



**TATA ELXSI**

## Python Scripting

Learning & Development Team

Monday, November 2, 2020

## Module – 4 : Functional Programming

## Disclaimer

- This material is developed for in house training at **TATA ELXSI**.
- *TATA ELXSI is not responsible for any errors or omissions, or for the results obtained from the use of this information. All information in this material is provided "as is", with no guarantee of completeness, timeliness or of the results obtained from the use of this information...*
- *This is a training material and the information contained herein is not intended to be a source of advice or research outcome with respect to the material presented, and the information and/or documents/examples contained in this material do not constitute authors or TATA ELXSI's advice.*
- The training material therefore was solely developed for educational purposes. Currently, commercialization of the prototype does not exist, nor is the prototype available for general usage.
- **TATA ELXSI**, its staff, its students or any other participants can use it as a part of training. This material should not be found on any website.
- For the preparation of this material we have referred from the below mentioned links or books.
- Excerpts of these material has been taken where there is no copy right infringements.

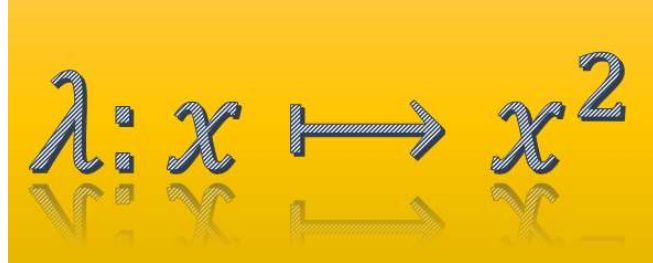
## Agenda

- Lambda Operator
- Map
- Filter
- Reduce

# Functional programming

- Functional programming is all about expressions.
- We may say that the Functional programming is an expression oriented programming.
- Expression oriented functions of Python provides are:
  - lambda
  - map(aFunction, aSequence)
  - filter(aFunction, aSequence)
  - reduce(aFunction, aSequence)

# Lambda Operator

A yellow rectangular box containing the mathematical expression  $\lambda: x \mapsto x^2$  in a stylized, metallic font. The expression represents a lambda function that takes an argument  $x$  and returns its square  $x^2$ .

- The lambda operator or lambda function is a way to create small anonymous functions.
- These functions are throw-away functions.
- Lambda functions can be used wherever function objects are required
- The general syntax of a lambda function is :

`lambda argument_list: expression`

- The argument list consists of a comma separated list of arguments and the expression is an arithmetic expression using these arguments.

## Lambda Operator

- The following example of a lambda function returns the sum of its two arguments:

```
>>> f = lambda x, y : x + y
>>> f(1,1)
2
```

- Like nested function definitions, lambda functions can reference variables from the containing scope:

```
>>> def make_incrementor(n):
    return lambda x: x + n
```

```
>>> f = make_incrementor(42)
>>> f(2)
44
>>>
```

## Built-in functions

- There are three built-in functions that are very useful when used with lists: `filter()`, `map()`, and `reduce()`.
- `map(function, sequence)` calls `function(item)` for each of the sequence's items and returns a list of the return values.
- `filter(function, sequence)` returns a sequence consisting of those items from the sequence for which `function(item)` is true.
- `reduce(function, sequence)` returns a single value constructed by calling the binary function “function” on the first two items of the sequence, then on the result and the next item, and so on.



## The map() Function

- The advantage of the lambda operator can be seen when it is used in combination with the map() function.
- map() is a function with two arguments:
- `r = map(func, seq)`
- The first argument func is the name of a function and the second a sequence (e.g. a list) seq.

## Example : map function

```
def fahrenheit(T):  
    return ((float(9)/5)*T + 32)
```

```
def celsius(T):  
    return (float(5)/9)*(T-32)
```

```
temp = (36.5, 37, 37.5, 39)
```

```
F = map(fahrenheit, temp)
```

```
C = map(celsius, temp)
```

```
for i in list(F):
```

```
    print(i)
```

```
print('-----')
```

```
for i in list(C):
```

```
    print(i)
```

## Map and lambda function

```
>>> Celsius = [39.2, 36.5, 37.3, 37.8]
```

```
>>> Fahrenheit = map(lambda x: (float(9)/5)*x + 32, Celsius)
```

```
>>> print (list( Fahrenheit ))
```

```
[102.56, 97.700000000000003, 99.140000000000001, 100.03999999999999]
```

```
>>> C = map(lambda x: (float(5)/9)*(x-32), Fahrenheit)
```

```
>>> print (list(C))
```

```
[39.200000000000003, 36.5, 37.300000000000004, 37.799999999999997]
```

```
>>>
```

## Map and lambda function

- `map()` can be applied to more than one list.
- The lists have to have the same length.
- `map()` will apply its lambda function to the elements of the argument lists,

```
>>> a = [1,2,3,4]
>>> b = [17,12,11,10]
>>> c = [-1,-4,5,9]
>>> list(map(lambda x,y:x+y, a,b))
[18, 14, 14, 14]
```

```
>>>list( map(lambda x,y,z:x+y+z, a,b,c))
[17, 10, 19, 23]
```

```
>>> list(map(lambda x,y,z:x+y-z, a,b,c))
[19, 18, 9, 5]
```

## filtering

- The function `filter(function, list)` offers an elegant way to filter out all the elements of a list, for which the function ***“function”*** returns True.
- The function `filter( f , l )` needs a function `f` as its first argument.
- `f` returns a Boolean value,
- i.e. either True or False.
- This function will be applied to every element of the list `l`.
- Only if `f` returns True will the element of the list be included in the result list.

