Python Tuple vs. List

Summary: in this tutorial, you'll learn the difference between tuple and list.

Both tuple (https://www.pythontutorial.net/python-basics/python-tuples/) and list (https://www.pythontutorial.net/python-basics/python-list/) are sequence types (https://www.pythontutorial.net/advanced-python/python-sequences/) . However, they have some main differences.

1) A tuple is immutable while a list is mutable

The following example defines a list and modifies the first element:

```
fruits = ['apple', 'orange', 'banana']
fruits[0] = 'strawberry'

print(fruits)
```

Output:

```
['strawberry', 'orange', 'banana']
```

As you can see clearly from the output, you can mutable a list. However, you cannot mutable a tuple. The following will result in an error:

```
fruits = ('apple', 'orange', 'banana')
fruits[0] = 'strawberry'
```

Error:

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

Python doesn't you to change the element of a tuple. But you can reference a new tuple. For example:

```
fruits = ('apple', 'orange', 'banana')
fruits = ('strawberry', 'orange', 'banana')
```

In this example, Python creates a new tuple and bounds the `fruits` variable to the newly created tuple.

If you examine the memory addresses of the tuple objects, you'll see that the `fruits` variable references a different memory address after the assignment:

```
fruits = ('apple', 'orange', 'banana')
print(hex(id(fruits)))

fruits = ('strawberry', 'orange', 'banana')
print(hex(id(fruits)))
```

Output:

```
0x1c018286e00
0x1c018286e40
```

2) The storage efficiency of a tuple is greater than a list

A list is mutable. It means that you can add more elements to it. Because of this, Python needs to allocate more memory than needed to the list. This is called over-allocating. The over-allocation improves performance when a list is expanded.

Meanwhile, a tuple is immutable therefore its element count is fixed. So Python just needs to allocate enough memory to store the initial elements.

As a result, the storage efficiency of a tuple is greater than a list.

To get the size of an object, you use the `getsizeof` function from the `sys` module.

The following shows the sizes of a list and a tuple that stores the same elements:

```
from sys import getsizeof

fruits = ['apple', 'orange', 'banana']
print(f'The size of the list is {getsizeof(fruits)} bytes.')

fruits = ('strawberry', 'orange', 'banana')
print(f'The size of the tuple is {getsizeof(fruits)} bytes.')
```

Output:

```
The size of the list is 80 bytes.
The size of the tuple is 64 bytes.
```

3) Copying a tuple is faster than a list

When you copy a list, Python creates a new list. The following example illustrates copying a list to another:

```
fruit_list = ['apple', 'orange', 'banana']
fruit_list2 = list(fruit_list)
print(id(fruit_list) == id(fruit_list2))  # False
```

However, when copying a tuple, Python just reuses an existing tuple. It doesn't create a new tuple because tuples are immutable.

```
fruit_tuple = ('apple', 'orange', 'banana')
fruit_tuple2 = tuple(fruit_tuple)
print(id(fruit_tuple) == id(fruit_tuple2))  # True
```

Therefore, copying a tuple always slightly faster than a list.

The following compares the time that needs to copy a list and a tuple 1 million times:

```
from timeit import timeit

times = 1_000_000

t1 = timeit("list(['apple', 'orange', 'banana'])", number=times)
print(f'Time to copy a list {times} times: {t1}')

t2 = timeit("tuple(('apple', 'orange', 'banana'))", number=times)
print(f'Time to copy a tuple {times} times: {t2}')

diff = "{:.0%}".format((t2 - t1)/t1)

print(f'difference: {diff}')
```

Summary

- A tuple is immutable while a list is mutable.
- The storage efficiency of a tuple is greater than a list.
- Copying a tuple is slightly faster than a list.
- Use a tuple if you don’t intend to mutable it.