Python

Python Tuple

A tuple in Python 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 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, 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 »

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 »

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'>`

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 »

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 'm'
```

Run Code »

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 »

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 »

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 to iterate over the elements of a tuple. For example,

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

# iterating through the tuple
for language in languages:
    print(language)
```

Run Code »

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 »

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.

Difference between List and Tuple in Python

List	Tuple
1. List is mutable.	1. Tuple is immutable.
2. List iteration is slower and is time consuming.	2. Tuple iteration is faster.
3. List consumes more memory.	3. Tuples consumes less memory
4.List operations are more error prone.	4.Tuples operations are safe.
5.List provides many in-built methods.	5. Tuples have less in-built methods.
6. List is useful for insertion and deletion operations.	6.Tuple is useful for readonly operations like accessing elements.