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. Both **def** statements and **lambda** expressions are used to define functions in Python. The main difference is that **def** statements define named functions with a block of statements, while **lambda** expressions define anonymous functions with a single expression. **def** statements can have multiple statements, can include docstrings and annotations, and can be used to define methods, while **lambda** expressions are limited to a single expression and can only return a value.
2. The benefit of **lambda** is that it allows for the creation of simple, anonymous functions without the need to define a named function. This can make code more concise and easier to read by avoiding the need to define and call small, one-time-use functions.
3. **map**, **filter**, and **reduce** are all higher-order functions in Python that take functions as arguments and apply them to iterables.

* **map** takes a function and an iterable and applies the function to each element of the iterable, returning a new iterable with the transformed values.
* **filter** takes a function and an iterable and applies the function to each element of the iterable, returning a new iterable with only the elements for which the function returns True.
* **reduce** takes a function and an iterable and applies the function to the first two elements, then applies the function to the result and the next element, and so on until all elements have been processed, returning a single value.

1. Function annotations are a feature in Python that allow you to attach metadata to the arguments and return value of a function. They are used to document the expected types or values of the arguments and return value, and can be used by tools like IDEs or linters to provide hints and warnings. Function annotations are expressed as expressions that follow the argument or return value, enclosed in parentheses and preceded by a colon.
2. Recursive functions are functions that call themselves as part of their execution. They are often used to solve problems that can be broken down into smaller, similar subproblems, such as computing the factorial of a number or traversing a tree structure. Recursive functions typically have a base case that terminates the recursion and one or more recursive cases that call the function on smaller subproblems.
3. Some general design guidelines for coding functions include:

* Functions should be focused on a single, well-defined task.
* Functions should be named descriptively to indicate their purpose.
* Functions should be relatively small and do one thing well.
* Functions should take a small number of arguments, and those arguments should be well-defined and unambiguous.
* Functions should have clear, easy-to-understand documentation.
* Functions should avoid side effects that modify global state.
* Functions should have a single entry and exit point.

1. Functions can communicate results to a caller in several ways:

* They can return a value using the **return** statement.
* They can modify a mutable argument passed in by reference.
* They can raise an exception to indicate an error or unexpected condition.
* They can print output to the console or a file.
* They can call other functions that in turn communicate results to the caller.