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

Solution - The def keyword is used to define normal functions, while the lambda keyword is used to define anonymous functions.

2. **What is the benefit of lambda?**

Solution - Lambda helps you use a function only once, and hence, avoids cluttering up the code with function definitions. In short, Python's lambda keyword lets you define a function in a single line of code and use it immediately.

3. **Compare and contrast map, filter, and reduce.**

Solution - 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.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

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.

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

Solution - Function annotations are some random expressions which are written with the functions, and they are evaluated at compile time. They do not exist at run time, and there is no meaning of these expressions to python. They are used and interpreted by a third party or external python libraries.

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

Solution - A recursive function is a function in code that refers to itself for execution. Recursive functions can be simple or elaborate. They allow for more efficient code writing, for instance, in the listing or compiling of sets of numbers, strings or other variables through a single reiterated process.

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

Solution - 1. Function naming: Use descriptive and concise names for your functions that accurately reflect their purpose. Function names should follow the lowercase_with_underscores naming convention.

2. Function length: Functions should be relatively short and focused on a single task or responsibility. Aim for functions that are no longer than a screenful of code, preferably less.

3. Function arguments: Functions should take arguments in a clear and consistent order. In general, required arguments should come before optional arguments. Use default values for optional arguments to make function calls more concise.

4. Function return values: Functions should return values in a clear and consistent way. If a function does not return a value, it should return `None` to indicate this. Use the `return` statement to terminate a function early if an error occurs or a specific condition is met.

5. Function documentation: Functions should be well-documented using docstrings that describe their purpose, arguments, and return values. Use the `help()` function to display the docstring for a function.

6. Function side effects: Functions should avoid side effects whenever possible. Side effects can make it harder to reason about the behavior of a function and can lead to bugs.

7. Function testing: Functions should be thoroughly tested using automated tests to ensure that they behave correctly under a variety of conditions. Use the `unittest` module or a third-party testing library like `pytest` to write and run tests for your functions.

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

Solution –

1. Return statement: A function can use the `return` statement to send a value back to the caller. The value can be a single value, a tuple of values, or any other data structure.

2. Global variables: A function can also modify global variables that can be accessed by the caller. However, this is generally not recommended as it can lead to unexpected behavior and make the code harder to maintain.

3. Side effects: A function can also communicate results through side effects, such as modifying an object passed as an argument or printing output to the console. However, this approach can make the code harder to reason about and lead to bugs.

4. Yield statement: A function can also use the `yield` statement to return a generator object that can be used to iterate over a sequence of values. This approach is useful when the function needs to generate a large sequence of values that cannot be stored in memory all at once.

5. Exceptions: A function can raise an exception to communicate an error condition to the caller. The caller can then handle the exception and take appropriate action, such as retrying the operation or displaying an error message to the user