Lambda Function - A better understanding

BEGINNER PROGRAMMING PYTHON

This article was published as a part of the Data Science Blogathon

Greetings! If they are working on a real-time application or a complicated activity, all programmers, regardless of the domain, will come across the phrase function block in coding. So, what this function block will do is make our work easier and more elegant by making our code look simple even though the action we used is complicated.

Introduction to Function

```
def sub(num1, num2):
    print("type number 1:", num1)
    print("type number 2:", num2)
    subtraction = num1 - num2

    return subtraction
```

```
result = sub(10, 5)
print(result)
```

```
type number 1: 10
type number 2: 5
5
```

Image Source: Author

A function is a code block that includes a few structured statements of code, works when called, and can be called n times inside a program. The block performs a specific operation and can be utilized at any time. The FUNCTIONS function is described in full in the diagram above.

As you can see from the function statement, there are two critical parameters,

1. sub – **Function name** (which describes what is the function for in simple!)

2. num1, num2 – **Parameters** (here we can give the value while we call the function)

When writing a function, it's apparent that the "def" word is critical; it produces the function and the function body, which explains the action or what this specific function will do? Finally, the "return" statement is used to print the value. As a consequence, the method described above may be used for any Python function.

Types of function

Based on the previous description, I feel we have a good understanding of how the function works, and we will see how functions are classified here.

1. **User-Defined Function (USF)** – Customized way to write a function

2. **Built-in Function (BiF):** Cannot customize, we need to use it as it comes in readily available.

Lambda Function

Unlike def, this function has no name. It's simple and straightforward, requiring only the argument(s) and expression, as well as the lambda keyword, and requiring only one line of code. It's essentially a second writing function. Additionally, this function just returns the expression and not the data. It's limited to one line and the same number of parameters as previously.

The animated gif below shows how the def and lambda functions work.

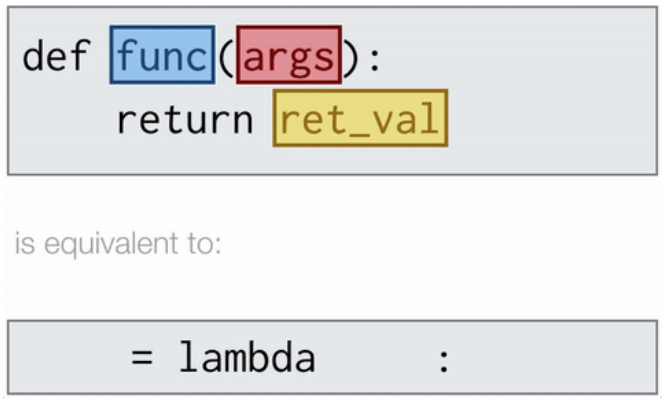

Image Source: images.google.com

History – Lambda:

Python is not the only language that supports the Lambda. Alonza Church, an American mathematician, and logician made a crucial contribution in the year 1930 – Lambda Calculus (computation based on abstraction purpose).

Need for Lambda function

To reduce the number of lines needed to specify functionality. This function is utilized because it is straightforward. Unlike the def function, we may call this lambda function right away (can be called but not immediately).

Another important consideration is when a function is only required sometimes rather than on a regular basis.

Difference between DEF and LAMBDA function

We'll look at the differences:

S.No	Def function	Lambda function
1	Keyword: def	Keyword: lambda
2	Return statement is must	no need for a return statement
3	Return value: Integer	Return value: function object
4	computation time is slower	computational time is faster
5	Depend on usage, the number of lines may vary	Simplified and Single line

Image Source: Author

Basic code for def function,

```
#by def def adds(a,b): return a+b print(adds(10,43)) 53
```

Basic code for lambda function,

```
#by lambda adds = lambda a,b: a+b print(adds(10,43)) 53
```

Where to use the Lambda function?

If you need to employ a function but don't want to use a def or a nameless condition, lambda is the way to go. Temporary or occasional use is the second essential use. This function may be used with other higher-order functions like map(), apply(), reduce(), and so on.

How to use the Lambda function?

The lambda function's essential structure,

a = lambda b: bb**

where,

a = function objects that stores the result

lambda = function keyword

b = argument

b**b = one line expression

Because it separates the expression from the argument, the semicolon after the lambda function is important (s).

```
(lambda s: s/2.5)(5)
```

```
2.0
```

In this simple example, lambda is the lambda function's keyword, s is an argument, (s/2.5) is the expression, and 5 is the argument's result.

