**2403a51268**

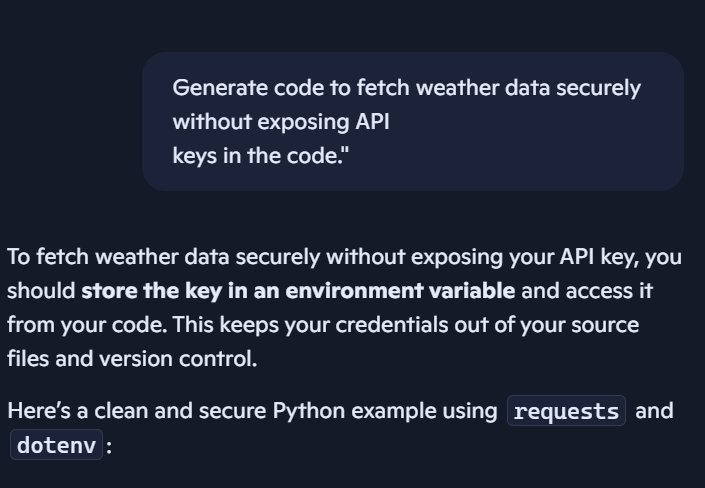
**Srinija G**

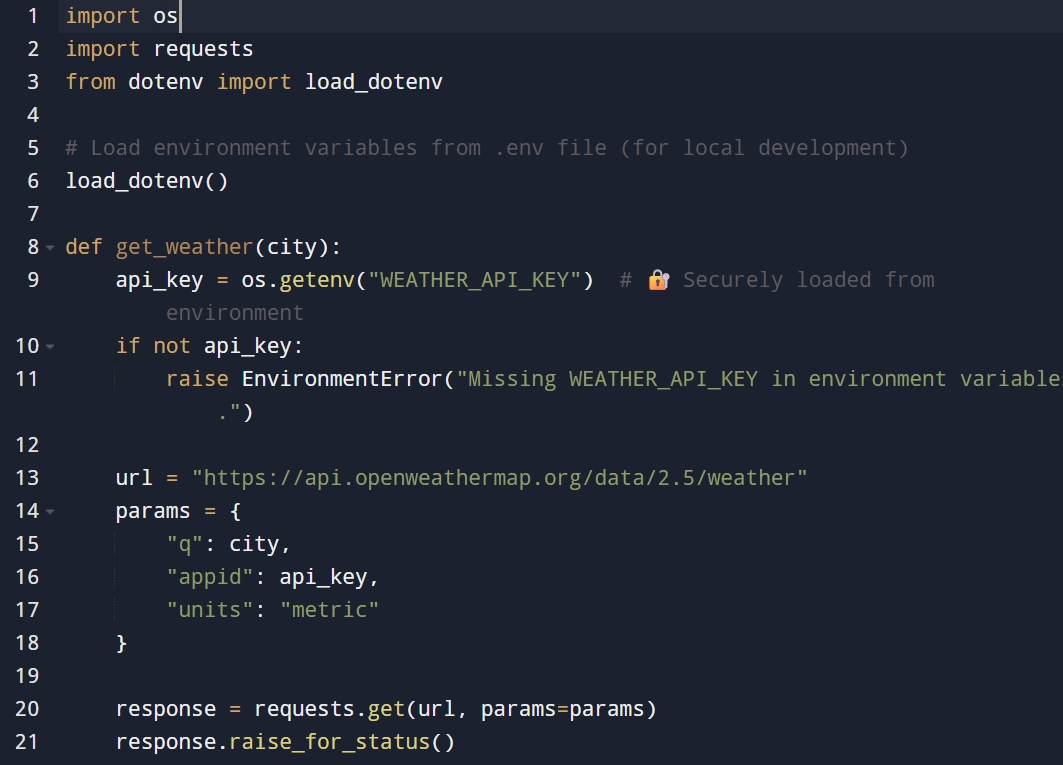
**Batch12**

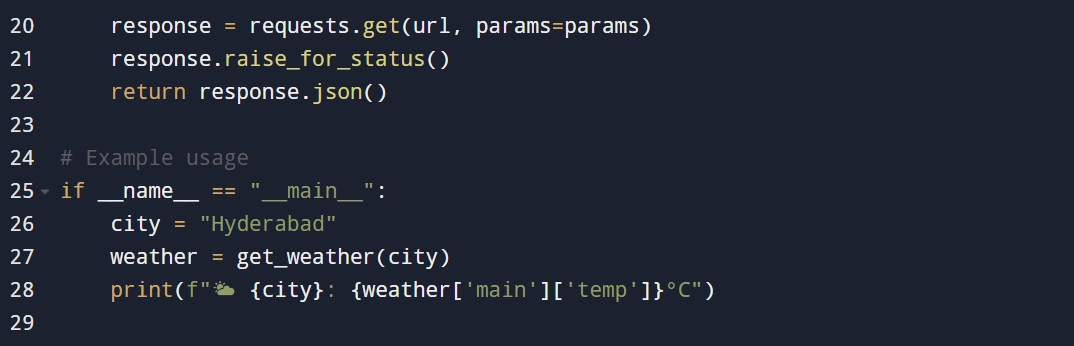
**AI-ASSISTED-CODING**

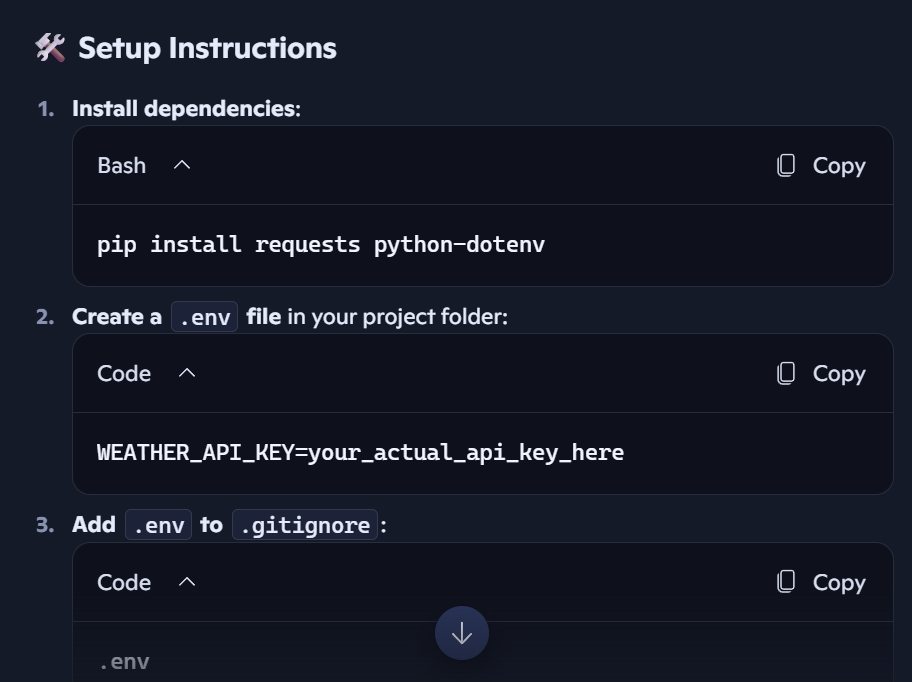
**Assignment-5.1**

**Task Description #1** (Privacy in API Usage)  
Task: Use an AI tool to generate a Python program that connects to a  
weather API.  
Prompt:  
"Generate code to fetch weather data securely without exposing API  
keys in the code."

