**ADITYA AMIN ASSIGN : 24**

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

while both def statements and lambda expressions are used to define functions in Python, they differ in terms of syntax, functionality, and use cases. def statements are typically used for complex and reusable functions, while lambda expressions are used for short-lived, simple, and concise functions, often used as arguments to higher-order functions.

1. What is the benefit of lambda?

The primary benefit of using lambda expressions, also known as lambda functions, in Python is their conciseness and ability to create small, anonymous functions on the fly.

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

map(), filter(), and reduce() are three built-in functions in Python that are used for data manipulation and transformation. They have different purposes, functionality, and use cases. map() is used to apply a function to each item in an iterable and create a new iterable with the results, filter() is used to selectively include or exclude elements from an iterable based on a condition, and reduce() is used to successively apply a function to the elements of an iterable and reduce it to a single value.

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

Function annotations are a feature in Python that allow you to add metadata, in the form of optional type hints or arbitrary expressions, to the parameters and return value of a function. Function annotations are used to provide hints about the expected types of function arguments and return values, but they do not affect the actual behavior of the function at runtime.

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

"""

Greets a person with the given name and age.

return f"Hello, {name}! You are {age} years old."

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

A recursive function is a function that calls itself during its own execution. In other words, a recursive function solves a problem by breaking it down into smaller instances of the same problem, and then solving those smaller instances using the same function.

def factorial(n):

if n == 0:

return 1 # base case

else:

return n \* factorial(n-1) # recursive case

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

Designing functions is an important aspect of writing clean, efficient, and maintainable code. Here are some general design guidelines for coding functions:

Single Responsibility Principle (SRP): Functions should have a single responsibility and do one thing well.

Modularity: Functions should be modular, meaning they should be self-contained and independent of each other.

Descriptive Naming: Functions should have descriptive and meaningful names that reflect their purpose and functionality. This makes the code more readable and understandable.

Proper Parameter Passing: Functions should receive input parameters in a clear and consistent way, and should not rely on global state or side effects.

Error Handling: Functions should handle errors and exceptions gracefully, and provide appropriate error messages or return values when necessary.

Code Reusability: Functions should be designed to be reusable, meaning they can be used in multiple contexts or scenarios.

Efficiency: Functions should be designed to be efficient, both in terms of time and space complexity.

Documentation: Functions should be properly documented, both in terms of inline comments and external documentation.

Testability: Functions should be designed to be easily testable, with well-defined input and output parameters.

Code Style: Functions should follow the established coding style guidelines of the programming language or framework being used.

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

There are several ways that functions can communicate results to a caller in programming. Here are three common ways:

Return Values: Functions can return a value to the caller using the return statement. The returned value can be of any data type, such as numbers, strings, booleans, objects, or custom data types. The caller can then capture and use the returned value as needed.

Example:

def add(a, b):

return a + b

result = add(3, 5) # Function returns 8, which is stored in 'result' variable

print(result) # Prints 8

Modifying Mutable Objects: Functions can also communicate results to the caller by modifying mutable objects that are passed as arguments. Mutable objects, such as lists or dictionaries, can be modified in place within the function, and the changes will be visible outside the function, as the same object is being modified.

Example:

def append\_element(lst, element):

lst.append(element)

my\_list = [1, 2, 3]

append\_element(my\_list, 4) # Function appends 4 to 'my\_list'

print(my\_list) # Prints [1, 2, 3, 4]

Side Effects: Functions can also communicate results to the caller through side effects, which are modifications or actions that the function performs outside of its return value or argument list. Side effects can include printing output to the console, writing to a file, updating a global variable, or sending data over a network.

Example:

def print\_greeting(name):

print("Hello, " + name + "!") # Function prints a greeting to the console

print\_greeting("Alice") # Prints "Hello, Alice!"