Python Fundamentals day 13

Today's Agenda

- General use of lambda functions
- filter()
- reduce()
- map()
- Other examples

General use of lambda functions

Lambda functions are used whenever the following built-in functions are used

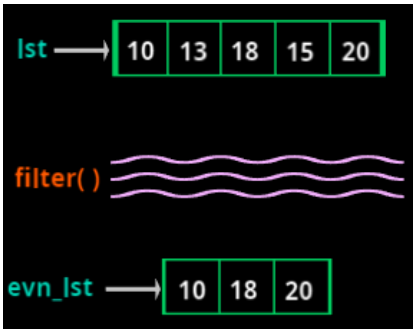
- filter()
- reduce()
- map()

filter( )

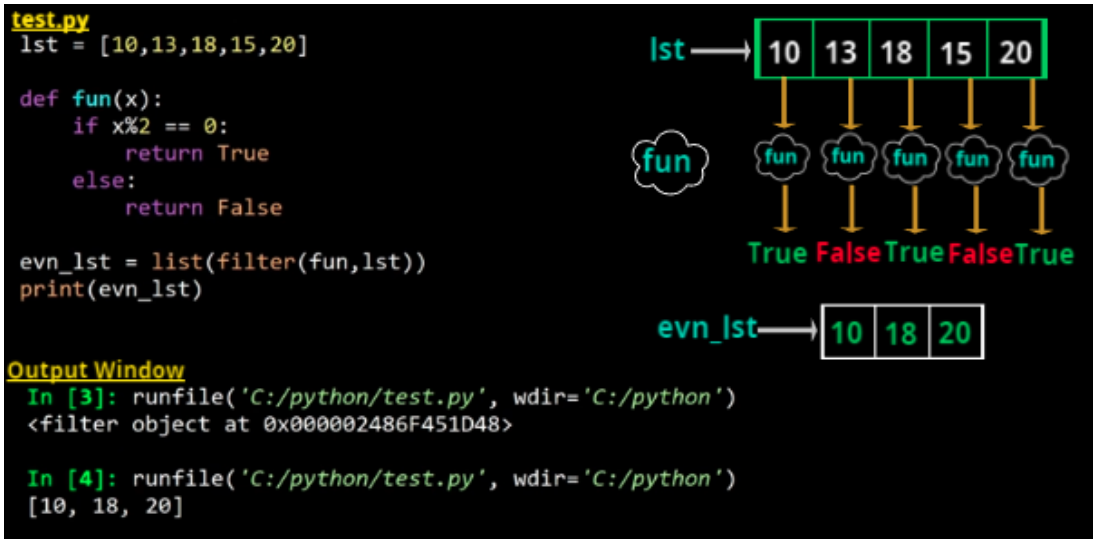
The **filter()** method filters the given sequence with the help of a function that tests each element in the sequence to be true or not.

Syntax:

filter (function,sequence)



Let us try to understand with the normal function declaration using def and then compare with lambda function



To each element in the list lst, fun function is applied and checks if the result is true or false. All the true resulted elements are collected in a new list evn_lst and displayed. In the output we can see that evn_lst is a filter object, to get the list we have to type cast it to list. And the new output displays the filtered list evn_lst.

Let us now see how the code changes when we use lambda function

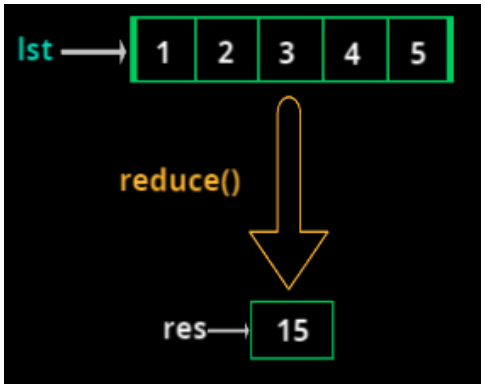
We can see how a six line code has been reduced to a single line code with less complexity.

reduce( )

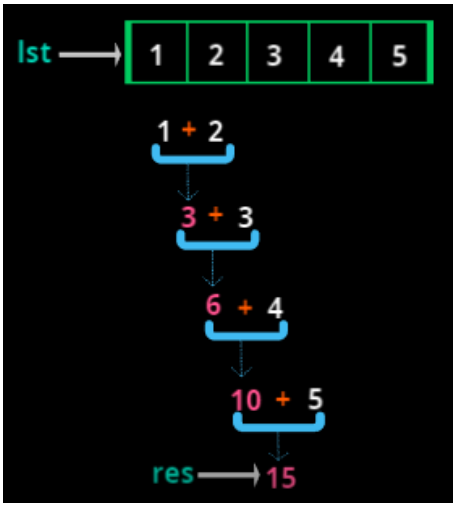
The reduce() function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along. This function is defined in **"functools"** module.

