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

Ans. Both def statements and lambda expressions are used to define functions in Python, but they have some key differences:

def statements: They are used to create named functions with a block of code inside the function body. You define a function using the def keyword followed by the function name, parameters (if any), and a colon. The function body is indented, and the function can have a return statement to return a value.

Lambda expressions: They are used to create anonymous functions, also known as "lambda functions." These functions do not have a name and can have only one expression. Lambda expressions are typically used for simple, short functions where a full def statement might be unnecessarily verbose.

2. What is the benefit of lambda?

Ans. - Lambda expressions allow you to write simple functions in a compact and concise manner without the need for a full def statement.

- They are often used for short and straightforward functions, making the code more readable and maintainable.

- Lambda functions are handy when we need a function for a short period and don't want to define a named function separately.

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

map(function, iterable): Applies the specified function to all items in the iterable and returns an iterator with the results.

filter(function, iterable): Filters the elements in the iterable based on the given function, returning an iterator with the elements that pass the filter.

reduce(function, iterable): Applies the function cumulatively to the items of the iterable from left to right, returning a single reduced value.

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

Function annotations allow you to attach metadata to function arguments and the return value. They are not used by the Python interpreter for any internal purpose but can be helpful for documentation, type checking, or code analysis tools. Annotations are optional, and their primary purpose is to provide additional information about the function's signature.

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

Ans. Recursive functions are functions that call themselves within their definition. They are a powerful concept used in solving problems that can be broken down into smaller, identical subproblems. Recursive functions have a base case that specifies when the function should stop calling itself to prevent infinite recursion.

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

Ans. Use descriptive names that indicate the purpose of the function.

Keep functions focused and relatively short to improve readability and maintainability.

Aim for a limited number of arguments and avoid using mutable objects as default values.

Clearly define what the function should return, and avoid using return values for multiple purposes.

Include docstrings to describe the purpose of the function, its arguments, and its return value.

Avoid overly complex functions; consider breaking complex tasks into smaller functions.

Design functions that can be easily reused in different parts of the codebase.

Handle exceptions and error cases appropriately and provide meaningful error messages.

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

Ans.

Return values: Functions can return one or more values using the return statement.

Printing: Functions can use print() to display information directly to the console.

Modifying mutable objects: Functions can modify mutable objects passed as arguments to communicate results back to the caller.

Function annotations: Annotations can provide additional information to the caller about the function's arguments and return value.

Exceptions: Functions can raise exceptions to indicate errors or exceptional conditions to the caller. The caller can then handle the exceptions accordingly.