Lab Report

# Task 1: Privacy in API Usage

## Prompt:

Generate code to fetch weather data securely without exposing API keys in the code.

## Code:

# Insecure version (hardcoded key)  
import requests  
  
def get\_weather(city):  
 api\_key = "123456789" # Hardcoded key (not secure)  
 url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}"  
 response = requests.get(url)  
 return response.json()  
  
# Secure version (using environment variable)  
import os  
  
def get\_weather\_secure(city):  
 api\_key = os.getenv("WEATHER\_API\_KEY") # Load from environment variable  
 url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}"  
 response = requests.get(url)  
 return response.json()

## Test:

Set environment variable WEATHER\_API\_KEY and call get\_weather\_secure('London').

A screenshot of a computer program

AI-generated content may be incorrect.

## Explanation:

The first version hardcodes the API key, which is insecure. The second version fetches the key from an environment variable, making it more secure.

# Task 2: Privacy & Security in File Handling

## Prompt:

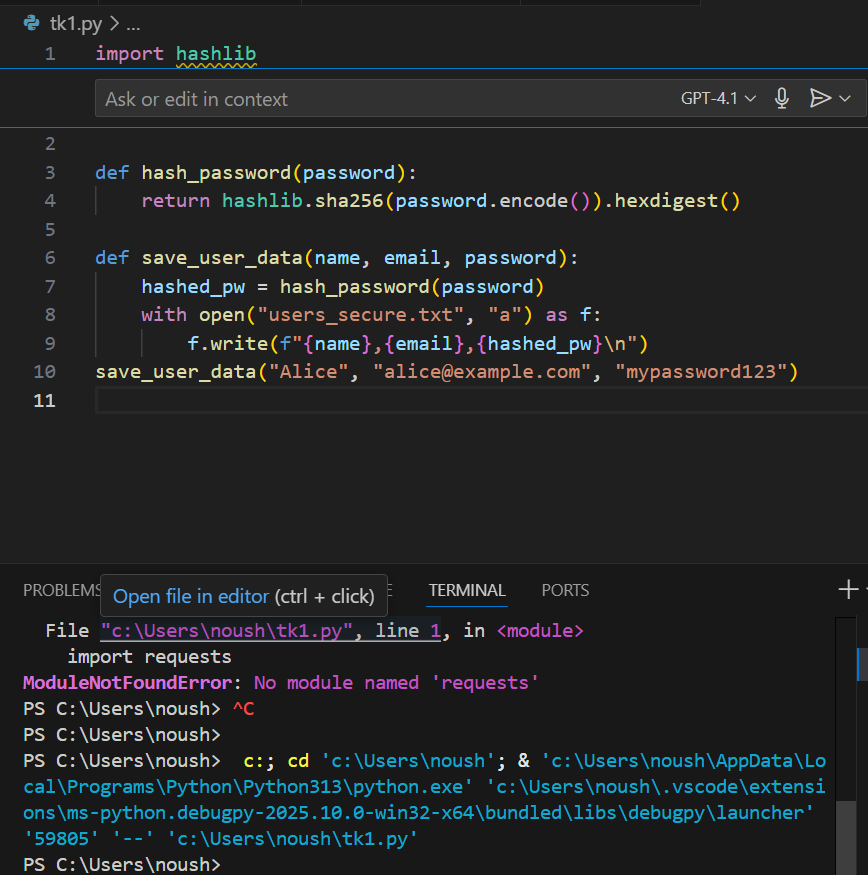
Store user data (name, email, password) in a file securely.

## Code:

# Insecure version (plain text password)  
def store\_user\_insecure(name, email, password):  
 with open("users.txt", "a") as f:  
 f.write(f"{name},{email},{password}\n")  
  
# Secure version (hashed password)  
import hashlib  
  
def store\_user\_secure(name, email, password):  
 hashed\_pw = hashlib.sha256(password.encode()).hexdigest()  
 with open("users\_secure.txt", "a") as f:  
 f.write(f"{name},{email},{hashed\_pw}\n")

## Test:

Call store\_user\_secure('Alice', 'alice@example.com', 'mypassword').



