Lambda functions or lambda expressions or Anonymous functions

Lambda function is anonymous function.
A function which does not have name is called lambda function or anonymous function.
Lambda functions are used with higher order functions.
A function which receives input as another function is called higher order function.
Lambda functions are inline function or single line function, which are having only one statement.

Syntax:
lambda <parameters>:statement

Lambda function is defined,

1. With parameters
2. Without parameters

Example:

```
a=lambda:print("Hello Lambda,...")
a()
a()
a()
a()
add=lambda a,b:a+b
res=add(10,20)
print(res)
```

Output

```
Hello Lambda,...
Hello Lambda,...
Hello Lambda,...
Hello Lambda,...
30
```

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=[1,6,8,9,3,5,11,25,29,9,12,35,67,97,53,35,76,63]

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

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

Output

```
1 9 3 5 11 25 29 9 35 67 97 53 35 63
6 8 12 76
```

Example

```
names=['naresh','kishore','raman','kiran','rajesh','suresh']

a=filter(lambda name:name[-1]=='h',names)
for value in a:
    print(value,end=' ')
```

Output

```
naresh rajesh suresh
```

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

list3=list(map(lambda a,b:a+b,list1,list2))
print(list3)

list4=list(map(lambda a:str(a),list1))
print(list4)

list5=["10","20","30","40","50"]
list6=list(map(lambda a:int(a),list5))
print(list5)
print(list6)
```

Output

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

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. For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5). The left argument, x, is the accumulated value and the right argument, y, is the update value from the iterable. If the optional initializer is present, it is placed before the items of the iterable in the calculation, and serves as a default when the iterable is empty. If initializer is not given and iterable contains only one item, the first item is returned.

Example:

```
import functools

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

```
res1=functools.reduce(lambda a,b:a+b,list1)
print(res1)
res2=functools.reduce(lambda a,b:a if a>b else b,list1)
res3=functools.reduce(lambda a,b:a if a<b else b,list1)
print(res2,res3)
```

Output

```
55
10 1
```

What is difference between normal function and lambda function?

Normal function(def function)	**Lambda function**
A normal function is defined with keyword “def”.	A lambda function is defined using keyword “lambda”
It is with name	It is without name
It required return statement to return value	It does not required return statement to return value
It contains one or more than one statement	It contains only one statement
It is not efficient for function having one statement	It is efficient for function having one statement

Function Recursion

Function recursion is calling function itself.
In recursion, calling function and called function both are same.
Python also accepts function recursion, which means a defined function can call itself. Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

Syntax:

```
def <function-name>([parameters]):
        statement-1
        statement-2
        function-name() # Recursive call
```