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

Both lambda and def create function objects to be called later. Because lambda is an expression, though, it returns a function object instead of assigning it to a name, and it can be used to nest a function definition in places where a def will not work syntactically. A lambda allows for only a single implicit return value expression, though; because it does not support a block of statements, it is not ideal for larger functions.

1. What is the benefit of lambda?

The **lambda** keyword in **Python** provides a shortcut for declaring small anonymous functions. **Lambda** functions behave just like regular functions declared with the def keyword. They can be used whenever function objects are required.

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

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

The syntax is:

map(function, iterable(s))

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

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 syntax is:

filter(function, iterable(s))

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.

Also, in Python 3 reduce() isn't a built-in function anymore, and it can be found in the functools module.

The syntax is:

reduce(function, sequence[, initial])

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

Function annotations are arbitrary python expressions that are associated with various part of functions. These expressions are evaluated at compile time and have no life in python’s runtime environment. Python does not attach any meaning to these annotations. They take life when interpreted by third party libraries, for example, mypy.

1. What are recursive functions, and how are they used?
2. Recursive functions call themselves either directly or indirectly in order to loop. They may be used to traverse arbitrarily shaped structures, but they can also be used for iteration in general (though the latter role is often more simply and efficiently coded with looping statements). Recursion can often be simulated or replaced by code that uses explicit stacks or queues to have more control over traversals.
3. What are some general design guidelines for coding functions?

Functions should generally be small and as self-contained as possible, have a single unified purpose, and communicate with other components through input arguments and return values. They may use mutable arguments to communicate results too if changes are expected, and some types of programs imply other communication mechanisms.

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

. Functions can send back results with return statements, by changing passed-in mutable arguments, and by setting global variables. Globals are generally frowned upon (except for very special cases, like multithreaded programs) because they can make code more difficult to understand and use. return statements are usually best, but changing mutables is fine (and even useful), if expected. Functions may also communicate results with system devices such as files and sockets, but these are beyond our scope here.