master Complete-Python-3-Bootcamp / 03-Methods and Functions / .ipynb_checkpoints / 03-Lambda expressions-checkpoint.ipynb

Go to file ...

Pierian-Data PYTHON 3 UPDATES Latest commit 6bdd11b on Feb 13, 2018 History

1 contributor

394 lines (394 sloc) | 7.73 KB

Raw Blame

lambda expressions

One of Python's 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 `def`s. There is a key difference that makes lambda useful in specialized roles:

A 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 than a `def`. We can only squeeze design, to limit program nesting. lambda is designed for coding simple functions, and `def` handles the larger tasks.

Let's slowly break down a lambda expression by deconstructing a function:

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

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

Out[2]: 4

Continuing the breakdown:

In [3]:
```
def square(num):
    return num**2
```

In [4]:

```
square(2)
```

Out[4]: 4

We can actually write this in one line (although it would be bad style to do so)

In [5]:
```
def square(num): return num**2
```

In [6]:
```
square(2)
```

Out[6]: 4

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

In [7]:
```
lambda num: num**2
```

Out[7]: <function __main__.<lambda>>

Note how we get a function back. We can assign this function to a label:

In [8]:
```
square = lambda num: num**2
```

In [9]:
```
square(2)
```

Out[9]: 4

And there you have it! The breakdown of a function into a lambda expression! Lets see a few more examples:

Example 1

Check that a number is even:

In [10]:
```
even = lambda x: x%2 == 0
```

```
even = lambda x: x%2==0
```

In [11]:
```
even(3)
```

Out[11]:
```
False
```

In [12]:
```
even(4)
```

Out[12]:
```
True
```

Example 2

Grab first character of a string:

In [13]:
```
first = lambda s: s[0]
```

In [14]:
```
first('hello')
```

Out[14]:
```
'h'
```

Example 3

Reverse a string:

In [15]:
```
rev = lambda s: s[::-1]
```

In [16]:
```
rev('hello')
```

Out[16]:
```
'olleh'
```

Example 4

Example 4

Just like a normal function, we can accept more than one argument into a lambda expression:

In [17]:
```
adder = lambda x,y : x+y
```

In [18]:
```
adder(2,3)
```

Out[18]: 5

lambda expressions really shine when used in conjunction with **map()**, **filter()** and **reduce()**. Each of those functions has its own lecture, so feel free to explore them if you're very interested in lambda.