**What is Lambda Function in Python?**

A **Lambda Function in Python** programming is an anonymous function or a function having no name. It is a small and restricted function having no more than one line. Just like a normal function, a Lambda function can have multiple arguments with one expression.

In Python, lambda expressions (or lambda forms) are utilized to construct anonymous functions. To do so, you will use the **lambda**keyword (just as you use **def**to define normal functions). Every anonymous function you define in Python will have 3 essential parts:

* The lambda keyword.
* The parameters (or bound variables), and
* The function body.

A lambda function can have any number of parameters, but the function body can only contain **one** expression. Moreover, a lambda is written in a single line of code and can also be invoked immediately.

## Syntax and Examples

The formal syntax to write a lambda function is as given below:

**lambda p1, p2: expression**

Here, p1 and p2 are the parameters which are passed to the lambda function. You can add as many or few parameters as you need.

However, notice that we do not use brackets around the parameters as we do with regular functions. The last part (expression) is any valid python expression that operates on the parameters you provide to the function.

### Example 1

Now that you know about lambdas let’s try it with an example. So, open your IDLE and type in the following:

adder = lambda x, y: x + y

print (adder (1, 2))

**Here is the output:**

3

### Code Explanation

Here, we define a variable that will hold the result returned by the lambda function.

**1.** The lambda keyword used to define an anonymous function.

**2.** x and y are the parameters that we pass to the lambda function.

**3.** This is the body of the function, which adds the 2 parameters we passed. Notice that it is a single expression. You cannot write multiple statements in the body of a lambda function.

**4.** We call the function and print the returned value.

## Using lambdas with Python built-ins

Lambda functions provide an elegant and powerful way to perform operations using built-in methods in Python. It is possible because lambdas can be invoked immediately and passed as an argument to these functions.

### IIFE in Python Lambda

**IIFE** stands for **immediately invoked function execution.**It means that a lambda function is callable as soon as it is defined. Let’s understand this with an example; fire up your IDLE and type in the following:

(lambda x: x + x)(2)

**Here is the output and code explanation:**

This ability of lambdas to be invoked immediately allows you to use them inside functions like map() and reduce(). It is useful because you may not want to use these functions again.

## lambdas in filter()

The filter function is used to select some particular elements from a sequence of elements. The sequence can be any iterator like lists, sets, tuples, etc.

The elements which will be selected is based on some pre-defined constraint. It takes 2 parameters:

* A function that defines the filtering constraint
* A sequence (any iterator like lists, tuples, etc.)

For example,

sequences = [10,2,8,7,5,4,3,11,0, 1]

filtered\_result = filter (lambda x: x > 4, sequences)

print(list(filtered\_result))

**Here’s the output:**

[10, 8, 7, 5, 11]

**Code Explanation:**

**1.** In the first statement, we define a list called sequences which contains some numbers.

**2.** Here, we declare a variable called filtered\_result, which will store the filtered values returned by the filter() function.

**3.** A lambda function which runs on each element of the list and returns true if it is greater than 4.

**4.**Print the result returned by the filter function.

## lambdas in map()

the map function is used to apply a particular operation to every element in a sequence. Like filter(), it also takes 2 parameters:

1. A function that defines the op to perform on the elements
2. One or more sequences

For example, here is a program that prints the squares of numbers in a given list:

sequences = [10,2,8,7,5,4,3,11,0, 1]

filtered\_result = map (lambda x: x\*x, sequences)

print(list(filtered\_result))

**Output:**

[100, 4, 64, 49, 25, 16, 9, 121, 0, 1]

Code Explanation:

1. Here, we define a list called sequences which contains some numbers.
2. We declare a variable called filtered\_result which will store the mapped values
3. A lambda function which runs on each element of the list and returns the square of that number.
4. Print the result returned by the map function.

## lambdas in reduce()

The reduce function, like map(), is used to apply an operation to every element in a sequence. However, it differs from the map in its working. These are the steps followed by the reduce() function to compute an output:

**Step 1)**Perform the defined operation on the first 2 elements of the sequence.

**Step 2)**Save this result

**Step 3)**Perform the operation with the saved result and the next element in the sequence.

**Step 4)**Repeat until no more elements are left.

It also takes two parameters:

1. A function that defines the operation to be performed
2. A sequence (any iterator like lists, tuples, etc.)

For example, here is a program that returns the product of all elements in a list:

from functools import reduce

sequences = [1,2,3,4,5]

product = reduce (lambda x, y: x\*y, sequences)

print(product)

**Here is the output:**

120

Code Explanation:

1. Import reduce from the functools module
2. Here, we define a list called sequences which contains some numbers.
3. We declare a variable called product which will store the reduced value
4. A lambda function that runs on each element of the list. It will return the product of that number as per the previous result.
5. Print the result returned by the reduce function.

## Why (and why not) use lambda functions?

As you will see in the next section, lambdas are treated the same as regular functions at the interpreter level. In a way, you could say that lambdas provide compact syntax for writing functions which return a single expression.

However, you should know when it is a good idea to use lambdas and when to avoid them. In this section, you will learn some of the design principles used by python developers when writing lambdas.

One of the most common use cases for lambdas is in functional programming as Python supports a paradigm (or style) of programming known as functional programming.

It allows you to provide a function as a parameter to another function (for example, in map, filter, etc.). In such cases, using lambdas offer an elegant way to create a one-time function and pass it as the parameter.

### When should you not use Lambda?

You should never write complicated lambda functions in a production environment. It will be very difficult for coders who maintain your code to decrypt it. If you find yourself making complex one-liner expressions, it would be a much superior practice to define a proper function. As a best practice, you need to remember that simple code is always better than complex code.