**Lambda functions or lambda expression**

**What is lambda function?**
Lambda function is anonymous function. A function which does not have any name is called anonymous function.
Lambda function is created using “lambda” keyword.

Lambda functions are used as higher order functions. A function which receives input as another function is called higher order function.

Lambda functions or expression is called single line function. It is not with return statement.

**Syntax:**
lambda [parameters]:statement

A lambda function can be defined,

1. With parameters
2. Without parameters

**Example:**

```
a=lambda:print("Hello Lambda") # defining lambda
function/expression
# a is name of lambda function/expression

a()
a()
a()
a()
```

**Output:**

```
Hello Lambda
Hello Lambda
Hello Lambda
Hello Lambda
```

**Example:**

```
sqr=lambda num:num**2  # lambda function with parameters

res1=sqr(4)
res2=sqr(6)

print(res1)
print(res2)
```

**Output:**

```
16
36
```

**filter(function, iterable)**

Construct an iterator from those elements of iterable for which function is true. iterable may be either a sequence, a container which supports iteration, or an iterator. If function is None, the identity function is assumed, that is, all elements of iterable that are false are removed.

**Example:**

```
list1=[1,2,3,4,5,6,7,8,9,10]
```

```
a=filter(lambda num:num%2==0,list1)
for value in a:
    print(value,end=' ')

b=filter(lambda num:num%2!=0,list1)
print()
for value in b:
    print(value,end=' ')
```

**Output:**

```
2 4 6 8 10
1 3 5 7 9
```

**Example:**

```
def calc(n1,n2,func):
    n3=func(n1,n2)
    return n3



res1=calc(10,20,lambda a,b:a+b)
res2=calc(10,5,lambda a,b:a-b)
res3=calc(5,2,lambda a,b:a*b)
print(res1,res2,res3)
```

**Output:**

```
30 5 10
```

**Example:**

```
grade_list=[['naresh','A'],
            ['suresh','A'],
            ['ramesh','B'],
            ['kishore','B'],
            ['rajesh','C']]

print(grade_list)
grade_a=list(filter(lambda stud:stud[1]=='A',grade_list))
print(grade_a)
grade_b=list(filter(lambda stud:stud[1]=='B',grade_list))
print(grade_b)
```

**Output:**

```
[['naresh', 'A'], ['suresh', 'A'], ['ramesh', 'B'], ['kishore', 'B'], ['rajesh', 'C']]
[['naresh', 'A'], ['suresh', 'A']]
[['ramesh', 'B'], ['kishore', 'B']]
```

**map(function, iterable, *iterables)**

Return an iterator that applies function to every item of iterable, yielding the results. If additional iterables arguments are passed, function must take that many arguments and is applied to the items from all iterables in parallel. With multiple iterables, the iterator stops when the shortest iterable is exhausted

**Example:**

```
list1=["10","20","30","40","50"]
list2=list(map(int,list1))
print(list1)
print(list2)
```

**Output:**

```
['10', '20', '30', '40', '50']
[10, 20, 30, 40, 50]
```

**Example:**

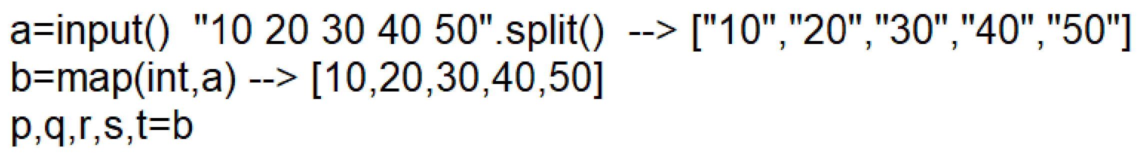
```
list1=[1,2,3,4,5]
list2=[10,20,30,40,50]

list3=list(map(lambda x,y:x+y,list1,list2))

print(list1)
print(list2)
print(list3)
```

**Output:**

```
[1, 2, 3, 4, 5]
[10, 20, 30, 40, 50]
[11, 22, 33, 44, 55]
```



```
a=input()  "10 20 30 40 50".split()  --> ["10","20","30","40","50"]
b=map(int,a) --> [10,20,30,40,50]
p,q,r,s,t=b
```

**Example**

```
>>> names=["naresh","ramesh","suresh","kishore"]
>>> print(names)
['naresh', 'ramesh', 'suresh', 'kishore']
>>> names1=list(map(lambda name:name.upper(),names))
>>> print(names1)
['NARESH', 'RAMESH', 'SURESH', 'KISHORE']
```

**functools.reduce(function, iterable[, initializer])**
Apply function of two arguments cumulatively to the items of iterable, from left to right, so as to reduce the iterable to a single value.
The left argument, x, is the accumulated value and the right argument, *y*, is the update value from the *iterable*.

**Example:**

```
import functools
list1=[10,20,30,40,50]
res1=functools.reduce(lambda x,y:x+y,list1)
print(list1)
print(res1)
```

**Output:**

```
[10, 20, 30, 40, 50]
150
```

**Recursive Functions or Function Recursion**