Theory: Operations with tuple

23 minutes 5 / 8 problems solved

You have already learned the tuple basics, now it's time to examine this datatype in more detail. As you know, tuples resemble lists in many aspects, but they are immutable and can't be changed once created. So, tuples are both similar to lists and different in some respects.

§1. Similarities

All operations on lists that do not alter the sequence itself can also be applied to tuples:

1. We can **add** tuples to each other and **multiply** them by a number:

```
first = (1, 2, 3, 4, 5)
the_rest = (6, 7, 8, 9)
print(first + the_rest)  # (1, 2, 3, 4, 5, 6, 7, 8, 9)
print(first * 2)         # (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

2. We can find the first **index** of an element in a tuple and **count** how many times an element occurs:

```

fruits = ("apple", "orange", "mango", "kiwi", "orange", "banana", "orange")
print(fruits.index("orange"))  # 1 (remember that we iterate from 0)
print(fruits.count("orange"))  # 3
```

We can also specify an interval for searching:

```
print(fruits.index("orange", 3, 5))  # 4
```

The first number is the index of the element to look from, the second one is the index of the element where to end the search. The last index is not included, so we've only checked the elements `"kiwi"` and `"orange"` with indexes 3 and 4 respectively.

> Be cautious, if the value is not present in a tuple, the `index()` method will raise `ValueError`.

3. Like lists, tuples can contain elements of any data type, which also implies that they can be **nested** in each other:

```
t = (1, 2, (3, 4, 5), 6)
print(len(t))             # 4
print(t[0] == 1)          # True
print(t[2] == (3, 4, 5))  # True
```

4. Tuples are iterable, so we can **iterate** and **enumerate** through them:

```
shopping_tuple = ("chicken", "rice", "curry sauce", "carrots", "milk")

for item in shopping_tuple:
    ...

for i, item in enumerate(shopping_tuple):
    ...
```

5. Finally, we can test if an element is **contained** in a tuple using operators `in` and `not in`:

Current topic:

- ✓ Operations with tuple

Topic depends on:

- ✓ Declaring a function Stage 1 10★
- ✓ For loop Stage 1 12★
- ✓ Hashable

Table of contents:

```
shopping_tuple = ("chicken", "rice", "curry sauce", "carrots", "milk")

print("soy sauce" in shopping_tuple)  # False

if "donuts" not in shopping_tuple:
    new_shopping_tuple = shopping_tuple + ("donuts",)
```

Everything discussed above applies to lists as well, but now it's time to learn more about the differences.

§2. Differences

It has been mentioned that tuples, as opposed to lists, can be used as dictionary keys. This is due to their immutability: only hashable objects can become dictionary keys.

```
# OK
spendings = {"food": 100, "apartment": 150, "gifts": 65}
spendings[("going out", "entertainment")] = 85

print(spendings)  # {"food": 100, "apartment": 150, "gifts": 60, ("going out", "entertainment"): 85}

# Error

spendings[["books", "magazines"]] = 70  # TypeError: unhashable type: 'list'
```

However, not every tuple can be a dictionary key but only the one all elements of which are also hashable.

```

spendings[('sports', ['sports drink', 'protein'])] = 55  # TypeError: unhashable type: 'list'
```

§3. Unpacking

As you know, tuples are immutable, so they are the optimal container to keep and pass data unchanged. Another feature of tuples, they can store data of any type. It's the reason why tuples are a usual format to pass mixed values, for example when we return several values from a function.

Consider a tuple with several facts about you: your name, your age, and what you did last summer:

```
# we use an imaginary person for an example
my_biography = ("Teddy", 22, "swimming and sunbathing")
```

It can be very useful to save each part in three different variables, so we can use the fact that tuples can be unpacked. Generally, the number of variables must match the length of a tuple:

```
name, age, what_i_did_last_summer = my_biography
print(name)                # Teddy


name, other = my_biography  # ValueError: too many values to unpack (expected 2)
```

Sometimes we need only a concrete variable, and keep the other in some `other` container. Use the unpacking operator `*` in this case, it will create a list for any variables of remaining elements:

```
name, *other = my_biography
print(name)   # Teddy
print(other)  # [22, "swimming and sunbathing"]

*other, what_i_did_last_summer = my_biography
print(other)                  # ["Teddy", 22]
print(what_i_did_last_summer)  # "swimming and sunbathing"
```

We can retrieve values from tuples two ways: getting elements by index or unpacking a tuple to variables.

§4. Swap values

An interesting thing characteristic of Python is connected to this fact. In programming, a common task is to swap the values of two variables. In most programming languages, you will need to use an additional temporary variable to store one of the values:

```
a = "letter A"
b = "letter B"
temp = a
a = b
b = temp
print(a, b)  # letter B letter A
```

However, Python allows us to do the swap in only one line:

```
a = "letter A"
b = "letter B"
a, b = b, a  # check it out!
print(a, b)  # letter B letter A
```

The thing is, Python evaluates assignments starting from the right-hand side. Firstly, a tuple of two elements, `b` and `a`, is created in the memory. Then, the already created tuple is assigned to the left-hand side. As the left-hand side is composed of two identifiers, the tuple is unpacked, so that the identifier `a` on the left is assigned to the first element of the tuple, and the identifier `b` — to the second element.

§5. Conclusion

In this topic, you have learned the features of tuples that are the same as those of lists: addition, multiplication, the methods `index()` and `count()`, the ability to be nested, iterability, and membership testing. You have also learned what sets tuples apart: immutability that allows them to be used as dictionary keys. What's more, you now know how to unpack tuples and swap values of two variables in only one line of code. Now, you are ready to use this knowledge in practice!

Report a typo

70 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (4) Hints (0) Useful links (0) Show discussion