1. What is the relationship between def statements and lambda expressions ?

Both lambda and def create the same kind of function – they have the same kind of metadata and capabilities. Their technical difference is syntactical:

A lambda is an expression producing a function.

A def is a statement producing a function.

This is everything that dictates how they can be used. Other apparent differences simply come from the information lambda/def can capture.

A lambda is more flexible as expressions can be part of more language constructs.

In contrast, a def is more powerful as it can consist of more language constructs.

These differences derive directly from one being an expression and the other being a statement. Python has no special rules to decide where a lambda/def may be used.

2. What is the benefit of lambda?

* Fewer Lines of Code − One of the most benefits of a lambda expression is to reduce the amount of code. We know that lambda expressions can be used only with a functional interface. For instance, Runnable is a functional interface, so we can easily apply lambda expressions.
* Sequential and Parallel execution support by passing behavior as an argument in methods − By using Stream API in Java 8, the functions are passed to collection methods. Now it is the responsibility of collection for processing the elements either in a sequential or parallel manner.
* Higher Efficiency − By using Stream API and lambda expressions, we can achieve higher efficiency (parallel execution) in case of bulk operations on collections. Also, lambda expression helps in achieving the internal iteration of collections rather than external iteration.

1. Compare and contrast map, filter, and reduce.

The map() Function:

The map() function iterates through all items in the given iterable and executes the function we passed as an argument on each of them.

Similar to map(), filter() takes a function object and an iterable and creates a new list.

The filter() Function:

As the name suggests, filter() forms a new list that contains only elements that satisfy a certain condition, i.e. the function we passed returns True.

The reduce() Function:

reduce() works differently than map() and filter(). It does not return a new list based on the function and iterable we've passed. Instead, it returns a single value. reduce() works by calling the function we passed for the first two items in the sequence. The result returned by the function is used in another call to function alongside with the next (third in this case), element.

4. What are function annotations, and how are they used?

Function annotations introduced in Python 3.0 adds a feature that allows you to add arbitrary metadata to function parameters and return value.

* Function annotations are completely optional both for parameters and return value.
* Function annotations provide a way of associating various parts of a function with arbitrary python expressions at compile time.
* The PEP-3107 makes no attempt to introduce any kind of standard semantics, even for the built-in types. All this work left to the third-party libraries
* def foo(x: expression, y: expression = 20):

5. What are recursive functions, and how are they used?

A recursive function is a [function](https://techterms.com/definition/function) that calls itself during its execution. The process may repeat several times, outputting the result and the end of each [iteration](https://techterms.com/definition/iteration).

function Count (integer N):

if (N <= 0) return "Must be a Positive Integer";

if (N > 9) return "Counting Completed";

else return Count (N+1);

end function

6. What are some general design guidelines for coding functions?

1. Limited use of globals:  
   These rules tell about which types of data that can be declared global and the data that can’t be.
2. Standard headers for different modules:  
   For better understanding and maintenance of the code, the header of different modules should follow some standard format and information. The header format must contain below things that is being used in various companies:
   * Name of the module
   * Date of module creation
   * Author of the module
   * Modification history
   * Synopsis of the module about what the module does
   * Different functions supported in the module along with their input output parameters
   * Global variables accessed or modified by the module
3. Naming conventions for local variables, global variables, constants and functions:  
   Some of the naming conventions are given below:
   * Meaningful and understandable variables name helps anyone to understand the reason of using it.
   * Local variables should be named using camel case lettering starting with small letter (e.g. localData) whereas Global variables names should start with a capital letter (e.g. GlobalData). Constant names should be formed using capital letters only (e.g. CONSDATA).
   * It is better to avoid the use of digits in variable names.
   * The names of the function should be written in camel case starting with small letters.
   * The name of the function must describe the reason of using the function clearly and briefly.
4. Indentation:  
   Proper indentation is very important to increase the readability of the code. For making the code readable, programmers should use White spaces properly. Some of the spacing conventions are given below:
   * There must be a space after giving a comma between two function arguments.
   * Each nested block should be properly indented and spaced.
   * Proper Indentation should be there at the beginning and at the end of each block in the program.
   * All braces should start from a new line and the code following the end of braces also start from a new line.
5. Error return values and exception handling conventions:  
   All functions that encountering an error condition should either return a 0 or 1 for simplifying the debugging.

On the other hand, Coding guidelines give some general suggestions regarding the coding style that to be followed for the betterment of understandability and readability of the code. Some of the coding guidelines are given below :

1. Avoid using a coding style that is too difficult to understand:  
   Code should be easily understandable. The complex code makes maintenance and debugging difficult and expensive.
2. Avoid using an identifier for multiple purposes:  
   Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.
3. Code should be well documented:  
   The code should be properly commented for understanding easily. Comments regarding the statements increase the understandability of the code.
4. Length of functions should not be very large:  
   Lengthy functions are very difficult to understand. That’s why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.
5. Try not to use GOTO statement:  
   GOTO statement makes the program unstructured, thus it reduces the understandability of the program and also debugging becomes difficult.

7. Name three or more ways that functions can communicate results to a caller.

The return value of a Python function can be any Python object. Everything in Python is an object. So, your functions can return numeric values ([int](https://docs.python.org/3/library/functions.html" \l "int), [float](https://docs.python.org/3/library/functions.html#float), and [complex](https://docs.python.org/3/library/functions.html#complex) values), collections and sequences of objects ([list, tuple](https://realpython.com/courses/lists-tuples-python/), [dictionary](https://realpython.com/courses/lists-tuples-python/), or [set](https://realpython.com/courses/sets-python/) objects), user-defined objects, classes, functions, and even [modules or packages](https://realpython.com/courses/python-modules-packages/).

An explicit return statement immediately terminates a function execution and sends the return value back to the caller code. To add an explicit return statement to a Python function, you need to use return followed by an optional return value

A Python function will always have a return value. There is no notion of procedure or routine in Python. So, if you don’t explicitly use a return value in a return statement, or if you totally omit the return statement, then Python will implicitly return a default value for you. That default return value will always be None.