Theory: Lambda functions

15 minutes 7 / 8 problems solved

Start practicing

§1. Defining a lambda function

Imagine that you want to write a function that takes a number and doubles it. If you already know how to define functions in Python using the **def** keyword, you will probably write something like that:

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

Well, there is actually another way to define such small functions in Python using the **lambda** keyword. The following function is completely equivalent to the one defined above:

```
lambda x: 2 * x
```

This function doesn't have a name and is, therefore, called **anonymous**. Since in Python anonymous functions are declared with the **lambda** keyword, they are often referred to as **lambda functions**.

Let's take a look at the syntax in its general form:

```
lambda arguments: expression
```

A lambda function can take any number of arguments separated by commas, but it must consist of a ***single*** expression. This expression is evaluated and the result is returned. Note that you do not need the **return** statement here. For example, the following anonymous function computes the remainder of the division of the sum of two numbers by two:

```
lambda x, y: (x + y) % 2
```

In case you want to put a condition in some lambda function, you'll have to use the so-called ternary operator `first_alternative if condition else second_alternative`:

```
# Yes
lambda x: 'even' if x % 2 == 0 else 'odd'

# No
lambda x:
    if x % 2 == 0:
        return 'even'
    else:
        return 'odd'
```

Classic conditional statements will not work within a lambda function.

§2. Invoking a lambda function

Alright, but how do we call such a function if it does not have a name?

Python syntax allows us to do so by enclosing the function in brackets and passing arguments right away:

```
(lambda x, y: (x + y) % 2)(1, 5)
# The output is 0
```

Alternatively, it is also possible to assign a function object to a variable:

```
func = lambda x, y: (x + y) % 2
func(1, 10)
# The output is 1
```

Current topic:

- ✓ Lambda functions

Topic depends on:

- ✓ Declaring a function (Stage 1) 10★
- ✓ Else statement (Stage 1) 15★

Topic is required for:

- Map and filter
- ✓ Sorting a list

Table of contents:

However, assigning an anonymous function does not comply with the official style guidelines. It's reasonable to declare your function explicitly with the **def** keyword in case you want it to have a name.

§3. When is it useful?

You might have noticed already that the function from our example above is fully equivalent to a 'normal' function defined as follows:

```
def my_func(x, y):
    return (x + y) % 2
```

But if we can always use a normal function instead, why are lambda functions useful?

Well, lambda functions are handy, for example, when you use them in combination with another function. Take a look at the following example:

```
def create_function(n):
    return lambda x: n * x
```

The function `create_function` takes one argument, number ***n***, and returns a function that multiplies any given number ***x*** by that ***n***. You can use it further in your program to quickly define a bunch of functions, for example:

```
# Creating a function that doubles its argument
doubler = create_function(2)

# This function will triple its argument
tripler = create_function(3)

doubler(2)
# Outputs 4


tripler(2)

# Outputs 6
```

As you can see, the functions `doubler()` and `tripler()` are designed rather uniformly: they take a single argument and return it multiplied by *2* and *3* respectively. Thus, lambda functions can be embedded into a larger function, like `create_function()` in our example.

§4. Conclusions

Let's go over the main points we discussed:

- **Anonymous functions** are functions defined without a name.
- You can use the **lambda** keyword to define anonymous functions in Python.
- A lambda function can only contain a single expression.
- Lambda functions are particularly handy for one-time use, or when combined with other functions.

Report a typo

214 users liked this theory. **10** didn't like it. **What about you?**

Start practicing

Comments (12) Hints (3) Useful links (1) Show discussion