Tuples

October 7, 2024

1 Tuples

Tuples work a lot like lists, except they cannot be modified; they are immutable. Instead of brackets [] we use parentheses () around the elements.

[1]:
```
food = ('eggs', 'bananas', 'lemonheads')
food[1]
```

[1]:
```
'bananas'
```

[2]:
```
food[1:3]
```

[2]:
```
('bananas', 'lemonheads')
```

[3]:
```
food[1] = 'steak'
food
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 food[1] = 'steak'
      2 food

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

Lists and tuples turn out to be sequence data types – this means they support indexing and slicing. We will see below that strings are also a sequence type. We can also define a tuple by simply separating values by commas:

[4]:
```
food = 'trail mix', 'nothing'
food[0]
```

[4]:
```
'trail mix'
```

And of course, we can nest tuples and lists interchangably:

[ ]:
```
foo = ([1, 2, 3], [4, 5, (7, 8, 9)], (10, 11))
foo
```

Tuples, like lists, can also be empty. Use len() to find the length of a tuple:

[ ]:
```
foo = ()
len(foo)
```

We can also unpack tuples, which means assigning each element of a tuple to a variable. For example:

[ ]:
```
food = 'trail mix', 'nothing'
tylerfood, chrisfood = food
tylerfood = food[0]
chrisfood = food[1]
tylerfood
chrisfood
```

1.0.1 Summary

- Tuples are a like lists, but they are immutable.
- Syntax: like lists, but use parentheses () instead of brackets [].
- Find more documentation about sequence types here:

1.0.2 Negative Indexing

Allows for 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])
print(letters[-3])
```

1.0.3 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')
print(my_tuple[1:4])
```

[ ]:
```
print(my_tuple[7:])
```

[ ]:
```
print(my_tuple[:])
```

1.0.4 Tuple Methods

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

[ ]:
```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p'))  # prints 2
print(my_tuple.index('l'))  # prints 3
```

1.0.5 Iterating through a Tuple in Python

[ ]:
```
languages = ('Python', 'Swift', 'C++')

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

1.0.6 Check if an Item Exists in the Tuple

[ ]:
```
languages = ('Python', 'Swift', 'C++')

print('C' in languages)
print('Python' in languages)
```

1.0.7 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.

[ ]:
```

```