Python → Builtins → Map and filter

Theory: Map and filter

20 minutes 0 / 5 problems solved

Skip this topic Start practicing

1607 users solved this topic. Latest completion was about 14 hours ago.

Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions. Even though Python isn't a purely functional language, it implements some functional programming functionality, for example, lambda functions that you are already familiar with. In this topic, you will learn about other useful functional programming tools: `map()` and `filter()`.

§1. map( )

Suppose you have a list containing some numbers:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Imagine you now want to obtain a new list that contains the numbers from the list above multiplied by 2.

You can of course always pass the numbers one by one and collect the output in a for loop:

```
doubled_numbers = [2*n for n in numbers]

print(doubled_numbers)
# The output is [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

However, there exists another way to do so with the help of the built-in `map()` function. Let's take a look at its syntax:

```
map(function, iterable)
```

`map()` takes a function object and a list (or any other iterable, e.g., an array or a dictionary). The function is then applied to each element of that list, and an **iterator** is returned. You can explicitly convert it to a list using the `list()` function to see the result.

So the code from the example above can be re-written as follows:

```
def doubler(x):
    return 2*x

doubled_numbers = map(doubler, numbers)

print(list(doubled_numbers))
# Outputs [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Note that we can always combine `map()` with lambda functions:

```
doubled_numbers = map(lambda x: 2*x, numbers)

print(list(doubled_numbers))
# Outputs [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

`map()` comes in handy when a function you want to apply several times takes multiple arguments. In that case, you can simply pass more than one list to `map()`. The values for each argument will be then taken from the corresponding list.

The code below, for example, computes the sum of three lists:

Current topic:

Map and filter ···

Topic depends on:

✓ Lambda functions ···

✓ List comprehension 5★ ···

Table of contents:

```
x_list = [1, 2, 3]
y_list = [4, 5, 6]
z_list = [7, 8, 9]

s = list(map(lambda x, y, z: x + y + z, x_list, y_list, z_list))

print(s)
# The output is [12, 15, 18]
```

§2. filter( )

Let's continue working with our list of numbers from 0 to 9:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Imagine now that you want to obtain yet another list which contains only the odd numbers from the list above.

To that end, the built-in `filter()` function comes in handy.

`filter()` takes a Boolean function, i.e., a function that returns True or False, along with a list, and constructs a new iterator from the elements of the list for which the function returns True. This iterator can be then converted to a list. The syntax is as follows:

```
filter(boolean_function, iterable)
```

Thus, we can collect all the odd numbers from our list in the following way:

```
odd_numbers = filter(lambda x: x % 2, numbers)

print(list(odd_numbers))
# The output is [1, 3, 5, 7, 9]
```

As you might have already noticed, you can do the same thing in a for loop:

```
odd_numbers = [n for n in numbers if n % 2]

print(odd_numbers)
# Also outputs [1, 3, 5, 7, 9]
```

§3. map( ) and filter( ) or list comprehension: what should I choose?

As you have probably noticed while reading this material, `map()` and `filter()` can always be replaced by list comprehensions. So, what should be preferred?

Even though Python implements the functional programming functionality such as `map()` and `filter()`, it isn't a functional programming language. Therefore, many developers argue that more *'pythonic'* ways should be preferred where possible. Also, some programmers find list comprehensions more readable and easy to understand.

At the same time, under some conditions, `map()` and `filter()` can be faster than list comprehensions, so it's definitely worth keeping them in mind.

§4. Conclusions

- `map()` and `filter()` come from the functional programming paradigm.
- `map()` evaluates some function on multiple parameter values in a list.
- `filter()` finds a subset of elements of a list that satisfy some condition.
- `map()` and `filter()` can be replaced with list comprehensions. However, sometimes they can be more computationally efficient.

Report a typo

132 users liked this theory. 4 didn't like it. What about you?

Start practicing

Comments (9) Hints (0) Useful links (0) Show discussion