6.2-Functions Advanced

KeytoDataScience.com

Table of Contents

1 Anonymous (`lambda`) Functions

Earlier we quickly covered the most common way of defining functions, the `def` statement.

You'll likely come across another way of defining short, one-off functions with the `lambda` statement.

It looks something like this:

In [1]:
```
add = lambda x, y: x + y
add(1, 2)
```

Out[1]:
```
3
```

This lambda function is roughly equivalent to

In [2]:
```
def add(x, y):
    return x + y
```

Lambdas differ from normal Python methods because they can have only one expression, can't contain any statements and their return type is a function object. So the line of code above doesn't exactly return the value x + y but the function that calculates x + y.

Lambda functions are frequently used with higher-order functions, which take one or more functions as arguments or return one or more functions.

A `lambda function` can be a higher-order function by taking a function (normal or lambda) as an argument like in the following example:

In [3]:

```
high_ord_func = lambda x, func: x + func(x)
```

In [4]:
```
high_ord_func(2, lambda x: x * x)
```

Out[4]:
```
6
```

Explanation: We have entered number 2 in `high_ord_func` which perform x+func(x).

So, if we put 2 as input, then output will be `2 + (2*2)` = **6**

In [5]:
```
high_ord_func(2, lambda x: x + 3)
```

Out[5]:
```
7
```

Explanation: We have entered number 2 in `high_ord_func` which perform x+func(x).

So, if we put 2 as input, then output will be `2 + (2+3)` = **7**

↥ back to top

2 The `map()` Function

The map() function iterates through all items in the given iterable and executes the function we passed as an argument on each of them.

The syntax is:

map(function, iterable(s)) We can pass as many iterable objects as we want after passing the function we want to use:

In [6]:
```
def starts_with_A(s):
    return s[0] == "A"

fruit = ["Apple", "Banana", "Pear", "Apricot", "Orange"]
map_object = map(starts_with_A, fruit)

print(list(map_object))
```

```
[True, False, False, True, False]
```

↥ back to top

3 Revise

3.1 Basic Functions

In [7]:
```
nums = [3,2,6,8,4,6,2,9]
nums
```

Out[7]:
```
[3, 2, 6, 8, 4, 6, 2, 9]
```

Print even numbers

In [8]:
```
evens = list(filter(lambda n : n%2==0,nums))
evens
```

Out[8]:
```
[2, 6, 8, 4, 6, 2]
```

Multiply by 2 - Only even numbers from above list

In [9]:
```
doubles = list(map(lambda n : n*2,evens))
print(doubles)
```

```
[4, 12, 16, 8, 12, 4]
```

3.2 Applying a function to a pandas Series or DataFrame

In [10]:
```
import pandas as pd
```

In [11]:
```
url = 'http://bit.ly/kaggletrain'
train = pd.read_csv(url)
#or
#train = pd.read_csv('titanic_train')
train.head(3)
```

Out[11]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	

map() function as a Series method

Mostly used for mapping categorical data to numerical data

In [12]:
```
# create new column
train['Sex_num'] = train.Sex.map({'female':0, 'male':1})
# let's compared Sex and Sex_num columns
# here we can see we map male to 1 and female to 0
train.loc[0:4, ['Sex', 'Sex_num']]
```

Out[12]:

	Sex	Sex_num

	Sex	Sex_num
0	male	1
1	female	0
2	female	0
3	female	0
4	male	1

apply() function as a Series method

Applies a function to each element in the Series

Calculate length of string in each string in "Name" column

In [13]:

```
# create new column
# we are applying Python's len function
train['Name_length'] = train.Name.apply(len)
# the apply() method applies the function to each element
train.loc[0:4, ['Name', 'Name_length']]
```

Out[13]:

	Name	Name_length
0	Braund, Mr. Owen Harris	23
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	51
2	Heikkinen, Miss. Laina	22
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	44
4	Allen, Mr. William Henry	24

↥ back to top

Great Job!