# Python lambda

In Python, anonymous function means that a function is without a name. As we already know that def keyword is used to define the normal functions and the lambda keyword is used to create anonymous functions. It has the following syntax:

**Syntax**

lambda arguments : expression

* This function can have any number of arguments but only one expression, which is evaluated and returned.

Example:

g = lambda x: x\*x\*x

print(g(7))

def power(n):

    return lambda a : a \*\* n

base = power(2) #base = lambda a : a \*\* n

print("8 powerof 2 = ", base(8)) # 64

**MAP**

**map()** function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

# Double all numbers using map and lambda

numbers = (1, 2, 3, 4)

result = map(lambda x: x + x, numbers)

print(list(result))

Output :

[2, 4, 6, 8]

# Add two lists using map and lambda

numbers1 = [1, 2, 3]

numbers2 = [4, 5, 6]

result = map(lambda x, y: x + y, numbers1, numbers2)

print(list(result))

Output :

[5, 7, 9]

**FILTER**

The filter() method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

# a list contains both even and odd numbers.

seq = [0, 1, 2, 3, 5, 8, 13]

# result contains odd numbers of the list

result = filter(lambda x: x % 2 != 0, seq)

print(list(result))

# result contains even numbers of the list

result = filter(lambda x: x % 2 == 0, seq)

print(list(result))

**REDUCE**

The reduce(fun,seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.This function is defined in “functools” module.

Working :

At first step, first two elements of sequence are picked and the result is obtained.

Next step is to apply the same function to the previously attained result and the number just succeeding the second element and the result is again stored.

This process continues till no more elements are left in the container.

The final returned result is returned and printed on console.

# python code to demonstrate working of reduce()

# importing functools for reduce()

import functools

# initializing list

lis = [ 1 , 3, 5, 6, 2, ]

# using reduce to compute sum of list

print ("The sum of the list elements is : ",end="")

print (functools.reduce(lambda a,b : a+b,lis))

# 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 sum of the list elements is : 17

The maximum element of the list is : 6

reduce() can also be combined with [operator functions](https://www.geeksforgeeks.org/operator-functions-in-python-set-1/) to achieve the similar functionality as with lambda functions and makes the code more readable.

import functools

import operator

# initializing list

lis = [ 1 , 3, 5, 6, 2, ]

print ("The sum of the list elements is : ",end="")

print (functools.reduce(operator.add,lis))

print ("The product of list elements is : ",end="")

print (functools.reduce(operator.mul,lis))

print ("The concatenated product is : ",end="")

print (functools.reduce(operator.add,["geeks","for","geeks"]))

Output

The sum of the list elements is : 17

The product of list elements is : 180

The concatenated product is : geeksforgeeks

**reduce() vs accumulate()**

Both reduce() and accumulate() can be used to calculate the summation of a sequence elements. But there are differences in the implementation aspects in both of these.

* reduce() is defined in “functools” module, accumulate() in “itertools” module.
* reduce() stores the intermediate result and only returns the final summation value. Whereas, accumulate() returns a iterator containing the intermediate results. The last number of the iterator returned is summation value of the list.

|  |
| --- |
| import itertools    # importing functools for reduce()  import functools    # initializing list  lis = [ 1, 3, 4, 10, 4 ]    # priting summation using accumulate()  print ("The summation of list using accumulate is :",end="")  print (list(itertools.accumulate(lis,lambda x,y : x+y)))    # priting summation using reduce()  print ("The summation of list using reduce is :",end="")  print (functools.reduce(lambda x,y:x+y,lis)) |

Output:

The summation of list using accumulate is :[1, 4, 8, 18, 22]

The summation of list using reduce is :22