Python Lambda Functions

Python Lambda Functions are anonymous function means that the function is without a name. As we already know that the *def* keyword is used to define a normal function in Python. Similarly, the *lambda* keyword is used to define an anonymous function in Python.

Python Lambda Function Syntax:
lambda arguments: expression

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming besides other types of expressions in functions.

Example: Lambda Function Example

- Python3

```
# Python program to demonstrate
# lambda functions


string ='GeeksforGeeks'

# lambda returns a function object
print(lambda string : string)
```

Output
<function <lambda> at 0x7f65e6bbce18>

In this above example, the lambda is not being called by the print function but simply returning the function object and the memory location where it is stored.

So, to make the print to print the string first we need to call the lambda so that the string will get pass the print.

Example:

- Python3

```
# Python program to demonstrate
# lambda functions


x ="GeeksforGeeks"
# lambda gets pass to print
(lambda x : print(x))(x)
```

Output
GeeksforGeeks

Difference Between Lambda functions and def defined function

Let's look at this example and try to understand the difference between a normal def defined function and lambda function. This is a program that returns the cube of a given value:

- Python

```
# Python code to illustrate cube of a number
# showing difference between def() and lambda().
def cube(y):
    return y*y*y

lambda_cube = lambda y: y*y*y

# using the normally
# defined function
print(cube(5))

# using the lambda function
print(lambda_cube(5))
```

Output:
125

125

As we can see in the above example both the cube() function and lambda_cube() function behave the same and as intended. Let's analyze the above example a bit more:

- **Without using Lambda:** Here, both of them return the cube of a given number. But, while using def, we needed to define a function with a name cube and needed to pass a value to it. After execution, we also needed to return the result from where the function was called using the *return* keyword.
- **Using Lambda:** Lambda definition does not include a "return" statement, it always contains an expression that is returned. We can also put a lambda definition anywhere a function is expected, and we don't have to assign it to a variable at all. This is the simplicity of lambda functions.

Let's see some more commonly used examples of lambda functions.

Example 1: Python Lambda Function with List Comprehension

In this example, we will use the lambda function with list comprehension and lambda with for loop. We will try to print the table of 10.

- Python3

```
tables = [lambda x=x: x*10 for x in range(1, 11)]

for table in tables:
    print(table())
```

Output

10

20

30

40

50

60

70

80

90

100

Example 2: Python Lambda Function with if-else

- Python3

```
# Example of lambda function using if-else
Max = lambda a, b : a if(a > b) else b

print(Max(1, 2))
```

Output

2

Example 3: Python Lambda with Multiple statements

Lambda functions does not allow multiple statements, however, we can create two lambda functions and then call the other lambda function as a parameter to the first function. Let’s try to find the second maximum element using lambda.

- Python3

```
List = [[2,3,4],[1, 4, 16, 64],[3, 6, 9, 12]]

# Sort each sublist
sortList = lambda x: (sorted(i) for i in x)

# Get the second largest element
secondLargest = lambda x, f : [y[len(y)-2] for y in f(x)]
res = secondLargest(List, sortList)

print(res)
```

Output

[3, 16, 9]

In the above example, we have created a lambda function that sorts each sublist of the given list. Then this list is passed as the parameter to the second lambda function which returns the n-2 element from the sorted list where n is the length of the sublist.

Lambda functions can be used along with built-in functions like filter(), map() and reduce().

Using lambda() Function with filter()

The filter() function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence "sequence", for which the function returns True. Here is a small program that returns the odd numbers from an input list:

Example 1:

- Python

```
# Python code to illustrate
# filter() with lambda()
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]

final_list = list(filter(lambda x: (x%2 != 0) , li))
print(final_list)
```

Output:
[5, 7, 97, 77, 23, 73, 61]

Example 2:

- Python3

```
# Python 3 code to people above 18 yrs
ages = [13, 90, 17, 59, 21, 60, 5]

adults = list(filter(lambda age: age>18, ages))

print(adults)
```

Output:
[90, 59, 21, 60]

Using lambda() Function with map()

The map() function in Python takes in a function and a list as an argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item. Example:

Example 1:

- Python

```
# Python code to illustrate
# map() with lambda()
# to get double of a list.
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
```

```
final_list = list(map(lambda x: x*2, li))
print(final_list)
```

Output:
[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]

Example 2:

- Python3

```
# Python program to demonstrate
# use of lambda() function
# with map() function
animals = ['dog', 'cat', 'parrot', 'rabbit']

# here we intend to change all animal names
# to upper case and return the same
uppered_animals = list(map(lambda animal: str.upper(animal), animals))

print(uppered_animals)
```

Output:
['DOG', 'CAT', 'PARROT', 'RABBIT']

Using lambda() Function with reduce()

The reduce() function in Python takes in a function and a list as an argument. The function is called with a lambda function and an iterable and a new reduced result is returned. This performs a repetitive operation over the pairs of the iterable. The reduce() function belongs to the ***functools*** module.

Example 1:

- Python

```
# Python code to illustrate
# reduce() with lambda()
# to get sum of a list

from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y: x + y), li)
print (sum)
```

Output:
193

Here the results of previous two elements are added to the next element and this goes on till the end of the list like (((((5+8)+10)+20)+50)+100).

Example 2:

- Python3

```
# python code to demonstrate working of reduce()
```

```
# with a lambda function

# importing functools for reduce()
import functools

# initializing list
lis = [ 1 , 3, 5, 6, 2, ]

# using reduce to compute maximum element from list
print ("The maximum element of the list is : ",end="")
print (functools.reduce(lambda a,b : a if a > b else b,lis))
```

Output:

```
The maximum element of the list is : 6
```