**LAMBDAS / DECORATORS / GENERATORS**

**LAMBDAS:**

A Lambdas is an anonymous function that will not have any name when we define a function in python do it using the def keyword followed by the name of the function then the argument list followed by body of the function within which will have multiple statements.

The lambda expression will return a function that will calculate the square of given x (Output below).

**Using Function:**

def square(x):

return x\*x

**Using Lambda:**

Syntax: lambda argument\_list:expression

Output: func = lambda x:x\*x

Lambda expression are helpful when we can use it like this by assigning it to a function and invoking it but lambda expressions are very helpful when we use them inside other functions.

**Code:**

#REGULAR

print(*"========== Using Regular Function ============"*)

def **cube**(n):

return n\*\*3

print(*"Regular Function : Cube Value : "*, cube(2))

#USING LAMBDA

print(*"========== Using Lambda Function ============"*)

fcube = lambda n:n\*\*3

print(*"Lambda Function : Cube Value : "*, cube(2))

**Output:**

========== Using Regular Function ============

Regular Function : Cube Value : 8

========== Using Lambda Function ============

Lambda Function : Cube Value : 8

**Using Lambda: Odd or Even**

**Code:**

bValue=lambda x: *'YES'* if (x%2 == 0) else *'NO'*

print(bValue(7))

print(bValue(12))

**Output:**

NO

YES

**Using Lambda: Odd or Even**

**Code:**

#REGULAR

print(*"========== Using Regular Function ============"*)

def **add**(a,b):

return a+b

print (add(100,200))

#USING LAMBDA

print(*"========== Using Lambda Function ============"*)

sumnos = lambda a,b:a+b

print(sumnos(125,175))

**Output:**

========== Using Regular Function ============

300

========== Using Lambda Function ============

300

**Using Filter retrieve an even numbers from a list**

**Code:**

lst=[12,31,51,72,62,85,90,76,21,102,25]

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

print(result)

**Output:**

[24, 62, 102, 144, 124, 170, 180, 152, 42, 204, 50]

**Using Map double the numbers from a list**

**Code:**

lst=[12,31,51,72,62,85,90,76,21,102,25]

result = list(map(lambda n:n\*2, lst))

print(result)

**Output:**

[24, 62, 102, 144, 124, 170, 180, 152, 42, 204, 50]

**Using Reduce function to find out the sum of all elements in a list**

**Code:**

from functools import \*

lst=[12,31,51,72,62,85,90,76,21,102,25]

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

print(result)

**Output:**

627

**DECORATORS**

Def itself says it’s a function that performs additional logic on a given function.

It also returns function back as the result.

**Eg:** def mydefFunc(func)

So a decorator takes a function and it will return a function but the function that is returned by the decorator will perform additional logic or additional processing on the function that is given as input.

**Using Decorator:**

**Code:**

def **decorfunction**(outfunciton):

def **innerfunction**():

result=outfunciton()

return result\*2

return innerfunction

#USING DECORATOR

def **numfunction**():

val = int(input(*"Enter the number to be doubled : "*))

return val

resultOutFunction = decor(numfunction)

print(resultOutFunction())

**Output:**

Enter the number to be doubled : 25

50

**Using Decorator @:**

**Code:**

def **decorfunction**(outfunciton):

def **innerfunction**():

result=outfunciton()

return result\*2

return innerfunction

#USING DECORATOR

def **numfunction**():

val = int(input(*"Enter the number to be doubled : "*))

return val

#USING @ IN DECORATOR

*@* **decorfunction**

def **numfunction1**():

val1=125 #VALUE PASSED INSTEAD OF READING AS INPUT

return val1

print(*"OUTPUT USING DECOR FUNCTION ONLY ============="*)

resultOutFunction = **decorfunction**(numfunction)

print(resultOutFunction())

print(*"OUTPUT USING DECOR AND @ FUNCTION ONLY ============="*)

print(numfunction1())

**Output:**

OUTPUT USING DECOR FUNCTION ONLY =============

Enter the number to be doubled : 25

50

OUTPUT USING DECOR AND @ FUNCTION ONLY =============

250

**Using String in Decorator and @:**

**Code:**

def **howareyoufun**(anotherfunction):

def **innerfun**(nameval):

result = anotherfunction(nameval)

result += *" !!!! How are you feeling now? "*

return result

return innerfun

*@howareyoufun*

def **hellofunciton**(name):

return *"Hello "*+name

stval = input(*"Enter the name : "*)

print(hellofunciton(stval))

**Output:**

Enter the name : Jazy

Hello Jazy !!!! How are you feeling now?

**Using Decorator Chain**

**Code:**

#DECORATOR CHAINING

def **square**(func1):

def **sqinnerfunc**():

inval1 = func1()

return inval1\*inval1

return sqinnerfunc

#DECORATOR

def **half**(func):

def **innerfunc**():

fullval = func()

return fullval/2

return innerfunc

*@square*

*@half*

def **initialfunc**():

inval = int(input(*"Enter the input value : "*))

return inval

print(initialfunc())

**Output:**

Enter the input value : 25

156.25

**IF THE ORDER IS CHANGED**

**Code:**

*@half*

*@square*

def **initialfunc**():

inval = int(input(*"Enter the input value : "*))

return inval

print(initialfunc())

**Output:**

Enter the input value : 25

312.5

**GENERATORS**

Generators are functions that return a sequence of values back.

A generated function is returned just like any other function but it uses a yield statement.

As we generate our sequence our custom sequence using the yield will be storing each value in that sequence and at the end of the function or at the end of the sequence generation, we will return the entire sequence back.

This is similar to range datatype, which is used in for loops etc;

**Using Custom Generator**

**Code:**

def **customgenerator**(x,y):

while x<y:

yield x

x+=1

result = customgenerator(20,30)

for i in result:

print(i)

**Output:**

20

21

22

23

24

25

26

27

28

29

**Program to find the key words of Python**

**Code:**

import keyword

#PROGRAM TO FIND THE KEYWORDS OF PYTHON

lst = list(keyword.kwlist)

for i in lst:

print(i)

**Output:**

False

None

True

and

as

assert

async

await

break

class

continue

def

del

elif

else

except

finally

for

from

global

if

import

in

is

lambda

nonlocal

not

or

pass

raise

return

try

while

with

yield

**QUIZ**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated