Python Lambda Functions

This study anonymous, commonly called lambda functions in Python. A lambda function can take n number of arguments at a time. But it returns only one argument at a time. We will understand what they are, how to execute them, and their syntax.

What are Lambda Functions in Python?

Lambda Functions in Python are anonymous functions, implying they don't have a name. The **def** keyword is needed to create a typical function in Python, as we already know. We can also use the **lambda** keyword in Python to define an unnamed function.

Syntax

The syntax of the Lambda Function is given below -

- lambda **arguments: expression**

This function accepts any count of inputs but only evaluates and returns one expression. That means it takes many inputs but returns only one output.
Lambda functions can be used whenever function arguments are necessary. In addition to other forms of formulations in functions, it has a variety of applications in certain coding domains. It's important to remember that according to syntax, lambda functions are limited to a single statement.

Example

Here we share some examples of lambda functions in Python for learning purposes.

Program Code 1:

Now we gave an example of a lambda function that adds 4 to the input number is shown below.

```
# Code to demonstrate how we can use a lambda function for adding 4 numbe
r
add = lambda num: num + 4
print(add(6))
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
10
```

Here we explain the above code. The lambda function is "lambda num: num+4" in the given programme. The parameter is num, and the computed and returned equation is num * 4.

There is no label for this function. It generates a function object associated with the "add" identifier. We can now refer to it as a standard function. The lambda statement, "lambda num: num+4", is written using the add function, and the code is given below:

Program Code 2:

Now we gave an example of a lambda function that adds 4 to the input number using the add function. The code is shown below -

```
def add( num ):
   return num + 4
print( add(6) )
```

Output:

Now we compile the above code in Python, and after successful compilation, we run it. Then the output is given below -

```
10
```

Program Code 3:

Now we gave an example of a lambda function that multiply 2 numbers and return one result. The code is shown below -

```
a = lambda x, y : (x * y)
print(a(4, 5))
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
20
```

Program Code 4:

Now we gave another example of a lambda function that adds 2 numbers and return one result. The code is shown below -

```
a = lambda x, y, z : (x + y + z)
print(a(4, 5, 5))
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
14
```

What's the Distinction Between Lambda and Def Functions?

Let's glance at this instance to see how a conventional def defined function differs from a function defined using the lambda keyword. This program calculates the reciprocal of a given number:

Program Code:

```
# Python code to show the reciprocal of the given number to highlight the
 difference between def() and lambda().
def reciprocal( num ):
    return 1 / num

lambda_reciprocal = lambda num: 1 / num

# using the function defined by def keyword
print( "Def keyword: ", reciprocal(6) )

# using the function defined by lambda keyword
print( "Lambda keyword: ", lambda_reciprocal(6) )
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
Def keyword:  0.16666666666666666
Lambda keyword:  0.16666666666666666
```

Explanation:

The reciprocal() and lambda_reciprocal() functions act similarly and as expected in the preceding scenario. Let's take a closer look at the sample above:

Both of these yield the reciprocal of a given number without employing Lambda. However, we wanted to declare a function with the name reciprocal and send a number to it while executing def. We were also required to use the return keyword to provide the output from wherever the function was invoked after being executed.

Using Lambda: Instead of a "return" statement, Lambda definitions always include a statement given at output. The beauty of lambda functions is their convenience. We need not allocate a lambda expression to a variable because we can put it at any place a function is requested.

Using Lambda Function with filter()

The filter() method accepts two arguments in Python: a function and an iterable such as a list.
The function is called for every item of the list, and a new iterable or list is returned that holds just those elements that returned True when supplied to the function.
Here's a simple illustration of using the filter() method to return only odd numbers from a list.

Program Code:

Here we give an example of lambda function with filter() in Python. The code is given below -

```
# This code used to filter the odd numbers from the given list
list_ = [35, 12, 69, 55, 75, 14, 73]
odd_list = list(filter( lambda num: (num % 2 != 0) , list_ ))
print('The list of odd number is:',odd_list)
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
The list of odd number is: [35, 69, 55, 75, 73]
```

Using Lambda Function with map()

A method and a list are passed to Python's map() function.
The function is executed for all of the elements within the list, and a new list is produced with elements generated by the given function for every item.
The map() method is used to square all the entries in a list in this example.

Program Code:

Here we give an example of lambda function with map() in Python. Then code is given below -

```
#Code to calculate the square of each number of a list using the map() fu
nction
numbers_list = [2, 4, 5, 1, 3, 7, 8, 9, 10]
squared_list = list(map( lambda num: num ** 2 , numbers_list ))
print( 'Square of each number in the given list:' ,squared_list )
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
Square of each number in the given list: [4, 16, 25, 1, 9, 49, 64, 81, 100]
```

Using Lambda Function with List Comprehension

In this instance, we will apply the lambda function combined with list comprehension and the lambda keyword with a for loop. Using the Lambda Function with List Comprehension, we can print the square value from 0 to 10. For printing the square value from 0 to 10, we create a loop range from 0 to 11.

Program Code:

Here we give an example of lambda function with List Comprehension in Python. Then code is given below -

```
#Code to calculate square of each number of lists using list comprehension
squares = [lambda num = num: num ** 2 for num in range(0, 11)]
for square in squares:
        print('The square value of all numbers from 0 to 10:',square(), end = " "
)
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
The square value of all numbers from 0 to 10: 0 1 4 9 16 25 36 49 64 81 100
```

Using Lambda Function with if-else

We will use the lambda function with the if-else block. In the program code below, we check which number is greater than the given two numbers using the if-else block.

Program Code:

Here we give an example of a lambda function with an if-else block in Python. The code is given below -

```
# Code to use lambda function with if-else
Minimum = lambda x, y : x if (x < y) else y
print('The greater number is:', Minimum( 35, 74 ))
```

Output:

Now we compile the above code in python, and after successful compilation, we run it. Then the output is given below -

```
The greater number is: 35
```

Using Lambda with Multiple Statements

Multiple expressions are not allowed in lambda functions, but we can construct 2 lambda functions or more and afterward call the second lambda expression as an argument to the first. We are sorting every sub-list from the given list in the below program. Let us use lambda to discover the third largest number from every sub-list.

Program Code:

Here we give an example of lambda function with Multiple Statements in Python. The code is given below -

```
# Code to print the third largest number of the given list using the lamb
da function

my_List = [ [3, 5, 8, 6], [23, 54, 12, 87], [1, 2, 4, 12, 5] ]
# sorting every sublist of the above list
sort_List = lambda num : ( sorted(n) for n in num )
# Getting the third largest number of the sublist
third_Largest = lambda num, func : [ l[ len(l) - 2] for l in func(num)]

result = third_Largest( my_List, sort_List)
print('The third largest number from every sub list is:', result )
```

Output:

Now we compile the above code, in python and after successful compilation, we run it. Then the output is given below -

```
The third largest number from every sub list is: [6, 54, 5]
```