Q1 . What is the relationship between def statements and lambda expressions ?

The relationship between def statements and lambda expressions is that both are used to define functions, but they have some key differences:

Lambda expressions are limited in terms of complexity due to their single-expression nature. They can't contain multiple statements or complex control structures. def statements, on the other hand, allow for more complex functions.

2. What is the benefit of lambda?

Lambda functions allow you to define small, simple functions in a single line of code. This is especially useful when the function logic is short and can be expressed within an expression.

Lambda functions are anonymous and don't have a separate name, which can be beneficial when the function is only used in a specific context.

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

map:

The map function applies a given function to each item in an iterable and returns an iterator containing the results of the function calls.

Syntax: map(function, iterable)

Use Case: Use map when you want to transform each item in an iterable using the same operation.

filter:

The filter function filters out items from an iterable based on a given function that returns a Boolean value (True or False).

Syntax: filter(function, iterable)

Example:

numbers = [1, 2, 3, 4, 5]

even\_numbers = filter(lambda x: x % 2 == 0, numbers)

Use Case: Use filter when you want to selectively keep items from an iterable based on a certain condition.

reduce:

Purpose: The reduce function (from the functools module) is used to repeatedly apply a binary function to the items of an iterable in a cumulative way, reducing the iterable to a single value.

Syntax: reduce(function, iterable[, initializer])

Example:

from functools import reduce

numbers = [1, 2, 3, 4, 5]

sum\_of\_numbers = reduce(lambda x, y: x + y, numbers)

Use Case: Use reduce when you want to perform a cumulative operation on the items of an iterable, like calculating the sum or product

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

A recursive function is a type of function that calls itself as a part of its execution process. In other words, it's a function that solves a problem by breaking it down into smaller instances of the same problem. Recursive functions are commonly used in programming to solve problems that can be divided into simpler sub-problems of the same kind.

Recursive functions have two main components:

Base Case: This is the simplest scenario or case that can be solved directly without further recursion. It acts as a stopping point for the recursion.

Recursive Case: This is the scenario where the function calls itself with a modified or smaller version of the problem. This leads to the problem being broken down into smaller subproblems, which are eventually solved by the base case.

def factorial(n):

if n == 0: # Base case

return 1

else: # Recursive case

return n \* factorial(n - 1)

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

Single Responsibility Principle (SRP): Each function should have a single, well-defined purpose or responsibility.

Descriptive Names: Choose clear and descriptive names for your functions that convey their purpose.

Function Length: Keep your functions relatively short and focused.

Use Parameters and Return Values: Pass necessary data as parameters to functions and return results using return values. Avoid relying on global variables whenever possible.

Modularity and Reusability: Design functions to be modular and reusable.

Avoid Nesting Too Deeply

Comments and Documentation: Include comments where necessary to explain complex logic, algorithms, or any non-obvious decisions. However, aim to write code that is self-explanatory through its variable names and structure.

Error Handling: Design functions to handle errors gracefully. Use exception handling mechanisms where appropriate to provide meaningful error messages and to prevent crashing the entire application.

Avoid Duplication (DRY Principle): Don't Repeat Yourself (DRY).

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

Return Values: A return value is a piece of data that the function produces and sends back to the code that called it.

Global Variables: Although not recommended in most cases, functions can also communicate results by modifying global variables. However, this can make the code harder to understand and maintain, and it can lead to unintended side effects.

Using a Container Object:

Functions can also communicate results by modifying the contents of a container object that is passed as an argument. This is often seen in cases where multiple values need to be returned from a function.