**ANSWERS**

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

**Ans.** The relationship between def. statements and lambda expressions is as follows:

* **def** statements are used to define named functions in Python. These functions can have a name, parameters, and a body that contains statements.
* Lambda expressions (also known as lambda functions) are anonymous functions that can be defined using the **lambda** keyword. They are often used for simple operations and can be defined in a single line.

**Q2. What is the benefit of lambda?**

**Ans.** The benefits of lambda is as follows:

* **Conciseness:** Lambda expressions allow you to define small, inline functions without the need for a full function definition using def.
* **Readability:** Lambda expressions can make the code more readable when the operation being performed is straightforward and doesn't require a separate function definition.
* **Functional programming:** Lambdas are commonly used in functional programming constructs like map, filter, and reduce.

**Q3. Compare and contrast map, filter, and reduce.**

**Ans.** Comparison and contrast between map, filter and reduce is as follows:

* map: Applies a given function to each element of an iterable and returns an iterator of the results.
* filter: Filters out elements from an iterable based on a given condition, returning an iterator of the elements that satisfy the condition.
* reduce: Applies a binary function to the elements of an iterable in a cumulative way, reducing the iterable to a single value.

**Q4. What are function annotations, and how are they used?**

**Ans.** Function annotations and their uses is as follows:

* Function annotations are optional metadata that can be added to the parameters and return value of a function using the syntax def function\_name(param: annotation) -> annotation.
* Annotations can be used to indicate the expected types of function arguments and return values. They are not enforced by the interpreter but can provide helpful information for documentation and code analysis.

**Q5. What are recursive functions, and how are they used?**

**Ans.** Recursive functions and their uses is as follows:

* Recursive functions are functions that call themselves in their own body. They are used to solve problems that can be broken down into simpler instances of the same problem.
* Base case(s) are crucial in recursive functions to prevent infinite recursion.
* Example: A classic recursive function is the factorial calculation: factorial(n) = n \* factorial(n-1) with the base case factorial(0) = 1.

**Q6. What are some general design guidelines for coding functions?**

**Ans.** Some general design guidelines for coding functions are as follows:

* **Single Responsibility Principle:** Functions should have a clear and singular purpose.
* **Modularity:** Break down complex tasks into smaller, reusable functions.
* **Descriptive Naming:** Use meaningful names that convey the function's purpose.
* **Avoid Side Effects:** Functions should ideally not modify global state and should rely on parameters and return values.
* **Maintainability:** Write code that's easy to understand and maintain.

**Q7. Name three or more ways that functions can communicate results to a caller.**

**Ans.** Ways through which functions can communicate results to a caller are as follows:

* **Return Statements:** Functions can use return statements to send data back to the caller.
* **Parameters:** Functions can accept parameters to receive input from the caller.
* **Exceptions:** Functions can raise exceptions to indicate errors or unexpected conditions to the caller.