## Examples: filter

### Example 1:

```
>>> def f(x): return x % 3 == 0 or x % 5 == 0
...
>>> filter(f, range(2, 25))
[3, 5, 6, 9, 10, 12, 15, 18, 20, 21, 24]
```

### Example 2:

```
>>> list(range(-5,5))
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
>>>
>>> list( filter((lambda x: x < 0), range(-5,5)))
[-5, -4, -3, -2, -1]
>>>
```

## Examples: filter

### Example 3:

```
>>> fib = [0,1,1,2,3,5,8,13,21,34,55]
```

```
>>> result = filter(lambda x: x % 2, fib)
```

```
>>> print result
```

```
[1, 1, 3, 5, 13, 21, 55]
```

```
>>> result = filter(lambda x: x % 2 == 0, fib)
```

```
>>> print result
```

```
[0, 2, 8, 34]
```

```
>>>
```

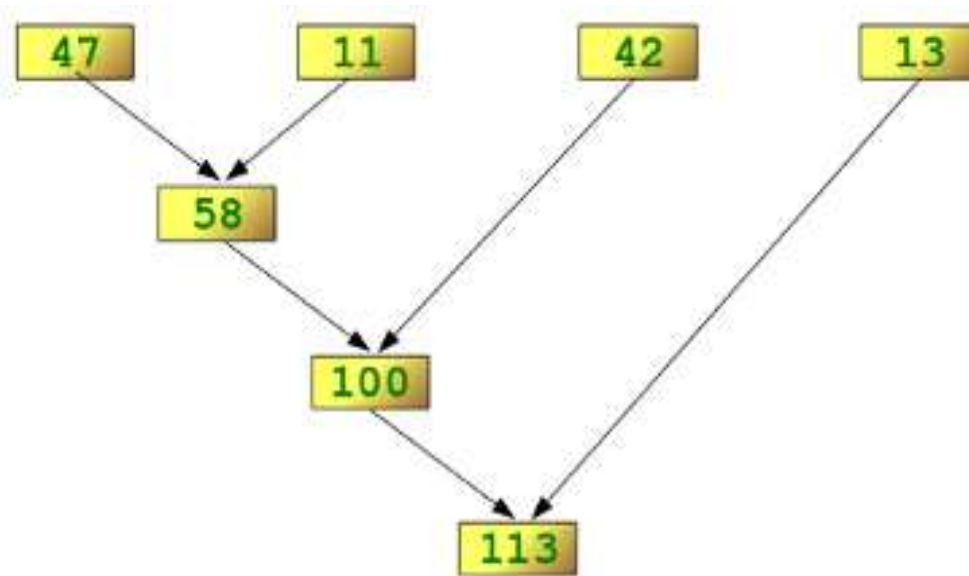
## Reduce: function

- `reduce(function, sequence)` : returns a single value constructed by calling the binary function “function” on the first two items of the sequence, then on the result and the next item, and so on.
- If `seq = [ s1, s2, s3, ... , sn ]`, calling `reduce(func, seq)` works like this:
  - At first the first two elements of `seq` will be applied to `func`, i.e. `func(s1,s2)` The list on which `reduce()` works looks now like this: `[ func(s1, s2), s3, ... , sn ]`
  - In the next step `func` will be applied on the previous result and the third element of the list, i.e. `func(func(s1, s2),s3)`
  - The list looks like this now: `[ func(func(s1, s2),s3), ... , sn ]`
- Continue like this until just one element is left and return this element as the result of `reduce()`



## Reduce: function

- We illustrate this process in the following example:
  - `>>> reduce(lambda x,y: x+y, [47,11,42,13])`
  - 113



## Examples: reduce

- Example 1:

For example, to compute the sum of the numbers 1 through 10:

```
>>> def add(x,y): return x+y
```

```
...
```

```
>>> reduce(add, range(1, 11))
```

```
55
```

- Example 2:

- Determining the maximum of a list of numerical values by using reduce:

```
>>> f = lambda a,b: a if (a > b) else b
```

```
>>> reduce(f, [47,11,42,102,13])
```

```
102
```

```
>>>
```

## Examples

```
>>> list(range(-5,5))
```

```
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
```

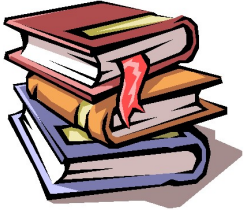
```
>>>
```

```
>>> list( filter((lambda x: x < 0), range(-5,5)))
```

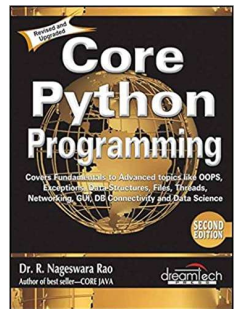
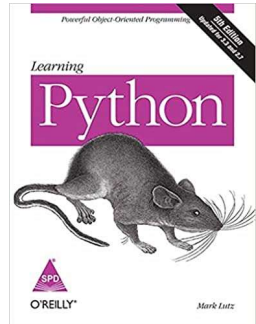
```
[-5, -4, -3, -2, -1]
```

```
>>>
```

## References



- Python 3.x.x documentation: <https://docs.python.org/3/>
- Learning Python: Powerful Object-Oriented Programming: 5th Edition
- Core Python Programming [by [R. Nageswara Rao](#) (Author)]



# Thank you

Learning & Development

Tata Elxsi

Bangalore

[www.tataelxsi.com](http://www.tataelxsi.com)

---

**Confidentiality Notice**

This document and all information contained herein is the sole property of Tata Elxsi Limited and shall not be reproduced or disclosed to a third party without the express written consent of Tata Elxsi Limited.

---

**TATA ELXSI**