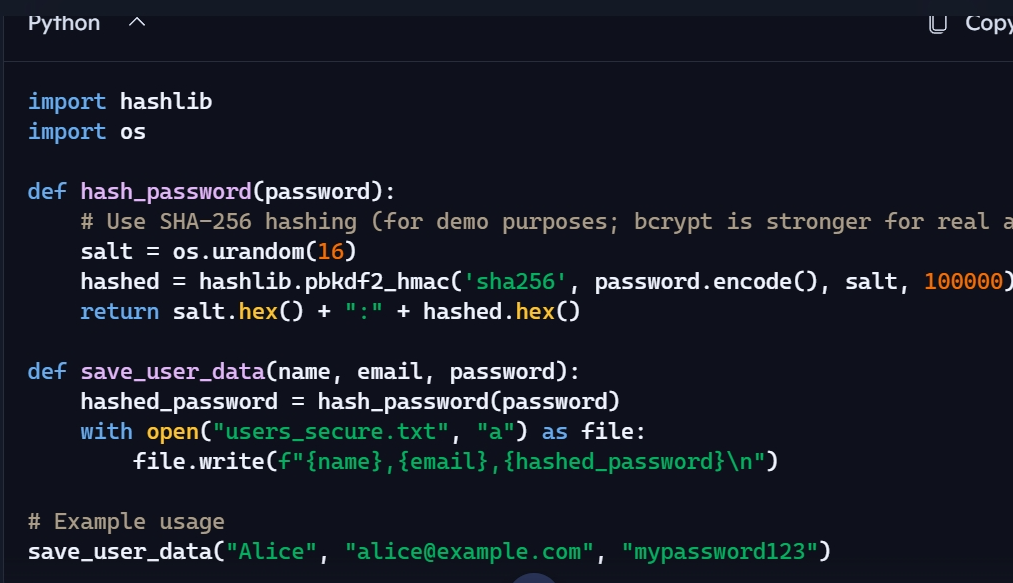
****

****

****

****

**Output:**  
• Original AI code (check if keys are hardcoded).  
• Secure version using environment variables.

**Task Description #2** (Privacy & Security in File Handling)  
Task: Use an AI tool to generate a Python script that stores user data  
(name, email, password) in a file.  
 

**Analyze:** Check if the AI stores sensitive data in plain text or without  
encryption.

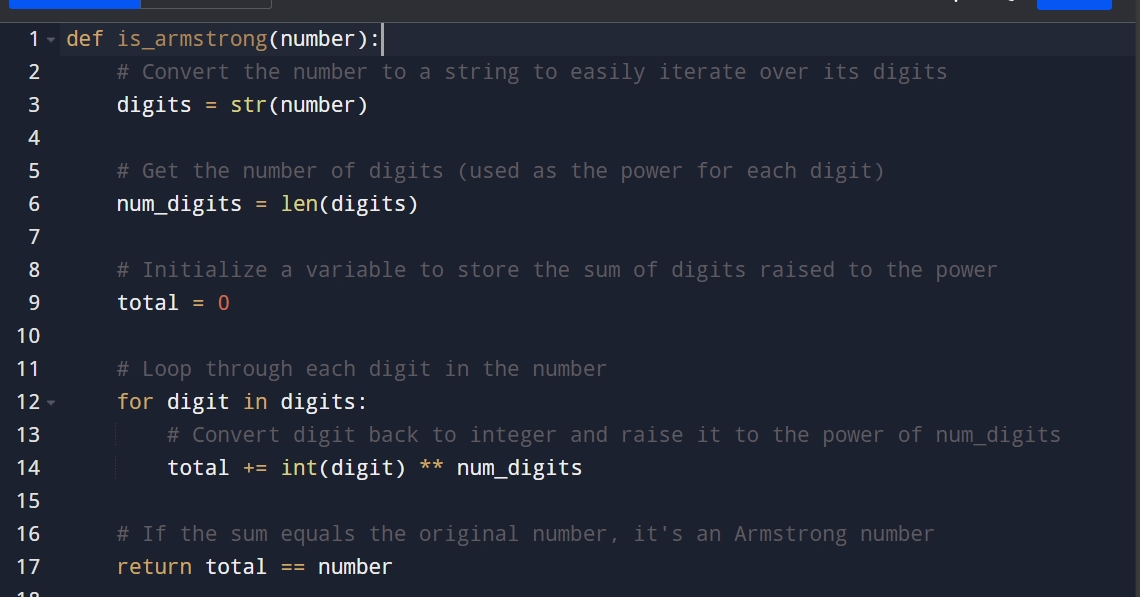
### Plain Text Storage Risks:

* **Passwords stored in plain text**: This is a major security flaw. If the file is accessed, all user credentials are exposed.
* **No encryption or hashing**: The script does not protect sensitive data in any way.
* **No access control or permissions**: Anyone with access to the file system could read or modify the data.
* **No input validation**: Could be vulnerable to injection or malformed data

