Pierian-Data / Complete-Python-3-Bootcamp Public

Code Issues 95 Pull requests 74 Actions Projects Security In

master

Complete-Python-3-Bootcamp / 03-Methods and Functions / .ipynb_checkpoints / 05-Lambda-Expressions-Map-and-Filter-checkpoint.ipynb

Pierian-Data added image link to all notebooks

1 contributor

580 lines (580 sloc) 12 KB

Content Copyright by Pierian Data

Lambda Expressions, Map, and Filter

Now its time to quickly learn about two built in functions, filter and map. Once we learn about how these operate, we can learn about the lambda expression, which will come in handy when you begin to develop your skills further!

map function

The **map** function allows you to "map" a function to an iterable object. That is to say you can quickly call the same function to every item in an iterable, such as a list. For example:

In [1]:
```
def square(num):
    return num**2
```

In [2]:
```
my_nums = [1,2,3,4,5]
```

In [5]:
```
map(square,my_nums)
```

Out[5]:
```
<map at 0x205baec21d0>
```

In [7]:
```
# To get the results, either iterate through
# or just cast to a list
list(map(square,my_nums))
```

Out[7]:
```
[1, 4, 9, 16, 25]
```

The functions can also be more complex

In [8]:
```
def splicer(mystring):
    if len(mystring) % 2 == 0:
        return 'even'
    else:
        return mystring[0]
```

In [9]:
```
mynames = ['John','Cindy','Sarah','Kelly','M
```

In [10]:
```
list(map(splicer,mynames))
```

Out[10]:
```
['even', 'C', 'S', 'K', 'even']
```

filter function

The filter function returns an iterator yielding those items of iterable for which function(item) is true. Meaning you need to filter by a function that returns either True or False. Then passing that into filter (along with your iterable) and you will get back only the results that would return True when passed to the function.

In [12]:
```
def check_even(num):
    return num % 2 == 0
```

In [13]:
```
nums = [0,1,2,3,4,5,6,7,8,9,10]
```

In [15]:
```
filter(check_even,nums)
```

Out[15]:
```
<filter at 0x205baed4710>
```

In [16]:
```
list(filter(check_even,nums))
```

Out[16]:
```
[0, 2, 4, 6, 8, 10]
```

lambda expression

One of Pythons most useful (and for beginners, confusing) tools is the lambda expression. lambda expressions allow us to create "anonymous" functions. This basically means we can quickly make ad-hoc functions without needing to properly define a function using def.

Function objects returned by running lambda expressions work exactly the same as those created and assigned by defs. There is key difference that makes lambda useful in specialized roles:

lambda's body is a single expression, not a block of statements.

- The lambda's body is similar to what we would put in a def body's return statement. We simply type the result as an expression instead of explicitly returning it. Because it is limited to an expression, a lambda is less general that a def. We can only squeeze design, to limit program nesting. lambda is designed for coding simple functions, and def handles the larger tasks.

Lets slowly break down a lambda expression by deconstructing a function:

In [17]:
```
def square(num):
    result = num**2
    return result
```

In [18]:
```
square(2)
```

Out[18]:
```
4
```

We could simplify it:

In [19]:
```
def square(num):
    return num**2
```

In [20]:
```
square(2)
```

Out[20]:
```
4
```

We could actually even write this all on one line.

In [21]:
```
def square(num): return num**2
```

In [22]:
```
square(2)
```

Out[22]:
```
4
```

This is the form a function that a lambda expression intends to replicate. A lambda expression can then be written as:

In [23]:
```
lambda num: num ** 2
```

Out[23]:
```
<function __main__.<lambda>>
```

In [25]:
```
# You wouldn't usually assign a name to a la
square = lambda num: num **2
```

In [26]:
```
square(2)
```

Out[26]:
```
4
```

So why would use this? Many function calls need a function passed in, such as map and filter. Often you only need to use the function you are passing in once, so instead of formally defining it, you just use the lambda expression. Let's