

Python Tuple

- In Python programming, a tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.

Advantages of Tuple over List

- Since, tuples are quite similar to lists, both of them are used in similar situations as well.
- However, there are certain advantages of implementing a tuple over a list. Below listed are some of the main advantages:
- We generally use tuple for heterogeneous (different) data types and list for homogeneous (similar) data types.
 - Since tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
 - Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
 - If you have data that doesn't change, implementing it as tuple will guarantee that it remains write-protected.

Creating a Tuple

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it.
- A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).

empty tuple

my_tuple = ()

print(my_tuple) #Output: ()

tuple having integers

my_tuple = (1, 2, 3)

print(my_tuple) #Output: (1, 2, 3)

Tuple with mixed data types

my_tuple = (1, "Hello", 3.4)

print(my_tuple) #Output: (1, "Hello", 3.4)

Tuple can be created without parentheses also called tuple packing

my_tuple = 3, 4.6, "dog"

print(my_tuple) # Output: 3, 4.6, "dog"

```
# tuple unpacking is also possible
my_tuple = 3, 4.6, "dog"
a, b, c = my_tuple
print(a) # 3
print(b) # 4.6
print(c) # dog
```

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 in fact a tuple.

```
# only parentheses is not enough
my_tuple = ("hello")
print(type(my_tuple))    # Output: <class 'str'>
```

```
# need a comma at the end
my_tuple = ("hello",)
print(type(my_tuple))    # Output: <class 'tuple'>
```

```
# parentheses is optional
my_tuple = "hello",
print(type(my_tuple))    # Output: <class 'tuple'>
```

Accessing Elements in a Tuple

There are various ways in which we can access the elements of a tuple.

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 index from 0 to 5. Trying to access an element other than (0, 1, 2, 3, 4, 5) will raise an IndexError. The index must be an integer, so we cannot use float or other types. This will result into TypeError. Likewise, nested tuple are accessed using nested indexing, as shown in the example below.

```
my_tuple = ('p','e','r','m','i','t')
print(my_tuple[0])    # Output: 'p'
```

Changing a Tuple

- *Unlike lists, tuples are immutable.*
- *This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.*
- *We can also assign a tuple to different values (reassignment).*

```
my_tuple = (4, 2, 3, [6, 5])
```

```
# we cannot change an element
```

```
# If you uncomment line 8
```

```
# you will get an error:
```

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

```
#my_tuple[1] = 9
```

```
# but item of mutable element can be changed
```

```
my_tuple[3][0] = 9
```

```
print(my_tuple)                # Output: (4, 2, 3, [9, 5])
```

```
# tuples can be reassigned
```

```
my_tuple = ('p','r','o','g','r','a','m','i','z') # Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

```
print(my_tuple)
```

We can use + operator to combine two tuples. This is also called **concatenation**.

We can also **repeat** the elements in a tuple for a given number of times using the * operator.

Both + and * operations result into a new tuple.

```
# Concatenation
```

```
print((1, 2, 3) + (4, 5, 6))    # Output: (1, 2, 3, 4, 5, 6)
```

```
# Repeat
```

```
print(("Repeat",) * 3)          # Output: ('Repeat', 'Repeat', 'Repeat')
```

Deleting a Tuple

- As discussed above, we cannot change the elements in a tuple. That also means we cannot delete or remove items from a tuple.
- But deleting a tuple entirely is possible using the keyword del.

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

```
#del my_tuple[3]                # not possible
```

```
# can delete entire tuple
```

```
del my_tuple
```

```
my_tuple                        # NameError: name 'my_tuple' is not defined
```

Python Tuple Methods

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

Python Tuple Method	
Method	Description
<code>count(x)</code>	Return the number of items that is equal to <code>x</code>
<code>index(x)</code>	Return index of first item that is equal to <code>x</code>

Some examples of Python tuple methods:

```
my_tuple = ('a','p','p','l','e',)
```

Count

```
print(my_tuple.count('p')) # Output: 2
```

Index

```
print(my_tuple.index('l')) # Output: 3
```

Other Tuple Operations

1. Tuple Membership Test

We can test if an item exists in a tuple or not, using the keyword `in`.

```
my_tuple = ('a','p','p','l','e',)
```

```
print('a' in my_tuple) # Output: True
```

```
print('b' in my_tuple) # Output: False
```

```
print('g' not in my_tuple) # Output: True
```

2. Iterating Through a Tuple

Using a `for` loop we can iterate through each item in a tuple.

Output:

```
# Hello John
```

```
# Hello Kate
```

```
for name in ('John','Kate'):
```

```
    print("Hello",name)
```

Built-in Functions with Tuple

Built-in functions like `all()`, `any()`, `enumerate()`, `len()`, `max()`, `min()`, `sorted()`, `tuple()` etc. are commonly used with tuple to perform different tasks.

Built-in Functions with Tuple

<i>Function</i>	<i>Description</i>
<u><i>all()</i></u>	<i>Return True if all elements of the tuple are true (or if the tuple is empty).</i>
<u><i>any()</i></u>	<i>Return True if any element of the tuple is true. If the tuple is empty, return False.</i>
<u><i>enumerate()</i></u>	<i>Return an enumerate object. It contains the index and value of all the items of tuple as pairs.</i>
<u><i>len()</i></u>	<i>Return the length (the number of items) in the tuple.</i>
<u><i>max()</i></u>	<i>Return the largest item in the tuple.</i>
<u><i>min()</i></u>	<i>Return the smallest item in the tuple</i>
<u><i>sorted()</i></u>	<i>Take elements in the tuple and return a new sorted list (does not sort the tuple itself).</i>
<u><i>sum()</i></u>	<i>Return the sum of all elements in the tuple.</i>
<u><i>tuple()</i></u>	<i>Convert an iterable (list, string, set, dictionary) to a tuple.</i>