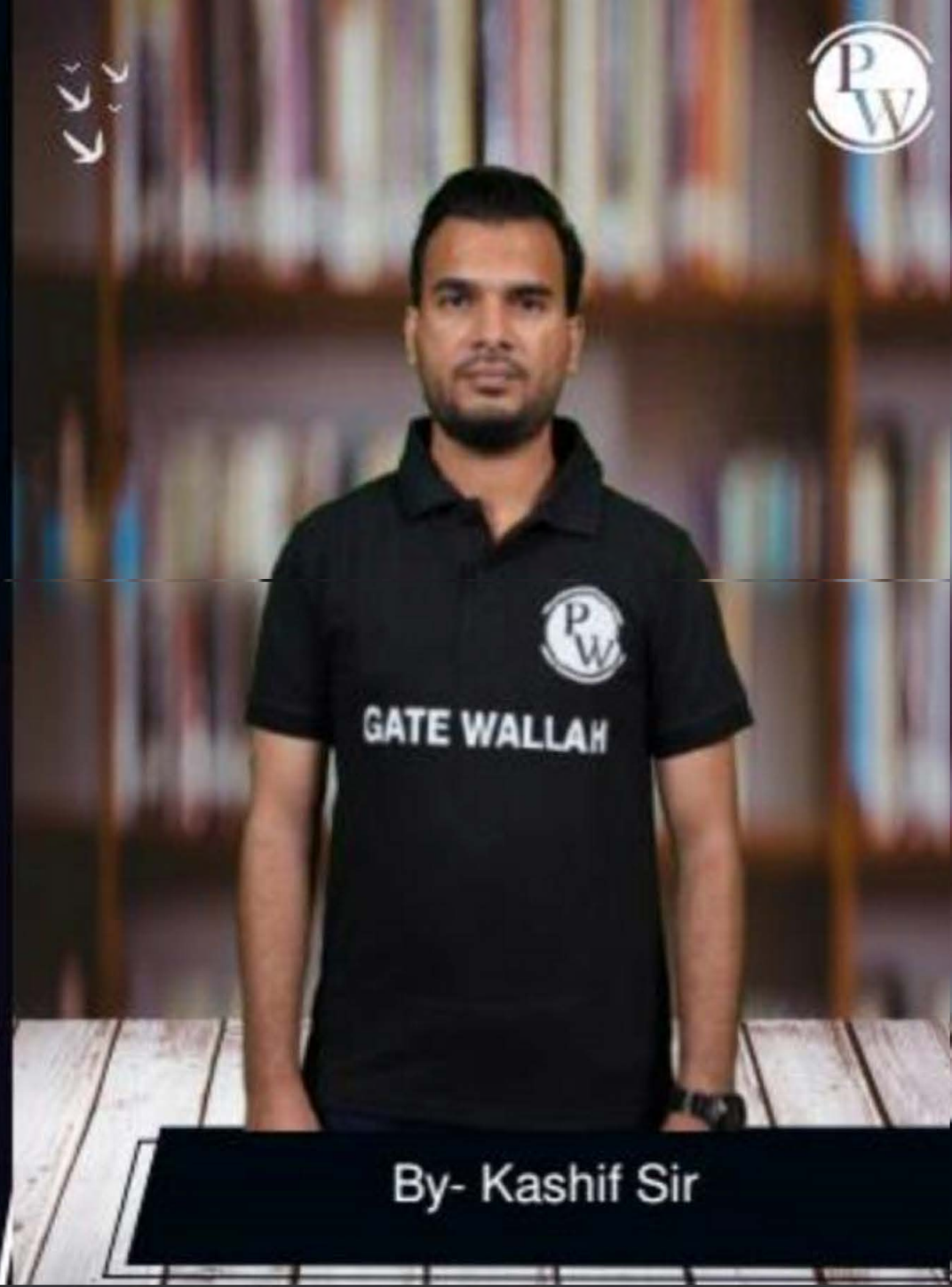
Data Science & Artificial Intelligence

Python for Data Science

Functions and Functional Programming

Lecture No. 04



By- Kashif Sir

Topics *to be covered*

- Functional Programming

RECAP

- Direct Recursion
- Indirect Recursion
- Nested Recursion

GATE CSE ↓ ⇒ Recursion Questions

- Recursion Function was given ⇒ You have to tell what that function was actually doing

Functional Programming

(Functional programming) is a programming paradigm that treats computation as an evaluation of mathematical functions.

lambda

In python lambda functions are anonymous functions, means they don't have name.