Example :

Some of the basic programming by using lambda functions is as follows,

```
a = lambda : "hello world!"


print(a())


hello world!
```

Calculating cube function by using lambda,

```
# calculate cubes using lambda


cubes = lambda a: a*a*a


print('Calculating cubes by using lambda: ', cubes(5))


Calculating cubes by using lambda: 125
```

Calculating for scalar value by using lambda function,

```
(lambda x: x*2)(12)


24
```

Applications of Lambda function

Unlike other topics, there are no specific applications here; the scope is the only application, as we will see through some Python programming. Before we go into the scope of the Lambda function with other functions, we must first establish a Dataframes with the aid of the pandas library, as seen below.

```
#importing libraries import pandas as pd #creating dataframe data = pd.DataFrame({ 'S.No':[1,2,3,4,5], 'name':['Anand','Ramesh','Surya','Dinesh','Dhamu'], 'expert':['ML','DS','Network','Banking','Electronics'], 'income':[45000,50000,75000,80000,85000], 'place':['Cheyyar','Chennai','Bangalore','Malaysia','Cheannai'], 'Experience':[3,2,10,9,5] }) data
```

Image Source: Author

Following the construction of Dataframes, we will examine how this function is useful for additional tasks such as Apply(), Filter(), Map(), Reduce(), and Conditional expressions.

Lambda function with Apply()

Let us imagine that after the 2021 year, all businesses have made the decision to improve their performance by 5500 irrespective of their experience and domain-specific, thus there are many alternatives to achieve with Python programming, but to keep it simple and concise, we use the function called Apply (),

```
data['income'] = data.apply(lambda i: i['income']+5500, axis=1) data
```

Image Source: Author

The idea is easy to understand; we are making an effort to raise the income column alone (axis=1) from the Dataframes' data, but not others.

Lambda function with Filter()

If we need to view people with more than 5 years of experience from the aforementioned Dataframes' data, we can utilize the Filter function (),

```
#filter function for segregation from the dataframe list(filter(lambda i: i>5, data['Experience'])) [10, 9]
```

The idea is, it's also fairly easy here; we need to filter the candidate's experience more than 5 years and explain how many years? In addition to the lambda function.

Lambda function with Map()

If this is the situation, then the Map() function is utilized to process it in order to improve employee satisfaction through appraisal by 15%.

```
#map() function data['income']=list(map(lambda i: int(i+i*0.15),data['income'])) data
```

Image Source: Author

Lambda function with Reduce()

Assume the names in the Dataframes are friends, and if you need to determine their total salary, we will use the reduce() function to process it, and in order to use this function, we will need to use the functools library.

```
import functools functools.reduce(lambda a,b: a+b,data['income']) 479404
```

Which is just addition of 66786+73398+106461+113073+119686 = 479404. It's similar to addition function.

Lambda function with Conditional statement

A conditional statement, such as if…then, if…then, if…then, for loop, and while loop, makes our program more interesting and robust. When it comes to statements, because of this circumstance, the lambda function is more powerful.

If we need to establish category conditions for the degree of expertise based on their experience, we can make use of conditional statements such as,

```
data['category']=data['Experience'].apply(lambda x: 'Expert level' if x>=5 else 'Begineer') data
```

	S.No	name	expert	income	place	Experience	category
0	1	Anand	ML	66786	Cheyyar	3	Begineer
1	2	Ramesh	DS	73398	Chennai	2	Begineer
2	3	Surya	Network	106461	Bangalore	10	Expert level
3	4	Dinesh	Banking	113073	Malaysia	9	Expert level
4	5	Dhamu	Electronics	119686	Cheannai	5	Expert level

Image Source: Author

Lambda function within a user-defined function

Another important application is using this function in def function,

```
#lambda in user defined def ads(x): return(lambda y:x+y) a = ads(4) print(ad(8)) 12
```

Advantages of Lambda function

We'll look at a few distinguishing qualities that distinguish Lambda functions from other types of functions.

1. Ease of creating the function
2. Variables with few options

Disadvantages of Lambda function

Let's consider its drawbacks in the below section-

1. Despite Python's popularity as the world's most popular programming language, the Lambda is not user-friendly.
2. The Lambda function will be a problem if we need to build a complex function.
3. Strictly limited to a single expression
4. Although it cannot access a global variable, it can access the lone local variable.
5. The simplicity of the Lambda function, which is a single statement, cannot be acceptable in all circumstances.
6. For clarity, most Python functions and modules contain documentation, which is another drawback of the Lambda function.

Conclusion

Did you find this article to be useful? Please leave your thoughts/opinions in the comments area below. Learn from your mistakes is my favorite quote; if you find something incorrect, simply highlight it; I am eager to learn from students like you.

About me in short, I am Premanand.S, Assistant Professor Jr and a researcher in Machine Learning. Love to teach and love to learn new things in Data Science. Mail me for any doubt or mistake, er.anandprem@gmail.com, and my Linkedin https://www.linkedin.com/in/premsanand/

Happy learning!!

Article Url - https://www.analyticsvidhya.com/blog/2021/10/lambda-function-a-better-understanding/

Premanand S