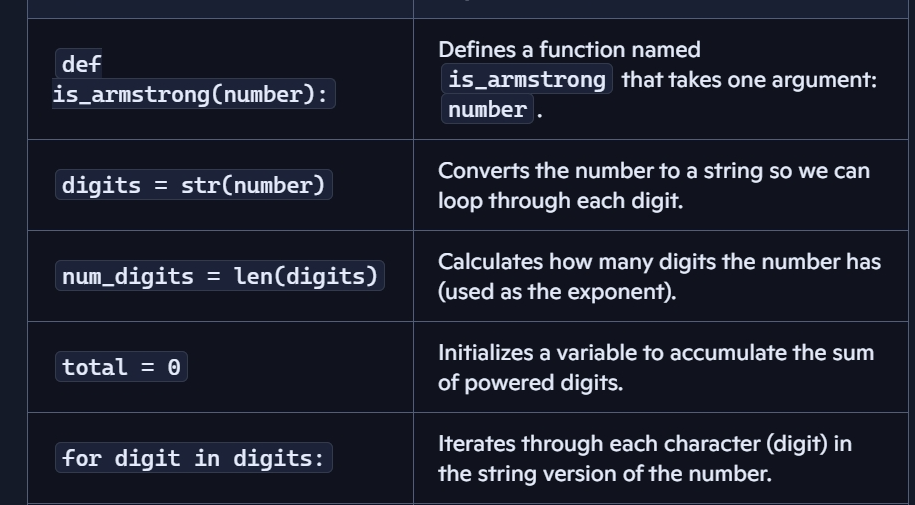
### Improvements:

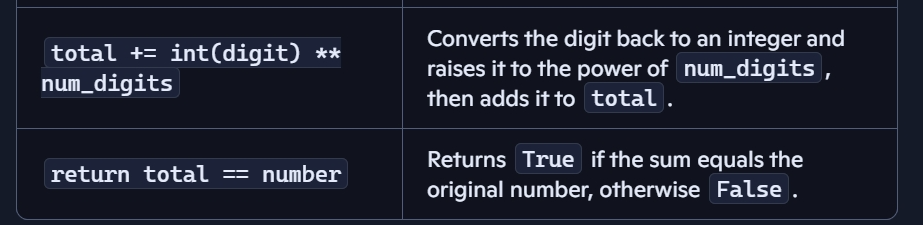
* **Password hashing with salt**: Protects against brute-force and rainbow table attacks.
* **No plain text passwords**: Even if the file is compromised, passwords are unreadable.
* **Modular design**: Easy to upgrade to stronger algorithms like bcrypt or argon2

**Output:**  
• Identified privacy risks.  
• Revised version with encrypted password storage (e.g., hashing).

**Task Description #3** (Transparency in Algorithm Design)  
Objective: Use AI to generate an Armstrong number checking function  
with comments and explanations.

**1. Ask AI to explain the code line-by-line.**



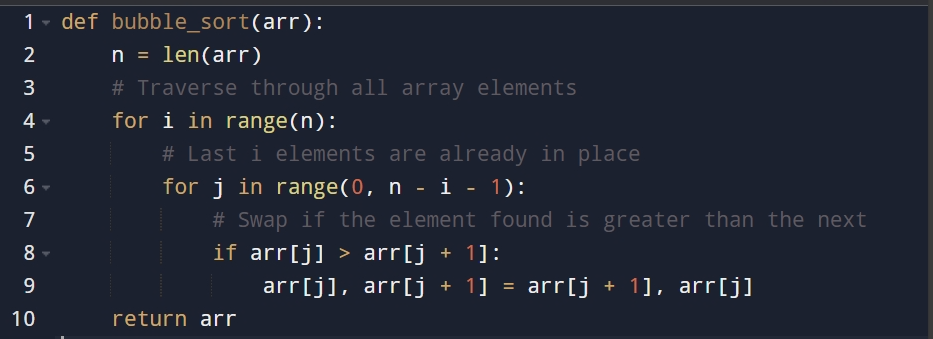
  
**2. Compare the explanation with code functionality.**

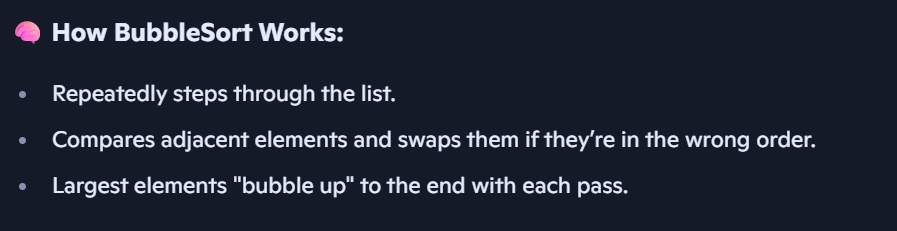
* **Explanation is accurate and complete.**
* **Each step is clearly described and matches the code’s behavior.**
* **No misleading or ambiguous logic.**
* **Transparent and beginner-friendly.**

