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

2. What is the benefit of lambda?

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

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

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

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

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

**ANS:**1. - Both `def` statements and lambda expressions are used to define functions in Python.

- `def` statements are used to create named functions, which can have multiple expressions and statements within them. They allow for more complex logic and can include documentation strings.

- Lambda expressions, on the other hand, are used to create anonymous functions. They are typically simpler and shorter than named functions and consist of a single expression. Lambdas are often used for short functions that are passed as arguments to other functions.

2.

- Conciseness: Lambda expressions are usually shorter and more compact than named functions, making them useful for simple operations.

- Readability: Lambdas can often make code more readable when used appropriately, especially in cases where the function's purpose is clear from context.

- Functional programming: Lambdas are often used in functional programming paradigms, such as with the `map`, `filter`, and `reduce` functions, to create concise and expressive code.

3.

- map: Applies a function to each element of an iterable and returns an iterator of the results.

- filter: Filters elements of an iterable based on a function that returns `True` or `False`, returning only the elements for which the function returns `True`.

- reduce: Applies a function of two arguments cumulatively to the items of an iterable, reducing it to a single value. In Python 3, `reduce` is part of the `functools` module.

4.

- Function annotations allow you to attach metadata to the parameters and return value of a function declaration.

- They are defined using a colon after the parameter name, followed by the annotation expression.

- Annotations are completely optional and do not change the behavior of the function, but they can be used by tools for documentation generation, static type checking, and IDE auto-completion.

- Example: `def greet(name: str, age: int) -> str:`

5. - Recursive functions are functions that call themselves in their definition.

- They are used to solve problems that can be broken down into smaller, similar sub-problems.

- Proper termination conditions (base cases) are necessary to prevent infinite recursion.

- Examples include factorial calculation, Fibonacci sequence generation, and tree traversal algorithms.

6.

- Follow the single responsibility principle: Each function should do one thing and do it well.

- Keep functions short and focused: Aim for functions that are easy to understand and maintain.

- Use meaningful names for functions and parameters: Choose descriptive names that convey the purpose of the function and its inputs.

- Avoid side effects: Functions should ideally have no side effects beyond their intended purpose.

- Write docstrings: Provide clear and concise documentation for each function, explaining its purpose, parameters, and return values.

7.

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

- Side effects: Functions can modify mutable objects or global variables, affecting program state.

- Exceptions: Functions can raise exceptions to indicate errors or exceptional conditions that the caller needs to handle.

- Callbacks: Functions can accept other functions (callbacks) as arguments and call them to communicate results or handle events asynchronously.