FUNCTIONS +

THE ANONYMOUS FUNCTION

- Python supports the creation of anonymous functions.
- Anonymous functions are not bound to a name at runtime, they are created using a construct called "**lambda**".
- The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name.
- These functions are throwaway functions, i.e. they are just needed where they have been created.
- Lambda functions are mainly used in combination with the functions filter(), map() and reduce().

The general syntax of a lambda function is as follows:

LAMBDA ARGUMENT_LIST **:** RETURN EXPRESSION

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

```
>>> lambda  x : x * 2
<function <lambda> at 0x03716C90>
>>>
```

The lambda creates an anonymous function

```
>>>
>>> for i in range(10):

        x=lambda arg : arg * 2

        print(x(i))

0
2
4
6
8
10
12
14
16
18
```

THE MAP, FILTER, AND REDUCE FUNCTION

THE MAP FUNCTION

The map() function applies a function to every member of an iterable (e.g.,a list) and returns the result.

```
>>> #let's first demonstrate a map() with a user-definied function
>>> #define a function which takes an input and return the squares
>>> def square(x):
                return x**2
>>> #list of numbers
>>> numbers =[1,2,3,4,5,6,7,8,9,10]
>>> #use map() and pass the square() and the list
>>> results = list(map(square, numbers))
>>> print (results)
```

The output of the above program is as follows:

```
 [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In the above program we use a list to output the result of the map object that applies the square function to every element of the iterable 'numbers'.-

results = list(map(square, numbers)).

If we compile without list() we will get the output shown below:

```
<map object at 0x02E75C70>
```

In the below program we will use a **lamda** function with the map() function.

```
>>> #list of numbers
>>> numbers =[1,2,3,4,5,6,7,8,9,10]
>>> #use map() and use lamda
>>> results = map(lambda x: x**2, numbers)
>>> print(list(results))
```

The output of the above program is as follows:

```
 [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

THE FILTER FUNCTION

Filter extracts each element in the sequence for which the function returns **True**. Filter takes a function returning True or False and applies it to a sequence, ***returning a list of only those members of the sequence for which the function returned True***.

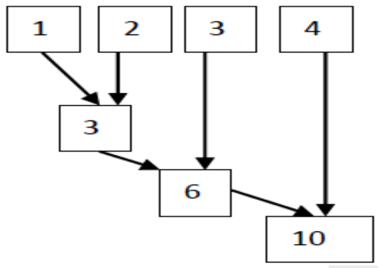
```
>>> numbers =[1,2,3,4,5,6,7,8,9,10]
>>> #use map() with lamda
>>> results =list(map(lambda x: x**2, numbers))
>>> partial_squares = list(filter(lambda x: x > 5 and x < 50, results))
>>> print(results)
>>> print(partial _squares)
```

The output of the above program is shown below:

```
 [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[9, 16, 25, 36, 49]
```

THE REDUCE FUNCTION

The reduce function continually applies the function to the sequence and returns a single value. For example if we need to add a sequence of numbers say 1,2,3,4. It would look like the below diagram:



The reduce function is in the **functools** in Python 3.x.x, so we have imported it in the example.

```
>>> from functools import reduce
>>> result = reduce( (lambda x, y: x + y), [1, 2, 3, 4] )
>>> print(result)
```

The output of the above program is as follows:

```
10
```

At each step, reduce passes the current addition, along with the next item from the list, to the passed-in lambda function. By default, the first item in the sequence initialized the starting value.