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

**Relationship Between def Statements and Lambda Expressions:**

* **def Statements:** Define a named function that can have multiple lines of code, including complex logic, multiple parameters, and return values.
* **Lambda Expressions:** Define an anonymous (unnamed) function in a single line. It’s used for simple operations and is typically employed in scenarios requiring short-lived functions.

1. What is the benefit of lambda?

**Benefits of Lambda:**

* **Conciseness:** Lambda functions allow you to write small, throwaway functions without the need for a named definition.
* **Inline Use:** Ideal for use in higher-order functions like map, filter, and sorted where a small function is needed temporarily.
* **Functional Programming:** Supports functional programming concepts like passing functions as arguments.

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

 **map:** Applies a function to all items in an iterable and returns a map object (an iterator) with the results.

* **Example:** map(lambda x: x + 1, [1, 2, 3]) results in [2, 3, 4].

 **filter:** Applies a function to each item in an iterable and returns an iterator with items that evaluate to True.

* **Example:** filter(lambda x: x % 2 == 0, [1, 2, 3]) results in [2].

 **reduce:** (from functools module) Applies a binary function cumulatively to items in an iterable, reducing it to a single value.

* **Example:** reduce(lambda x, y: x + y, [1, 2, 3]) results in 6.

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

**Function Annotations:**

* **Definition:** Provide a way to attach metadata to function parameters and return values. They don’t affect the function's behavior but can be used for documentation, type checking, and other purposes.
* **Syntax:** Added after parameter names and return type using ->.

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

 **Definition:** Functions that call themselves to solve a smaller instance of the same problem.

 **Usage:** Useful for problems that can be broken down into smaller, similar subproblems (e.g., factorial, Fibonacci sequence).

def factorial(n: int) -> int:

if n == 0:

return 1

else:

return n \* factorial(n - 1)

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

**Single Responsibility:** Each function should perform a single task or operation.

**Clarity:** Use descriptive names and keep the function’s logic clear and understandable.

**Modularity:** Functions should be modular, allowing easy reuse and testing.

**Avoid Side Effects:** Functions should ideally not modify global state or have unintended effects.

**Parameter Handling:** Limit the number of parameters to avoid complexity.

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

**1.Return Values:** Use the return statement to send results back to the caller.

**Example:** return result

**2.Output Parameters:** Modify parameters passed by reference (e.g., lists or dictionaries).

**Example:** def update\_list(lst): lst.append(1)

**3.Side Effects:** Functions can alter external state (e.g., global variables, files).

**Example:** global\_variable = 42