Tuples

+4
Published Sep 1, 2021•**Updated Apr 21, 2023**

Contribute to Docs

A **tuple** is a data structure comprised of comma-separated values.
Unlike dictionaries, which are unordered and mutable, tuples are immutable and their elements are ordered by insertion (similar to lists). Tuple elements can be of different data types.

Tuples also support built-in sequence functions such as len(), min(), and max().

Syntax

There are multiple ways to create tuples in Python:

```
# With built-in function
tuple_instance(value1, value2, ..., valueN)

# With parentheses (multi-item)
tuple_instance = (value1, value2, ..., valueN)

# With parentheses (single item with a trailing comma)
tuple_instance = (singleValue, )

# With no parentheses (but with a trailing comma)
tuple_instance = singleValue,
```

The built-in tuple() function accepts an iterable value such as a list and returns a new tuple. Tuples can also be created with just parentheses (or no parentheses).

Note: If the tuple is made with just parentheses and has only one element, it must contain a trailing comma. Otherwise, Python may interpret the surrounding parentheses as an expression instead of a tuple:

```
# This will be treated as an int, not a tuple.
streaming_days = (3)

print(f"Value is {streaming_days}. Type is {type(streaming_days)}")
# Output: Value is 3. Type is <class 'int'>
```

Packing and Unpacking Tuples

When values are assigned to a tuple, it is “packed.” When those same values are utilized later on in a program, the tuple is “unpacked.”

```
# Packed tuple
my_tuple = (1, 2, 3)

# Unpacked tuple
(one, two, three) = my_tuple

print(one)
print(two)
print(three)

"""
Output:
1
2
3
"""
```

Note: The number of variables must be equal to the number of values in the tuple. Otherwise, an asterisk (*) must be used to gather the remaining values in a list:

```
# Packed tuple
my_tuple = (1, 2, 3, 4, 5)

# Unpacked tuple
(one, two, *three) = my_tuple

print(one)
print(two)
print(three)
"""
Output:
1
2
[3, 4, 5]
"""
```

Accessing and Updating Tuples

Like most sequence types in Python, tuple elements can be accessed by index:

```
my_tuple = ("Code Ninja", "Python")

print(my_tuple[1]) # Output: Python
```

Referencing a non-existent index will raise an `IndexError`.

Tuples can also be sliced with the following syntax:

```
tuple[start_index : stop_index : step]
```

The following is an example of slicing a tuple:

```
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
print(my_tuple[1 : 9 : 2])
# Output: (2, 4, 6, 8)
```

Like strings, tuples can also be concatenated to form a single tuple:

```
tuple_1 = (1, 2, 3)

tuple_2 = (4, 5, 6)

print(tuple_1 + tuple_2)
# Output: (1, 2, 3, 4, 5, 6)
```

Codebyte Example

Even though tuples themselves cannot be changed, and their elements can't be reassigned, after creation, there are some workarounds to this immutability. For example, if any of the tuple elements are themselves a mutable data type or structure, it can be modified or their elements reassigned:

```
list_1 = ["Netflix", "Hulu"]
list_2 = ["Amazon", "Apple TV"]

streaming_platforms = (list_1, list_2)

print(f"Original tuple: {streaming_platforms}\n")
```

```
streaming_platforms[0].append("YouTube")
# This is allowed

print(f"After appending to list: {streaming_platforms}\n")

streaming_platforms[0][-1] = "YT"
# This is also allowed

print(f"After reassigning list item: {streaming_platforms}\n")

streaming_platforms[1] = "Twitch"
# Reassignment of list to string will throw an error
```

Video Walkthrough

Watch this video which illustrates the characteristics of Python tuples.

Tuples

.count()

Returns the number of occurrences of a specific value in a tuple.

.index()

Returns the index of the first occurrence of a specific value in a tuple.