('t', 'l', 'n')

## Slice a Tuple

To slice a Tuple in Python, use slice() builtin function. We can just provide stop position for slicing a tuple; or provide both start and stop positions to slice() function.

slice() function returns indices. We use these indices along with the original tuple, to create the resulting sliced tuple.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Count() Method

The count() method of Tuple returns the number of times the given element appears in the tuple.

**Syntax:**

tuple.count(element)

Where the element is the element that is to be counted.

**Example 1: Using the Tuple count() method**

# Creating tuples

Tuple1 = (0, 1, 2, 3, 2, 3, 1, 3, 2)

Tuple2 = ('python', 'geek', 'python',

          'for', 'java', 'python')

# count the appearance of 3

res = Tuple1.count(3)

print('Count of 3 in Tuple1 is:', res)

# count the appearance of python

res = Tuple2.count('python')

print('Count of Python in Tuple2 is:', res)

**Output:**

Count of 3 in Tuple1 is: 3

Count of Python in Tuple2 is: 3

**Counting tuples and lists as elements in Tuples**

# Creating tuples

Tuple = (0, 1, (2, 3), (2, 3), 1,

         [3, 2], 'geeks', (0,))

# count the appearance of (2, 3)

res = Tuple.count((2, 3))

print('Count of (2, 3) in Tuple is:', res)

# count the appearance of [3, 2]

res = Tuple.count([3, 2])

print('Count of [3, 2] in Tuple is:', res)

**Output:**

Count of (2, 3) in Tuple is: 2

Count of [3, 2] in Tuple is: 1

**Index() Method**

The Index() method returns the first occurrence of the given element from the tuple.

**Syntax:**

tuple.index(element, start, end)

**Parameters:**

* **element:**The element to be searched.
* **start (Optional):**The starting index from where the searching is started
* **end (Optional):**The ending index till where the searching is done

**Note:**This method raises a ValueError if the element is not found in the tuple.

**Example 1: Using Tuple Index() Method**

|  |
| --- |
| # Creating tuples  Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)  # getting the index of 3  res = Tuple.index(3)  print('First occurrence of 3 is', res)  # getting the index of 3 after 4th  # index  res = Tuple.index(3, 4)  print('First occurrence of 3 after 4th index is:', res) |

**Output:**

First occurrence of 3 is 3

First occurrence of 3 after 4th index is: 5

**Example 2: Using Tuple() method when the element is not found**

|  |
| --- |
| # Creating tuples  Tuple = (0, 1, 2, 3, 2, 3, 1, 3, 2)  # getting the index of 3  res = Tuple.index(4) |

Output:

ValueError: tuple.index(x): x not in tuple