Python → Collections → Operations with dictionary

Theory: Operations with dictionary

28 minutes 3 / 5 problems solved

Skip this topic Start practicing

1197 users solved this topic. Latest completion was 2 minutes ago.

You have learned about basic methods that are used to work with dictionaries. Let's talk about other operations. They will help you discover new features of dictionaries.

§1. Membership testing in a dictionary

Sometimes you need to check whether a specific item is present in your dictionary. For example, you have a furniture catalog where products (keys) are listed along with prices (values), and you want to find out if it has a blue sofa in it or not. In this case, you can use operators `in` and `not in` for this purpose. The syntax is quite simple: `key in dictionary` returns `True` if `key` exists in `dictionary` and `False` otherwise. The `not in` operator does the opposite, it returns `True` if `key` does not exist in the dictionary:

```
catalog = {'green table': 5000, 'brown chair': 1500, 'blue sofa': 15000, 'wardrobe': 10000}

print('blue sofa' in catalog)        # True
print('green table' not in catalog)  # False

```

Note that the membership operator looks for keys, not values:

```
print(1500 in catalog)  # False
```

§2. Iterating over keys

You already know that the `for` loop allows us to iterate over elements of an object. So what does iteration over a dictionary give us? Let's take a look at the following example:

```
tiny_dict = {'a': 1, 'b': 2, 'c': 3}

for obj in tiny_dict:
    print(obj)
```

We see the *keys* of the dictionary in the output:

```
a
b
c
```

A similar way to iterate over keys is to use the `keys` method, which creates a special iterable object — a collection of dictionary keys:

```
print(tiny_dict.keys())  # dict_keys(['a', 'b', 'c'])
```

Now let's try to write our loop using the `keys` method and check whether the output remains the same:

```
for obj in tiny_dict.keys():
    print(obj)
# a
# b
# c
```

§3. Including values in iteration

What if we want to get more than just the dictionary keys when iterating?

Current topic:

Operations with dictionary Stage 1

Topic depends on:

✓ Dictionary methods Stage 1

Topic is required for:

✓ Kwargs

Processing requests Stage 1

Table of contents:

The `values` method is quite similar to the previous one, the only difference is that you get the values, not the keys. It provides a collection of values, without any information about keys that are used to get these values from the dictionary:

```
for value in tiny_dict.values():
    print(value)
# 1
# 2
# 3

print(tiny_dict.values())  # dict_values([1, 2, 3])
```

Finally, the `items` method provides complete iteration in case you need both keys and values. It returns the collection of `(key, value)` pairs (tuples):

```
for obj in tiny_dict.items():
    print(obj)
# ('a', 1)
# ('b', 2)
# ('c', 3)

print(tiny_dict.items())  # dict_items([('a', 1), ('b', 2), ('c', 3)])
```

§4. Dictionary comprehension

Dictionary comprehension is a very convenient and concise way to create a new dictionary with one line of code. The minimal template looks like this:

```
dictionary = {key: value for element in iterable}
```

Let's take a closer look. The expression is grouped in curly brackets — `{}`. What happens inside? The `for` loop goes over the elements of an iterable object (list, another dictionary, etc.). To create a dictionary, we need to specify the `key`, which must be bound with an iterable object, and then the `value`, which can be arbitrary:

```
dictionary = {key + 5: 'some_value' for key in range(3)}
print(dictionary)  # {5: 'some_value', 6: 'some_value', 7: 'some_value'}
```

However, the `value` is usually also associated with the iterable:

```
dictionary = {n + 10: n + 100 for n in range(5)}
print(dictionary)  # {10: 100, 11: 101, 12: 102, 13: 103, 14: 104}
```

In the example above, we retrieve keys and values by performing operations on elements in the iterable object.

However, dictionary comprehension is used more often to create a new dictionary by changing values in another dictionary. Imagine that we have a dictionary that contains the names of the planets and their diameters in kilometers. You need to create a new dictionary where the diameters are in miles. Without the dictionary comprehension, it would be like this:

```
planets_diameter_km = {'Earth': 12742, 'Mars': 6779}

# correct but long way
planets_diameter_mile = {}
for key, value in planets_diameter_km.items():
    planets_diameter_mile[key] = round(value / 1.60934, 2)

print(planets_diameter_mile)  # {'Mars': 4212.29, 'Earth': 7917.53}
```

Now let's wrap the same operation with the dictionary comprehension; we will convert the values from kilometers into miles:

```
# convenient and short!
planets_diameter_mile = {key: round(value / 1.60934, 2) for (key, value) in
                         planets_diameter_km.items()}
print(planets_diameter_mile)  # {'Mars': 4212.29, 'Earth': 7917.53}
```

We can devise some conditions in our expression. For now, we want to include only the planets that are bigger than 10000 km in the new dictionary:

```
planets_diameter_mile = {key: round(value / 1.60934, 2) for (key, value) in
                         planets_diameter_km.items() if value > 10000}
print(planets_diameter_mile)  # {'Earth': 7917.53}
```

So, the dictionary comprehension streamlines the process of creating a dictionary, and the logic of the process is understandable. However, be careful not to make your code hard to read.

You can find more information about dictionary comprehension on the official Python website.

§5. Recap

In this topic, you've learned some tricks about dictionaries:

- `in` and `not in` operators allow to test for membership in a dictionary, though, they look for keys only;
- the `for` loop can iterate through the keys of a dictionary;
- `keys` and `values` methods give you access to the keys and values of a dictionary and the `items` method — to both at the same time.
- the dictionary comprehension is a quick and easy way to create a dictionary.

Report a typo

96 users liked this theory. 0 didn't like it. **What about you?**

Start practicing

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