**Python Tuple with example**

In Python, a tuple is similar to [List](https://beginnersbook.com/2018/02/python-list/) except that the objects in tuple are immutable which means we cannot change the elements of a tuple once assigned. On the other hand, we can change the elements of a list.

**1. Tuple vs List**

1. The elements of a **list** are mutable whereas the elements of a **tuple** are immutable.  
2. When we do not want to change the data over time, the **tuple** is a preferred data type whereas when we need to change the data in future, **list** would be a wise option.  
3. Iterating over the elements of a **tuple** is faster compared to iterating over a **list**.  
4. Elements of a **tuple** are enclosed in parenthesis whereas the elements of **list** are enclosed in square bracket.

**2. How to create a tuple in Python**

To create a tuple in Python, place all the elements in a () parenthesis, separated by commas. A tuple can have heterogeneous data items, a tuple can have string and list as data items as well.

**2.1 Example – Creating tuple**

In this example, we are creating few tuples. We can have tuple of same type of data items as well as mixed type of data items. This example also shows nested tuple (tuples as data items in another tuple).

# tuple of strings

my\_data = ("hi", "hello", "bye")

print(my\_data)

# tuple of int, float, string

my\_data2 = (1, 2.8, "Hello World")

print(my\_data2)

# tuple of string and list

my\_data3 = ("Book", [1, 2, 3])

print(my\_data3)

# tuples inside another tuple

# nested tuple

my\_data4 = ((2, 3, 4), (1, 2, "hi"))

print(my\_data4)

Output:

('hi', 'hello', 'bye')

(1, 2.8, 'Hello World')

('Book', [1, 2, 3])

((2, 3, 4), (1, 2, 'hi'))

**2.2 Empty tuple:**

# empty tuple

my\_data = ()

**2.3 Tuple with only single element:**

Note: When a tuple has only one element, we must put a comma after the element, otherwise Python will not treat it as a tuple.

# a tuple with single data item

my\_data = (99,)

If we do not put comma after 99 in the above example then python will treat my\_data as an int variable rather than a tuple.

**3. How to access tuple elements**

We use indexes to access the elements of a tuple. Lets take few example to understand the working.

**3.1 Accessing tuple elements using positive indexes**

We can also have negative indexes in tuple, we have discussed that in the next section. Indexes starts with 0 that is why we use 0 to access the first element of tuple, 1 to access second element and so on.

# tuple of strings

my\_data = ("hi", "hello", "bye")

# displaying all elements

print(my\_data)

# accessing first element

# prints "hi"

print(my\_data[0])

# accessing third element

# prints "bye"

print(my\_data[2])

Output:

('hi', 'hello', 'bye')

hi

bye

**Note:**  
1. **TypeError**: If you do not use integer indexes in the tuple. For example my\_data[2.0] will raise this error. The index must always be an integer.  
2. **IndexError**: Index out of range. This error occurs when we mention the index which is not in the range. For example, if a tuple has 5 elements and we try to access the 7th element then this error would occurr.

**3.2 Negative indexes in tuples**

Similar to [list](https://beginnersbook.com/2018/02/python-list/) and [strings](https://beginnersbook.com/2018/02/python-strings/) we can use negative indexes to access the tuple elements from the end.  
-1 to access last element, -2 to access second last and so on.

my\_data = (1, 2, "Kevin", 8.9)

# accessing last element

# prints 8.9

print(my\_data[-1])

# prints 2

print(my\_data[-3])

Output:

8.9

2

**3.3 Accessing elements from nested tuples**

Lets understand how the double indexes are used to access the elements of nested tuple. The first index represents the element of main tuple and the second index represent the element of the nested tuple.

In the following example, when I used my\_data[2][1], it accessed the second element of the nested tuple. Because 2 represented the third element of main tuple which is a tuple and the 1 represented the second element of that tuple.

my\_data = (1, "Steve", (11, 22, 33))

# prints 'v'

print(my\_data[1][3])

# prints 22

print(my\_data[2][1])

Output:

v

22

**4. Operations that can be performed on tuple in Python**

Lets see the operations that can be performed on the tuples in Python.

**4.1 Changing the elements of a tuple**

We cannot change the elements of a tuple because elements of tuple are immutable. However we can change the elements of nested items that are mutable. For example, in the following code, we are changing the element of the list which is present inside the tuple. List items are mutable that’s why it is allowed.

my\_data = (1, [9, 8, 7], "World")

print(my\_data)

# changing the element of the list

# this is valid because list is mutable

my\_data[1][2] = 99

print(my\_data)

# changing the element of tuple

# This is not valid since tuple elements are immutable

# TypeError: 'tuple' object does not support item assignment

# my\_data[0] = 101

# print(my\_data)

Output:

(1, [9, 8, 7], 'World')

(1, [9, 8, 99], 'World')

**4.2 Delete operation on tuple**

We already discussed above that tuple elements are immutable which also means that we cannot delete the elements of a tuple. However deleting entire tuple is possible.

my\_data = (1, 2, 3, 4, 5, 6)

print(my\_data)

# not possible

# error

# del my\_data[2]

# deleting entire tuple is possible

del my\_data

# not possible

# error

# because my\_data is deleted

# print(my\_data)

Output:

(1, 2, 3, 4, 5, 6)

**4.3 Slicing operation in tuples**

my\_data = (11, 22, 33, 44, 55, 66, 77, 88, 99)

print(my\_data)

# elements from 3rd to 5th

# prints (33, 44, 55)

print(my\_data[2:5])

# elements from start to 4th

# prints (11, 22, 33, 44)

print(my\_data[:4])

# elements from 5th to end

# prints (55, 66, 77, 88, 99)

print(my\_data[4:])

# elements from 5th to second last

# prints (55, 66, 77, 88)

print(my\_data[4:-1])

# displaying entire tuple

print(my\_data[:])

Output:

(11, 22, 33, 44, 55, 66, 77, 88, 99)

(33, 44, 55)

(11, 22, 33, 44)

(55, 66, 77, 88, 99)

(55, 66, 77, 88)

(11, 22, 33, 44, 55, 66, 77, 88, 99)

**4.4 Membership Test in Tuples**

**in**: Checks whether an element exists in the specified tuple.  
**not in**: Checks whether an element does not exist in the specified tuple.

my\_data = (11, 22, 33, 44, 55, 66, 77, 88, 99)

print(my\_data)

# true

print(22 in my\_data)

# false

print(2 in my\_data)

# false

print(88 not in my\_data)

# true

print(101 not in my\_data)

Output:

(11, 22, 33, 44, 55, 66, 77, 88, 99)

True

False

False

True

**4.5 Iterating a tuple**

# tuple of fruits

my\_tuple = ("Apple", "Orange", "Grapes", "Banana")

# iterating over tuple elements

for fruit in my\_tuple:

print(fruit)

Output:

Apple

Orange

Grapes

Banana