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

Both def statements and lambda expressions are used to define functions in programming languages. However, they differ in their syntax, purpose, and capabilities:

1. def statements:
    - A def statement is used to define a named function in most programming languages.
    - It consists of the keyword def, followed by the function name, parameter list in parentheses, a colon, and a block of indented code that defines the function's behavior.
    - Defining a function with def allows for more complex functions, including multiple statements, conditional logic, loops, and more.
    - Functions defined using def can be reused throughout the code.
2. lambda expressions:
    - A lambda expression (also known as a lambda function or anonymous function) is a compact way to define small, simple functions in languages that support functional programming paradigms.
    - It is usually expressed using the lambda keyword, followed by parameter list and an expression that gets evaluated when the lambda is called.
    - Lambdas are often used for short, simple operations and are commonly passed as arguments to higher-order functions like map, filter, and sort.
    - Lambdas are more limited than def statements, as they are restricted to a single expression and cannot contain complex statements or multiple lines of code.

2. What is the benefit of lambda?

The benefit of lambda expressions is their conciseness and ability to define simple functions inline, without the need for a separate def statement. This is particularly useful when you need to pass a small function as an argument to another function (e.g., in functional programming constructs like map, filter, and sort). Lambdas save you from writing a full function definition when you only need a brief operation

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

The map function is used to apply a given function to each element of a collection (like a list or an array) and returns a new collection with the results. The filter function is used to select elements from a collection that satisfy a given condition, and it returns a new collection containing only the elements that meet the condition. The reduce function (also known as fold in some languages) is used to apply a binary operation to the elements of a collection in a cumulative way, resulting in a single value.

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

Function annotations are used to indicate the expected types of function parameters and return values, improving code clarity and enabling potential static type checking.

```
def sample(s:str)->str:
```

```
    return s*2
```

In above code example:

- **s: str** is a parameter annotation, indicating that the **s** parameter is expected to be a string.
- **-> str** is a return type annotation, indicating that the function will return a string.

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

Recursive function is function that call itself. The function has 2 parts. The first part is base case which is simplistic condition where function does not call itself rather return some value. The base case acts as stopping condition for recursive function. The second part of function is recursive case where function calls itself each time with modified input/condition. Example. of recursive function is as follows:

```
def fibonacci(n):
  if n <= 0:return 0
  elif n == 1:return 1
  else:return fibonacci(n - 1) + fibonacci(n - 2)
```

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

Some general guidelines to consider when coding functions:

1. Use lowercase letters and underscores (snake_case) for function names. Choose descriptive names that convey the purpose of the function.
2. Use lowercase letters and underscores for function arguments. If an argument name consists of multiple words, separate them with underscores.
3. Use docstrings (triple-quoted strings) to document your functions. Describe the purpose of the function, its parameters, return values, and any notable behavior.
4. PEP 8 suggests that functions should ideally be no longer than 20 lines.
5. Functions should have a single responsibility. If a function performs multiple tasks, consider breaking it into smaller functions.
6. Use blank lines to separate logical blocks within a function, enhancing readability.
7. Single Responsibility Principle (SRP): Each function should have a clear and specific purpose. It should perform one task and do it well. This enhances readability and makes code easier to maintain.
8. Use comments sparingly, but when necessary, ensure they are clear and concise. Comments should explain why something is being done, not what is being done (which should be clear from the code itself).

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

return value, yield statements, exceptions, global variable, modifying mutables(list,dict)