**Lambda Functions**

In Python, anonymous functions are called as Lambda functions.

The functions that are created without using name are called anonymous or Lambda functions.

While normal functions are defined using the **def keyword**, in Python anonymous functions are defined using the **lambda keyword**.

**Syntax**

lambda arguments : expression

Lambda functions can have any number of arguments but only one expression. The expression is evaluated and returned.

**Normal Function Example**

>>> def add(x=0,y=0):

return x+y

>>> print(add(10,20))

30

**Lambda Function**

>>> result = lambda x,y : x+y

>>> print(result(10,20))

30

**Lambda Function with default values**

>>> result = lambda x=10,y=20 : x+y

>>> print(result)

<function <lambda> at 0x0000000002FB46A8>

>>> print (result())

Output: 30

>>> print (result(500,600))

Output: 1100

**Lambda Function with named or keyword arguments**

>>> result = lambda x=10,y=20 : x+y

>>> print(result)

<function <lambda> at 0x0000000001D01E18>

>>> print(result(y=-40,x=20))

-20

**Using if else in lambda functions**

result = lambda no1,no2:no1 if no1>no2 else no2

no1 = int(input("Enter 1st No: "))

no2 = int(input("Enter 2nd No: "))

print(result(no1,no2))

**Using lambda function in map**

**Syntax**

**map(function\_object, iterable1, iterable2,...)**

Map functions expects a function object and any number of iterables like list, dictionary, etc.

Map executes the function\_object for each element in the sequence and returns a map of the elements modified by the function object.(Convert into list)

**Example**

l1 = [1,2,3,4]

print(list(map(lambda x: x\*2,l1)))

Output : [2, 4, 6, 8]

**Example**

l1 = [1,2,3,4]

l2 = [5,6,7,8]

print(list(map(lambda x,y: x\*y,l1,l2)))

Output: [5, 12, 21, 32]

**Using filter function in lambda**

**Syntax**

**filter(function\_object, iterable)**

*filter*function expects two arguments, function\_object and an iterable. function\_object returns a boolean value.

function\_object is called for each element of the iterable and filter returns only those element for which the function\_object returns *true*.

Like *map*function, *filter*function also returns a map of element. Unlike *map*function *filter*function can only have one iterable as input.

**Example**

a = [1, 2, 3, 4, 5, 6]

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

Output: [2, 4, 6]

**Using reduce function in lambda**

The reduce() function takes in a function and a list as argument. The function is called with a lambda function and a list and a new reduced result is returned. This performs a repetitive operation over the pairs of the list. **This is a part of functools module.**

**Example**

from functools import reduce

li = [5, 8, 10, 20, 50, 100,150,200]

sum = reduce((lambda x, y: x + y), li)

print (sum)