Lambda is a keyword which is used to creat functions without names.

```
def fun(n):
    return n**2
```

```
lambda n : n**2
```

```
lambda x,y : x+y
```

Syntax: lambda arguments : expression

```
fun(10)
```

100

```
a = lambda n : n**2
print( type(a) )
```

function

```
a(10)
```

100

```
b = lambda x,y : x+y
print( b(2,3) )
```

5

Generally people use lambda function when the function is of single line or if the function to be used only one time

a(x,y) → a = lambda x, y : x+y

a(x,y) = lambda x, y : x-y

a = lambda x,y,z,a : x*y+z-a

Lambda function can take any no. of arguments

a(4, 7, 8, 1)

lf1 = lambda : 10

lf2 = lambda : print(10)

print(type(lf1)) => function

print(type(lf2)) => function

lf1( ) => returning 10

lf2( ) => printing 10

```
def myfun(n):
    return lambda a : a*n
```

x = myfun(2)

lambda a : a*2

print(x(10))

x([1,2,3])

→ l = myfun(3)

→ print(l(7))

21

[1,2,3]*2

[1,2,3,1,2,3]

a = lambda x : lambda y : x+y

b = a(3)

↳ lambda y : 3+y

b(10) ⇒ 13

a(3)(5) = 8

filter()

Syntax: filter(function, Iterable)

return filter object

⇒ List = [10, 90, 70, 25, 60, 41, 80]

newlist = list(filter(fun, List))

def fun(x):
    if x > 40:
        return True
    else
        return False

newlist = [90, 70, 60, 41, 80]

list(filter(lambda x : x > 40, List))

21

```
data = [5, 12, 17, 18, 24, 32]
filtered = list(filter(lambda x: x%12==0, data))
print(filtered)
```

A) [12, 18, 24, 32]

B) [5, 17]

C) [5, 12, 17]

D) [18, 24, 32]

[12, 18, 24, 32]

```
def apply(f, x):
    return f(x)

print(apply(lambda x: x**3, 3))
```

A) 9

B) 27

C) 81

D) Error

map()

map(function, iterable)

return type => map object

list1 = list(map(lambda x : x**2 , [1, 2, 3, 7]))

list1 = [1, 4, 9, 16]

[3, 6, 9, 12]

list2 = list(map(lambda x : x*3 , [1, 2, 3, 4]))

reduce() ⟹ from functools import reduce

Syntax: reduce(function, iterable, initializer (optional))

Summation of all elements ⟹ reduce(lambda x, y : x+y , [1,2,3,7])

Step 1 : 1+2 = 3

Step 2 : 3+3 = 6

Step 3 : 6+4 = 10

reduce(f , [a,b,c,d])

Step 1 = res1 = f(a,b)

Step 2 = res2 = f(res1, c)

Step 3 = res3 = f(res2, d)

res3

reduce(lambda x,y : x+y , [1,2,3,4] , 5)

Step1 : 5 + 1 = 6

Step2 : 6 + 2 = 8

Step3 : 8 + 3 = 11

Step4 : 11 + 4 = 15

l = reduce(lambda x,y : x if x>y else y , [2,9,3,10,4,7])

print(l)

10

Step1 : x=2 , y=9 ⇒ 9

Step2 : x=9 , y=3 ⇒ 9

Step3 : x=9 , y=10 , ⇒ 10

Step4 : x=10 , y=4 ⇒ 10

Step5 : x=10 , y=7

⇒ 10

a = reduce(lambda x,y : x+y if y%2==0 else x, [1,2,3,4,5],0)

Step 1: x=0, y=1 ⇒ 0

Step 2: x=0, y=2 ⇒ 2

Step 3: x=2, y=3 ⇒ 2

Step 4: x=2, y=4 ⇒ 6

Step 5: x=6, y=5 ⇒ 6

```
a = reduce(lambda x,y : x +"-"+ y , ["RAVI", "ISHAN", "ADITYA"])
```

RAVI - ISHAN - ADITYA

```
b = reduce( lambda x,y: x-y , [1,2,3,4] )
print(b)
```

Steps: 1 - 2 = -1
-1 - 3 = -4
-4 - 4 = -8

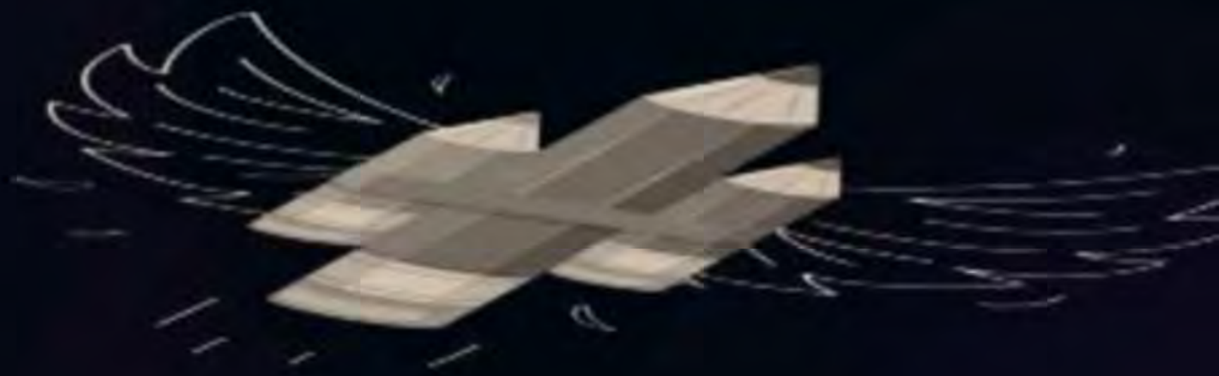
```
c = list(map( lambda x : x[0]*x[1] , [(2,3), (5,6)] ))
print(c)
```

(2,3) (5,6)
x[0] x[1]

[6, 30]

Summary

- lambda
- map
- filter
- reduce



THANK - YOU