**Output:**  
• Transparent, commented code.  
• Correct, easy-to-understand explanation.

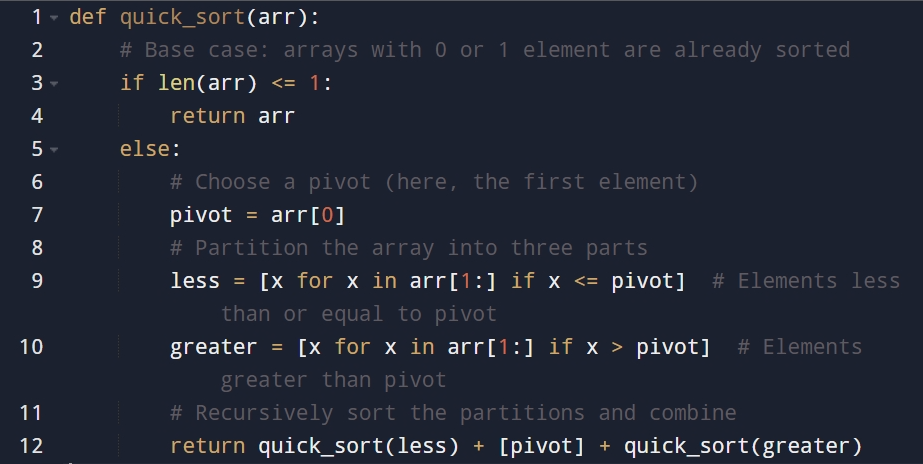
**Task Description #4** (Transparency in Algorithm Comparison)  
Task: Use AI to implement two sorting algorithms (e.g., QuickSort and  
BubbleSort).  
Prompt:  
"Generate Python code for QuickSort and BubbleSort, and include  
comments explaining step-by-step how each works and where they  
differ."