Syntax:

reduce (function,sequence)

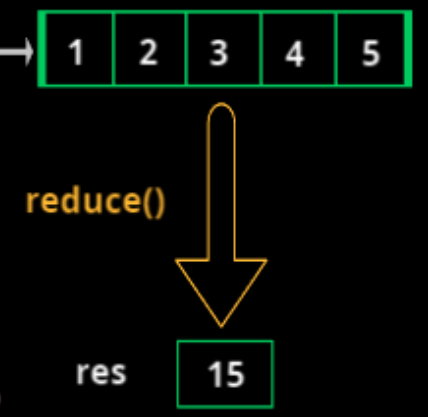


In the above diagram we are trying to reduce the list lst by adding all the elements and storing the result in res.



Let us see how to use reduce() with normal def function before comparing with lambda function

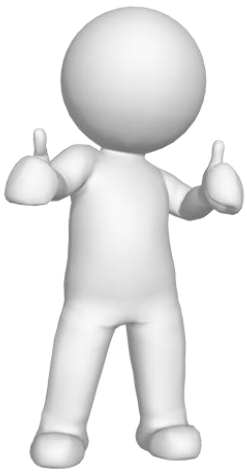
```
test.py
from functools import reduce

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

def fun(x,y):
    return x+y

res = reduce(fun,lst)
print(res)

Output Window
In [3]: runfile('C:/python/test.py', wdir='C:/python')
15
```



Using def function we had to write the logic in 4 lines let us see how the code changes with the use of lambda function

```
test.py
from functools import reduce

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

'''def fun(x,y):
    return x+y

res = reduce(fun,lst)
print(res) '''

result = reduce(lambda x,y: x+y,lst)
print(result)

Output Window
In [3]: runfile('C:/python/test.py', wdir='C:/python')
15
```

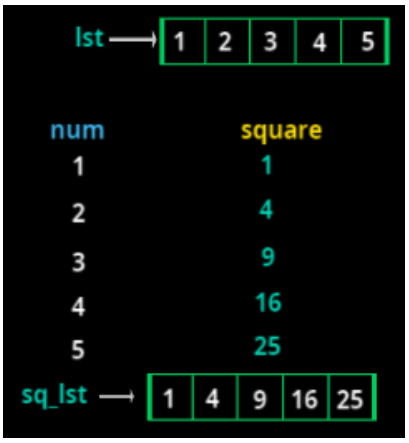
We can see that the code reduced to a single line statement with less complexity.



map( )

map() function returns a **map** object of the results after applying the given function to each item of a given sequence (list, tuple etc.)

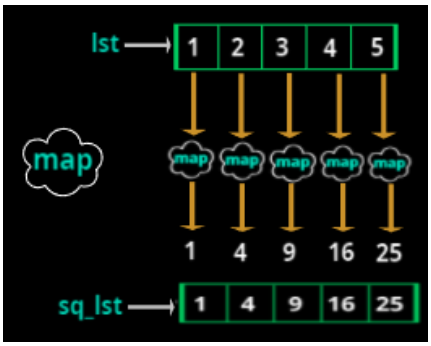
For example let's take a list with certain numbers and map it with its squared values.



Syntax:

map (function,sequence)

Let us try to code the above logic using def function first and then use lambda function




```
lst=[1,2,3,4,5]

def fun(x):
    return x**2

sq_lst=list(map(fun,lst))
print(sq_lst)
```

Output:

```
In [6]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
<map object at 0x0000021303701C08>

In [7]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
[1, 4, 9, 16, 25]
```

In the above output we see that similar to filter, in mapping also map object is created and to see the result in list format we have to type cast it to list.

Let us see the same code with lambda function now

```
lst=[1,2,3,4,5]

'''def fun(x):
    return x**2

sq_lst=list(map(fun,lst))
print(sq_lst)'''
sq_lst=list(map(lambda x : x**2,lst))
print(sq_lst)
```

Output:

```
In [8]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
[1, 4, 9, 16, 25]
```

Certainly the code has again reduced with less complexity here as well.

Other examples

In python a function can not only be given as input to a function. If you choose to, you can design a function in such a way that about calling a function it would return a function as output to you.

Let us see how

```
def fun1(num):
    return lambda x : x*num

result=fun1(2)(5)  #(num)(x)
print(result)
```

Output:

```
In [9]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
10
```

But what if we want to call it multiple times?? Let us see the solution

```
def fun1(num):
    return lambda x : x*num

'''result=fun1(2)(5)  #(num)(x)
print(result)'''

fun2=fun1(2) #first function call
print(fun2(5)) #call to lambda function
```

Output:

```
In [10]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
10
```

Like this we can call lambda function multiple times.

Now let us see using fun1() how can we create a **mathematical table of any given number**

```
def fun1(num):
    return lambda x : x*num

n= int(input("Enter a number\n"))
math_table=fun1(n)

for i in range(1,11):
    print(n,"X",i,"=",math_table(i))
```

Output:

```
In [11]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
7
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70
```

```
In [12]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')

Enter a number
12
12 X 1 = 12
12 X 2 = 24
12 X 3 = 36
12 X 4 = 48
12 X 5 = 60
12 X 6 = 72
12 X 7 = 84
12 X 8 = 96
12 X 9 = 108
12 X 10 = 120
```

Similarly we can give any number and generate its mathematical table. Why not try yourself and verify