lambda

September 13, 2023

1 Lambda Function with (filter, map, apply, applymap, reduce)

A lambda function is like a secret helper that can do a simple task for you. It doesn't have a name, and it can do only one thing. You can tell it what to do, and it will give you the answer right away, but it can only do one thing at a time.

The anatomy of a lambda function includes three elements:

1. The keyword lambda — an analog of def in normal functions
2. The parameters — support passing positional and keyword arguments, just like normal functions
3. The body — the expression for given parameters being evaluated with the lambda function

We use a lambda function to evaluate only one short expression (ideally, a single-line) and only once, meaning that we aren't going to apply this function later. Usually, we pass a lambda function as an argument to a higher-order function (the one that takes in other functions as arguments), such as Python built-in functions like filter(), map(), or reduce().

[4]:
```
lambda x:x**2

# It is same like square function

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

[5]:
```
square(2)
```

[5]:
```
4
```

[10]:
```
x=2
lambda x:x**2

# this function only create a function object does't return any value , becuase␣
 ↪it is not a correct way to pass the arguments
```

[10]:
```
<function __main__.<lambda>(x)>
```

[8]:
```
(lambda x:x**2)(2)
```

[8]: 4

1. When we make a lambda function, we don't use () around what it works on, but when we use it, we put () around both the lambda function and the thing we want it to work on.
2. With a lambda function, we can make it work right away and get the answer. This is called "immediately doing the job."
3. If a lambda function has more than one thing to work on, we list them with commas when we make it and also when we ask it to work. Like this:

lambda_function = lambda parameter1, parameter2: expression result = lambda_function(argument1, argument2)

[11]:
```
(lambda x: 'even' if (x%2==0) else 'odd')(10)
```

[11]: 'even'

2 Applications of a Lambda Function in Python

Lambda with the filter(), map(), apply() Function

3 filter function used with lambda function

We use the filter() function in Python to pick out specific things from a group (like lists, sets, or other groups) by telling it how to decide what to keep. It needs two things:

1. A rule for picking things (like big or small)
2. A group of things to pick from

Imagine you have a bunch of things, like a list of numbers or a table of data (like Excel). You want to pick out only the things that follow a rule you set. A lambda function is like a mini-rule that helps you do this. You use it with the filter function to find and keep only the things that match your rule.

[21]:
```
list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
result = sorted(filter(lambda x: x % 2 != 0, list1))
result
```

[21]: [1, 3, 5, 7, 9]

[22]:
```
filter(lambda x: x % 2 != 0, list1)
```

[22]: <filter at 0x1fc7c22b3a0>

[59]:
```
import pandas as pd

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],
        'Age': [25, 30, 22, 35]}
```

```
df = pd.DataFrame(data)
df
```

[59]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	22
3	David	35

```
[60]: df[df['Age'].apply(lambda x: x > 25)]
```

[60]:

	Name	Age
1	Bob	30
3	David	35

4 lambda function used with map function

When you have a bunch of things (like numbers in a list), and you want to do something to each of them (like double them), you can use the map() function. It helps you do that same thing to every item one by one.

```
[61]: lst = [1, 2, 3, 4, 5]
print(map(lambda x: x * 10, lst))
tpl = tuple(map(lambda x: x * 10, lst))
tpl
```

```
<map object at 0x000001FC00E27910>
```

[61]: (10, 20, 30, 40, 50)

Difference Between map() and filter() Functions :

The main difference between map() and filter() is that map() always gives you an output with the same number of items as the input. So, if you have a bunch of things (like numbers or data in a table) and you want to do something to each of them and keep all of them, you use map().

Using map() with a DataFrame (Table of Data):

For example, if you have ages in one column and you want to add 5 to all of them and create a new column with the new ages, you can use map() with a DataFrame.

```
[62]: df
```

[62]:

	Name	Age
0	Alice	25
1	Bob	30
2	Charlie	22
3	David	35

[63]:
```
df['col3'] = df['Age'].map(lambda x: x + 5)
df
```

[63]:

	Name	Age	col3
0	Alice	25	30
1	Bob	30	35
2	Charlie	22	27
3	David	35	40

5 lambda function used with Apply function

Imagine you have a table of data (like a spreadsheet) or a list of things (like numbers). You want to do something specific to each item in that table or list. The apply() function lets you use a small, custom rule (lambda function) to do that something for each item, one at a time

[64]:
```
df['Age Squared'] = df['Age'].apply(lambda x: x ** 2)
df
```

[64]:

	Name	Age	col3	Age Squared
0	Alice	25	30	625
1	Bob	30	35	900
2	Charlie	22	27	484
3	David	35	40	1225

6 applymap (for DataFrames only):

Use applymap when you want to apply a function to each individual cell in a DataFrame. It operates element-wise and doesn't provide access to entire rows or columns. It's useful for simple element-wise operations. Often used for straightforward data type conversions or transformations that don't require looking at other elements in the DataFrame. It returns a DataFrame with the same shape as the original

Suppose i want to change the datatype of all columns then i am not able to do this with map i , havt to use there apply because map is only work with series

[65]:
```
df[['Name', 'Age']] = df[['Name', 'Age']].applymap(lambda x: str(x))
```

[66]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Name         4 non-null      object
 1   Age          4 non-null      object
 2   col3         4 non-null      int64 
```

```
 3   Age Squared  4 non-null      int64
dtypes: int64(2), object(2)
memory usage: 256.0+ bytes
```

[ ]:
```
df['sum'] = df.apply(lambda row : row['col3']+row['Age Squared'],axis=1)
df
```

	Name	Age	col3	Age Squared	sum
0	Alice	25	30	625	655
1	Bob	30	35	900	935
2	Charlie	22	27	484	511
3	David	35	40	1225	1265

7 Lambda with the reduce() Function

The reduce() function is related to the functools Python module, and it works in the following way:

1. Operates on the first two items of an iterable and saves the result
2. Operates on the saved result and the next item of the iterable
3. Proceeds in this way over the pairs of values until all the items of the iterable are used

[68]:
```
from functools import reduce
lst = [1, 2, 3, 4, 5]
reduce(lambda x, y: x + y, lst)
```

[68]: 15

[69]:
```
numbers = [1, 2, 3, 4, 5]

# Using reduce with a lambda function to find the product of elements
product = reduce(lambda x, y: x * y, numbers)

print("Product of numbers:", product)
```

```
Product of numbers: 120
```

[70]:
```
numbers = [12, 45, 23, 67, 8, 90, 34]

# Using reduce with a lambda function to find the maximum element
max_number = reduce(lambda x, y: x if x > y else y, numbers)

print("Maximum number:", max_number)
```

```
Maximum number: 90
```

[71]:
```
df
```

[71]:

	Name	Age	col3	Age Squared
0	Alice	25	30	625
1	Bob	30	35	900
2	Charlie	22	27	484
3	David	35	40	1225

[73]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   Name         4 non-null      object
 1   Age          4 non-null      object
 2   col3         4 non-null      int64 
 3   Age Squared  4 non-null      int64 
dtypes: int64(2), object(2)
memory usage: 256.0+ bytes
```