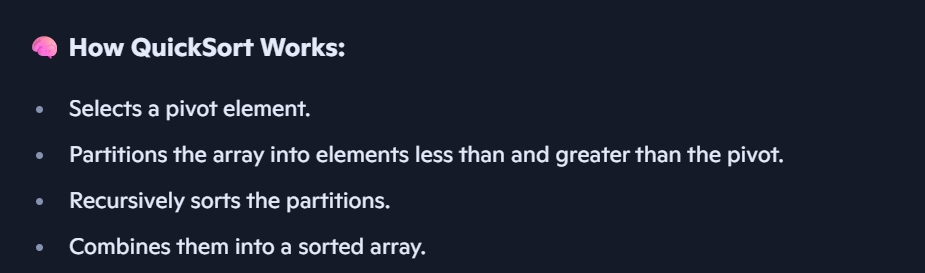
**BUBBLE-SORT:**





**QUICKE-SORT:**



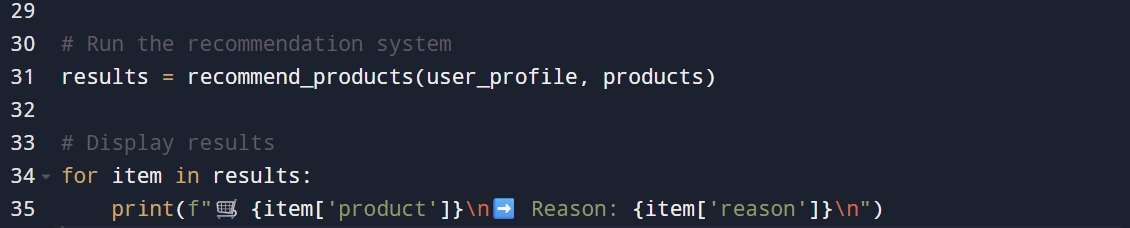


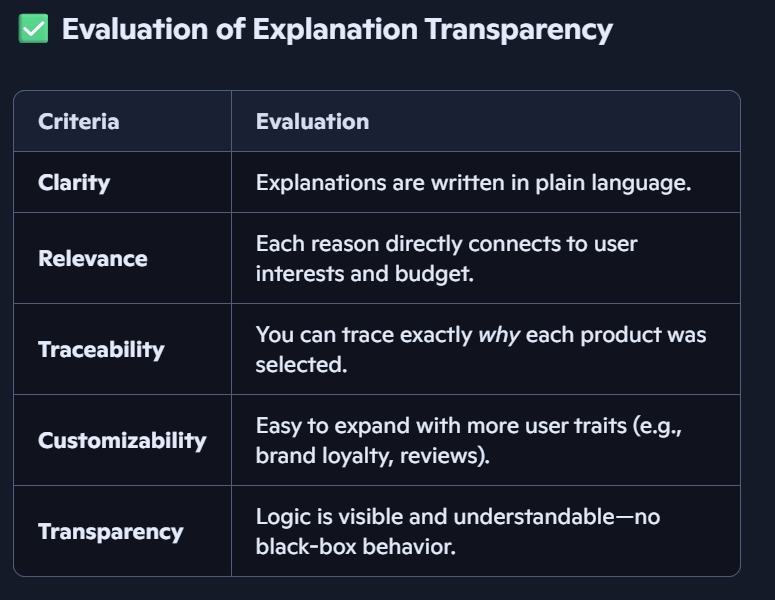


**Output:**  
• Code for both algorithms.  
• Transparent, comparative explanation of their logic and  
efficiency.

**Task Description #5** (Transparency in AI Recommendations)  
Task: Use AI to create a product recommendation system.  
Prompt:  
"Generate a recommendation system that also provides reasons for  
each suggestion."  
 

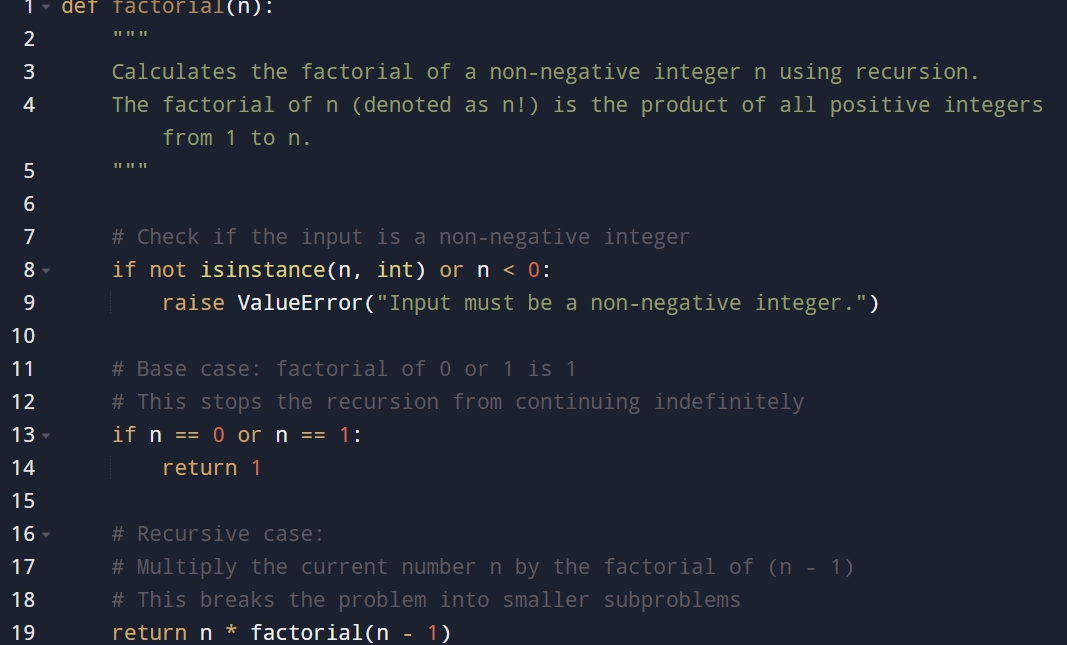






**Output:**  
• Code with explainable recommendations.  
• Evaluation of whether explanations are understandable

**Task Description #6** (Transparent Code Generation)  
Task: Ask AI to generate a Python function for calculating factorial  
using recursion.  
Prompt:  
"Generate a recursive factorial function with comments that explain  
each line and a final summary of the algorithm’s flow."