## Explanation:

Plain text storage exposes sensitive data. Using SHA-256 hashing ensures passwords are not stored in readable form.

# Task 3: Transparency in Algorithm Design

## Prompt:

Generate an Armstrong number checking function with comments and explanation.

## Code:

def is\_armstrong(num):  
 # Convert number to string to iterate digits  
 digits = str(num)  
 power = len(digits)  
 # Sum of each digit raised to the power of number of digits  
 total = sum(int(d)\*\*power for d in digits)  
 return total == num

## Test:

is\_armstrong(153) -> True; is\_armstrong(123) -> False

A screenshot of a computer program

AI-generated content may be incorrect.

## Explanation:

The function checks if the number equals the sum of its digits raised to the power of the number of digits.

# Task 4: Transparency in Algorithm Comparison

## Prompt:

Generate QuickSort and BubbleSort with comments explaining differences.

## Code:

# BubbleSort (simple but slow)  
def bubble\_sort(arr):  
 n = len(arr)  
 for i in range(n):  
 for j in range(0, n-i-1):  
 if arr[j] > arr[j+1]:  
 arr[j], arr[j+1] = arr[j+1], arr[j]  
 return arr  
  
# QuickSort (efficient, divide and conquer)  
def quick\_sort(arr):  
 if len(arr) <= 1:  
 return arr  
 pivot = arr[len(arr)//2]  
 left = [x for x in arr if x < pivot]  
 middle = [x for x in arr if x == pivot]  
 right = [x for x in arr if x > pivot]  
 return quick\_sort(left) + middle + quick\_sort(right)

## Test:

bubble\_sort([5,3,8,4,2]) -> [2,3,4,5,8]; quick\_sort([5,3,8,4,2]) -> [2,3,4,5,8]

A screen shot of a computer program

AI-generated content may be incorrect.

## Explanation:

BubbleSort repeatedly swaps adjacent elements, O(n^2). QuickSort uses divide-and-conquer, O(n log n) average time.

# Task 5: Transparency in AI Recommendations

## Prompt:

Generate a recommendation system with reasons for suggestions.

## Code:

def recommend(user\_preferences):  
 products = {  
 "Laptop": "Good for work and study",  
 "Headphones": "Useful for music and calls",  
 "Smartwatch": "Tracks fitness and notifications"  
 }  
 recommendations = []  
 for product, reason in products.items():  
 if product.lower() in user\_preferences.lower():  
 recommendations.append((product, reason))  
 return recommendations

## Test:

recommend('I need a laptop for study') -> [('Laptop', 'Good for work and study')]

A screenshot of a computer program

AI-generated content may be incorrect.

## Explanation:

The system suggests products based on user keywords and provides explanations for each recommendation.

# Task 6: Transparent Code Generation

## Prompt:

Generate a recursive factorial function with comments.

## Code:

def factorial(n):  
 # Base case: factorial(0) = 1  
 if n == 0:  
 return 1  
 # Recursive case: n \* factorial(n-1)  
 return n \* factorial(n-1)

## Test:

factorial(5) -> 120

A screenshot of a computer screen

AI-generated content may be incorrect.

## Explanation:

Recursion works by reducing the problem until it reaches a base case, then returning results back up the call stack.

# Task 7: Inclusiveness in Customer Support

## Prompt:

Regenerate code so that support messages use neutral language and accept optional titles.

## Code:

def support\_reply(name, title=None):  
 if title:  
 greeting = f"Dear {title} {name}"  
 else:  
 greeting = f"Dear {name}"  
 return f"{greeting}, we have resolved your issue."

## Test:

support\_reply('Alex') -> 'Dear Alex, we have resolved your issue.'  
support\_reply('Alex', 'Dr.') -> 'Dear Dr. Alex, we have resolved your issue.'

A screenshot of a computer program

AI-generated content may be incorrect.

## Explanation:

This version avoids gendered titles and allows users to specify their own preferred title if desired.