Theory: Tuple

14 minutes 10 / 10 problems solved

Start practicing

By now, you definitely know how to handle a list, the most popular collection in Python. Now let's discover an equally useful data type — **tuples**. You should remember that they are almost identical to lists. What sets them apart is their **immutability**.

§1. Define a tuple

Since tuples cannot be changed, tuple creation is similar to opening a box of a fixed size, then putting several values into this box and sealing it. Once the box has been sealed, you cannot modify its size or content.

Use a pair of **parentheses** to define a tuple:

```
empty_tuple = ()
print(type(empty_tuple))  # <class 'tuple'>
```

Empty tuples are easy to create. Then what went wrong in the following example?

```
not_a_tuple = ('cat')
print(not_a_tuple)        # 'cat'
print(type(not_a_tuple))  # <class 'str'>
```

As you can see, the variable we created stores a string. It's actually a **comma** that makes a tuple, not parentheses. Let's fix this piece of code:

```
now_a_tuple = ('cat',)
print(now_a_tuple)        # ('cat',)
print(type(now_a_tuple))  # <class 'tuple'>
```

So, always use a comma when defining a singleton tuple. In fact, even if your tuple contains more than one element, separating items with commas will be enough:

```
weekend = 'Saturday', 'Sunday'
print(weekend)        # ('Saturday', 'Sunday')
print(type(weekend))  # <class 'tuple'>
```

The built-in function `tuple()` turns strings, lists and other **iterables** into a tuple. With this function, you can create an empty tuple as well.

```
# another empty tuple
empty_tuple = tuple()
print(empty_tuple)        # ()
print(type(empty_tuple))  # <class 'tuple'>

# a list turned into a tuple
bakers_dozen = tuple([12, 1])
print(bakers_dozen == (12, 1))  # True


# a tuple from a string

sound = tuple('meow')

print(sound)  # ('m', 'e', 'o', 'w')
```

§2. What can we do with tuples?

First, let's examine what characteristics lists and tuples have in common.

Current topic:

- ✓ Tuple

Topic depends on:

- ✓ Indexes (Stage 3) 7★

Topic is required for:

- ✓ Collections module
- ✓ Args
- Socket module
- ✓ Hashable
- POS tagging

Table of contents:

↑ Tuple

§1. Define a tuple

§2. What can we do with tuples?

§3. Immutability and its advantages

§4. Summary

Feedback & Comments

Both lists and tuples are **ordered**, that is, when passing elements to these containers, you can expect that their order will remain the same. Tuples are also indifferent to the nature of data stored in them, so you can **duplicate values** or **mix different data types**:

```
tiny_tuple = (0, 1, 0, 'panda', 'sloth')

print(len(tiny_tuple))  # 5
print(tiny_tuple)       # (0, 1, 0, 'panda', 'sloth')
```

Just like lists, tuples support **indexing**. Be careful with indexes though, if you want to get along without `IndexErrors`.

```
empty_tuple = ()
print(empty_tuple[0])  # IndexError

numbers = (0, 1, 2)
print(numbers[0])   # 0
print(numbers[1])   # 1
print(numbers[2])   # 2
print(numbers[3])   # IndexError
```

And here the first distinctive feature of tuples comes into play. What they don't support is **item assignment**. While you can change an element in a list referring to this element by its index, it's not the case for tuples:

```
# ex-capitals
capitals = ['Philadelphia', 'Rio de Janeiro', 'Saint Petersburg']

capitals[0] = 'Washington, D.C.'
capitals[1] = 'Brasília'
capitals[2] = 'Moscow'
print(capitals)  # ['Washington, D.C.', 'Brasília', 'Moscow']

former_capitals = tuple(capitals)

former_capitals[0] = 'Washington, D.C.'  # TypeError
```

In the example above, we tried to update the tuple and it didn't end well. You can't add an item to a tuple or remove it from there (unless you delete the entire tuple). However, immutability has a positive side. We'll discuss it in the next section.

§3. Immutability and its advantages

By this time, one question might have come to your mind: why use tuples when we have lists? Predictably, all answers conduce to immutability. Let's dwell on its upsides:

- Tuples are **faster** and **more memory-efficient** than lists. Whenever you need to work with large amounts of data, you should give it a thought. If you are not going to modify your data, perhaps you should decide on tuples.
- A tuple can be used as a **dictionary key**, whereas lists as keys will result in `TypeError`.
- Last but not least, it's **impossible to change** by accident the data stored in a tuple. It may prove a safe and robust solution to some tasks.

§4. Summary

Those were the very basics of tuples in Python. Just like lists, tuples are **ordered** and **iterable**. Unlike lists, they are **immutable**. You'll learn more of tuple features in the next topics, now it's time to write your first programs with them!

Report a typo

554 users liked this theory. **8** didn't like it. **What about you?**

Start practicing

Comments (26) Hints (0) Useful links (1) Show discussion