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.

ANSWER

1. **Relationship between def statements and lambda expressions:**
   * Both **def** statements and **lambda** expressions are used to create functions in Python.
   * **def** is used to define named functions with a block of code, while **lambda** is used to create anonymous functions (functions without a name) with a single expression.
   * **def** functions are typically used for more complex logic and can include multiple statements, whereas **lambda** functions are often used for simple, one-liner functions.
2. **Benefits of lambda:**
   * Conciseness: Lambdas are concise and allow you to define small, simple functions in a single line.
   * Readability: They can make code more readable when used appropriately, especially with functions like **map**, **filter**, and **sorted**.
   * Avoiding unnecessary function names: For short-lived functions that are used only once, lambdas can eliminate the need to define a full function with a name.
3. **Comparison of map, filter, and reduce:**
   * **map**: Applies a given function to each item in an iterable and returns a new iterable with the results.
   * **filter**: Filters elements from an iterable based on a given function's condition and returns a new iterable with the filtered elements.
   * **reduce**: Accumulates values from an iterable by applying a function cumulatively and returns a single result. Note that in Python 3, **reduce** has been moved to the **functools** module.
4. **Function annotations and their use:**
   * Function annotations are a way to add metadata to the parameters and return values of functions. They don't affect the function's behavior but provide information about the function's intended usage.
   * Annotations are specified using colons (e.g., **def function\_name(param: annotation) -> annotation:**).
   * They can be used for documentation, type hinting, and code analysis tools. Type hints can help catch type-related errors in your code.
5. **Recursive functions and their use:**
   * Recursive functions are functions that call themselves as part of their operation. They are used to solve problems that can be divided into smaller, similar sub-problems.
   * Recursion is commonly used for tasks like traversing tree structures, computing factorials, and solving problems in a divide-and-conquer fashion.
6. **Design guidelines for coding functions:**
   * Keep functions focused on a single task (the "single responsibility principle").
   * Use descriptive and meaningful function and parameter names.
   * Document your functions with docstrings to explain their purpose and usage.
   * Avoid using global variables inside functions when possible.
   * Ensure your functions have clear input and output expectations.
   * Follow the DRY principle (Don't Repeat Yourself) by reusing code within functions.
7. **Ways functions can communicate results to a caller:**
   * Return values: Functions can return data or results using the **return** statement.
   * Printing: Functions can print information to the console for immediate output.
   * Modifying mutable objects: Functions can modify objects passed as arguments (e.g., lists, dictionaries) in-place, allowing changes to be seen outside the function.
   * Raising exceptions: Functions can raise exceptions to signal errors or exceptional conditions to the caller.