1. **Lambda Functions (Anonymous Functions)**

Lambda functions are small, anonymous functions defined using the lambda keyword. They are used when a small function is needed for a short period.

**Syntax:**

lambda arguments: expression

**Example:**

add = lambda a, b: a + b

print(add(5, 3))  # Output: 8

1. **Recursion in Functions**

A function is called **recursive** if it calls itself. Recursion is a powerful tool for solving problems that can be broken down into smaller, similar problems.

**Example: Factorial using Recursion**

def factorial(n):

    """This function returns the factorial of a number using recursion."""

    if n == 1:

        return 1

    else:

        return n \* factorial(n - 1)

print(factorial(5))  # Output: 120

1. **Variable-Length Arguments**

Sometimes, you may need a function that can accept any number of arguments. Python provides two ways to handle this:

* **\*args** for non-keyword arguments
* **\*\*kwargs** for keyword arguments

**Example:**

def add\_all(\*args):

    """This function returns the sum of all arguments."""

    return sum(args)

print(add\_all(1, 2, 3))        # Output: 6

print(add\_all(10, 20, 30, 40)) # Output: 100

1. **Scope of Variables**

* **Local Scope:** Variables defined inside a function are in the local scope and are not accessible outside the function.
* **Global Scope:** Variables defined outside all functions are in the global scope and can be accessed anywhere.

**Example:**

x = 10  # Global variable

def my\_function():

    x = 5  # Local variable

    print(x)

my\_function()  # Output: 5

print(x)       # Output: 10

1. **Recursive Function to Calculate Fibonacci Series**

# This function returns the nth Fibonacci number using recursion.

def fibonacci(n):

    if n <= 1:

        return n

    else:

        return fibonacci(n - 1) + fibonacci(n - 2)

# Print the first 10 Fibonacci numbers

for i in range(10):

    print(fibonacci(i), end=" ")

1. **\*\*Function with Keyword Arguments Using kwargs**

# This function displays information about a person using keyword arguments.

def display\_info(\*\*kwargs):

    for key, value in kwargs.items():

        print(f"{key}: {value}")

# Call the function with keyword arguments

display\_info(name="Alice", age=30, profession="Engineer")

**Explanation of the Optimization**

1. **Using Sets for Membership Checks**:
   * Checking membership in a set (e.g., num in A) is O(1) on average, whereas checking in a list is O(n).
   * By converting A and B to sets, you minimize the time complexity of membership checks.