## **Line-by-Line Explanation**

* def factorial(n): Defines a function named factorial that takes one argument n.
* """ ... """ Docstring explaining what the function does and the mathematical definition of factorial.
* if n == 0 or n == 1: Checks for the base case. The factorial of 0 or 1 is defined as 1.
* return 1 Returns 1 when the base case is met, stopping further recursion.
* return n \* factorial(n - 1) This is the recursive step. The function calls itself with n - 1, multiplying the result by n.

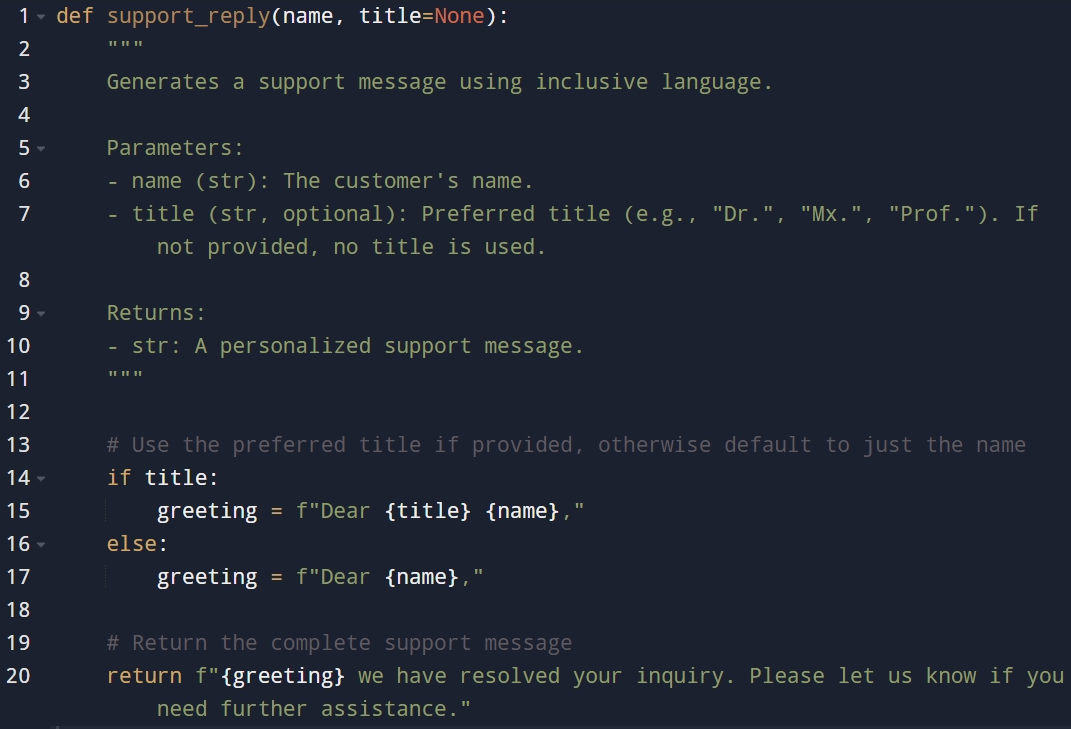
## **Recursive Flow**

## Recursion works by breaking down the problem into smaller subproblems:

1. **Start** with factorial(n).
2. **Call** factorial(n - 1) until it reaches the base case (n == 0 or 1).
3. **Unwind** the recursive calls by multiplying each result as the stack returns:
   * factorial(3) → 3 \* factorial(2)
   * factorial(2) → 2 \* factorial(1)
   * factorial(1) → 1 (base case)
   * Final result: 3 \* 2 \* 1 = 6

**Output:**  
• Fully commented code.  
• Clear documentation of how recursion works.

**Task Description #7** (Inclusiveness in Customer Support)  
Code Snippet:  
Task:  
Regenerate the code so that support messages use neutral language (e.g.,  
“Dear {name}”) and optionally accept preferred titles



**Output:**  
• Neutral, user-friendly support responses