1. def is used for defining functions where we define a function by a specific name of function whereas lambda keyword is used to define function without a name. def takes multiple lines to define a complete function whereas lambda defines whole function in 1 single line.

2.Benefit of lambda compared to def is that it can define a function within 1 single line without any explicit name of function unlike def which requires a name of function to be defined and takes multiple lines of code.

3.

* map takes 2 arguements, 1st is the function/method with which we want to map each and every value inside any iterable(list/tuple etc.), and 2nd arguement is the iterable(list/tuple etc.). Makes an iterator that computes the function using arguments from each of the iterables. Stops when the shortest iterable is exhausted. map() maps each and every value of iterable(list/tuple etc.) to the function passed argument.
* filter()- to filter out some specific values out of an iterable as per some condition, and it returns only those values which satisfy True. Only true outcomes returned. Return an iterator yielding those items of iterable for which function(item) is true. If function is None, return the items that are true. filter returns only when True and map returns all elements always.
* Apply a function of two arguments cumulatively to the items of a sequence

from left to right, so as to reduce the sequence to a single value

For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates

((((1+2)+3)+4)+5). If initial is present, it is placed before the items

of the sequence in the calculation, and serves as a default when the

sequence is empty.

4. Function annotations introduced in Python 3.0 adds a feature that allows you to add arbitrary metadata to function parameters and return value. Since python 3, function annotations have been officially added to python (PEP-3107). The primary purpose was to have a standard way to link metadata to function parameters and return value.

5. Recursive functions are the functions which call themselves, in recursive functions, we assign some 1 or 2 base values for initialization of possible results based on which the function performs similar operations later on.

One of the best egs. of recursive functions is Fibonacci series:

**def** fibonacci(n):

**if**(n <= 1):

**return** n

**else**:

**return**(fibonacci(n-1) + fibonacci(n-2))

base value is n=0 or n=1 for which value returned is n.

6. There are some guidelined for modular coding for functions in python and some of them are:

* Excpetion Handling: Keep handling exceptions using try except after every small operation in a code to handle run time exceptions which may occur to make the code resistant to exceptions and our project is always up and running and execution is not stopped.
* OOPS: Always write code in the form of classes, methods and objects as a part of modular industrial standard of coding.
* Logging: Never use print() in an industrial level code, instead use logging after every operation in a code which helps us keep a track of all events happening or any error occurred

7. Functions can communicate results via print() or return values, print though prints the values inside console but datatype remains NoneType whereas return returns exact datatype. We can call function explicitly to get output or assign it to some variable and then printing it.