**Anonymous Function (Lambda Expressions):**

If a function does not have any name, then it is called as anonymous function.

If the function contains only one statement and if the function is returning a value then, we can define a anonymous function. To create an anonymous function, we use lambda expression.

**Syntax of lambda expression:**

lambda paramaterlist : expression

In the lambda expression, we can specify any number of parameters and the lambda expressions will return a value.

**Program:** to calculate the double of a number with and without lambda expression.

def double(n):

return n\*2

res = double(5)

print(res)

with lambda expression:

res = lambda n : n\*2

print(res(50))

Program: to calculate the square of a number with and without lambda expression.

def square(x):

return x\*x

res = square(5)

print(res)

with lambda expression:

res = lambda x : x\*x

print(res(6))

Program: to calculate the sum of two numbers with and without lambda expression.

def add(a,b):

return a+b

res = add(4,5)

print(res)

with lambda expression:

res = lambda a,b : a+b

print(res(5,6))

Program: to calculate the value of an expression with and without lambda expression.

def calculate(a,b):

return a\*\*2+b\*b+2\*a\*b

res = calculate(2,3)

print(res)

with lambda expression:

res = lambda a,b : a\*\*2+b\*b+2\*a\*b

print(res(2,3))

Program: to display a message with and without lambda expression

def msg():

return 'hello'

print(msg())

with lambda expression:

res = lambda : 'hello'

print(res())

**Types of Variables**

Python supports 2 types of variables.

1. Global Variables

2. Local Variables

1. Global Variables

The variables which are declared outside of function are called global variables.

These variables can be accessed in all functions of that module.

Eg:

a=10 # global variable

def f1():

a = 30 # local variable

b = 100

print(a)

def f2():

print(a)

f1()

f2()

Output

10

10

2. Local Variables:

The variables which are declared inside a function are called local variables.

Local variables are available only for the function in which we declared it. i.e from outside of function we cannot access.

Eg:

def f1():

a=10

print(a) # valid

def f2():

print(a) #invalid

f1()

f2()

NameError: name 'a' is not defined

**global keyword:**

We can use global keyword for the following 2 purposes:

1. To declare global variable inside function

2. To make global variable available to the function so that we can perform required modifications

Example:

a=10

def f1():

a=777

print(a)

def f2():

print(a)

f1()

f2()

**Example:**

a=10

def f1():

global a

a=777

print(a)

def f2():

print(a)

f1()

f2()

Output

777

777

**Example:**

def f1():

a=10

print(a)

def f2():

print(a)

f1()

f2()

**Note**: If global variable and local variable having the same name then we can access global variable inside a function as follows

a=10 #global variable

def f1():

a=777 #local variable

print(a)

**print(globals()['a'])**

f1()

Output

777

10

**Recursion:**

If a **function is calling itself multiple times**, then it is called as recursion and such function are called as recursive functions.

2 Cases

1. Base Case (terminating from Recursion)
2. Recursive Case (Calling itself)

**Program:** to display factorial using recursion

def factorial(n):

if (n==0 or n==1):

return 1

else:

return n\*factorial(n-1)

n=int(input('enter number: '))

obj=factorial(n)

print(obj)

**Result:**

enter number: 5

120

----------------------

n=5

5! =120

5\*4!

5\*4\*3!

5\*4\*3\*2!

5\*4\*3\*2\*1!

**Program: Fibanocci series by using recursion**

def fibonacci(x):

if x==0:

return 0

elif x==1:

return 1

else:

return fibonacci(x-2)+fibonacci(x-1)

for i in range(10):

print(fibonacci(i))

**Result:**

0

1

1

2

3

5

8

13

21

34

**map():**

The map() can be used for applying some logic on each and every value available in a sequence.

Or

The **map() allows you to 'map' a function to an inerrable object.** That is to say you can quickly call the same function to every item in an inerrable, such as a list.

**Syntax:**

map(functionname, sequence)

functionname --> f

sequence --> v1,v2,v3,...vn

map(f,v1,v2,v3,...,vn) --> f(v1),f(v2),f(v3),...,f(vn)

**Example:** Double of Number (without lambda with map())

a= [1,2,3,4,5,6,7,8,9,10]

def double(n):

return n\*2

res=list(map(double,a))

print(res)

**Example:** Double of Number (with lambda with map())

a= [1,2,3,4,5,6,7,8,9,10]

res=list(map(lambda x:x\*2,a))

print(res)

**Example:** Add Two lists

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

b=[10,20,30,40,50]

res=list(map(lambda x,y:x+y,a,b))

print(res)

**filter():** The filter can be **used for applying some logic on each and every value available in a sequence and returns those values which satisfies a condition.** Filter can filter the unwanted elements.

**Syntax:**

filter(functionname,sequence)

**Example:** display even numbers from 1 to 10

**Normal Scenario:**

for i in range(1,11):

if i%2==0:

print(i)

**By using filter()**

res=list(filter(lambda x:x%2==0,range(1,10)))

print(res)

**filter() with one sequence**

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

res=list(filter(lambda x:x%2==0,a))

print(res)

**reduce():**

Python’s **reduce() is a function that implements a mathematical technique called folding or reduction**. reduce() is useful when you need to apply a function to an iterable and reduce it to a single cumulative value.

The filter() method constructs an iterator from elements of an iterable for which a function returns true.

**Syntax:**

filter(function, iterable)

Note: to execute reduce (), we have to **import functools**

**Example:** sum of list

import functools

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

res=functools.reduce(lambda x,y:x+y,a)

print(res)