

AI Assisted Coding

Name:D.udayan

Hall.no:2303A54017

Assignment: 5.1

Task1:

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."

Expected Output:

- Original AI code (check if keys are hardcoded).
- Secure version using environment variables.

Ai generated code:

```
import requests

# WARNING: This is an INSECURE way to handle an API key.
# If you share this code or push it to a public repository,
# your API key will be exposed.
OPENWEATHER_API_KEY = "YOUR_HARDCODED_OPENWEATHER_API_KEY_HERE"
# Please replace 'YOUR_HARDCODED_OPENWEATHER_API_KEY_HERE' with your actual key if you wish to use it.

CITY = "London"
BASE_URL = "http://api.openweathermap.org/data/2.5/weather?"

url = f"{BASE_URL}q={CITY}&appid={OPENWEATHER_API_KEY}&units=metric"

try:
    response = requests.get(url)
    response.raise_for_status() # Raises an HTTPError for bad responses (4xx or 5xx)
    data = response.json()

    print(f"--- Weather in {CITY} (Insecure Version) ---")
    print(f"Description: {data['weather'][0]['description'].capitalize()}")
    print(f"Temperature: {data['main'][0]['temp']}°C")
    print(f"Feels Like: {data['main'][0]['feels_like']}°C")
    print(f"Humidity: {data['main'][0]['humidity']}%")
    print(f"Wind Speed: {data['wind']['speed']} m/s")
except requests.exceptions.RequestException as e:
```

Version 2: Secure (API key via environment variables)

This approach uses environment variables to store and access the API key. In a Google Colab environment, `os.getenv` works by reading from the system's environment variables, which can be populated via `google.colab.userdata.get()` if you've stored your key in Colab's Secrets Manager.

To make this work in Colab:

1. Click the "⚡" icon on the left panel.
2. Click "+ New secret".
3. Name it `OPENWEATHER_API_KEY` (must match exactly).
4. Paste your actual API key into the "Value" field.
5. Ensure "Notebook access" is enabled for this secret.

If running outside Colab: Set the environment variable before running your script (e.g., `export OPENWEATHER_API_KEY='your_api_key'` in bash).

```
[3] 1s
import requests
import os
from google.colab import userdata # For Colab's secure secret access

# For Colab: Retrieve API key securely from Colab Secrets and set it as an environment variable
# In a production environment outside Colab, the environment variable would be set externally

# In a production environment outside Colab, the environment variable would be set externally
try:
    os.environ['OPENWEATHER_API_KEY'] = userdata.get('OPENWEATHER_API_KEY')
    print("API key loaded from Colab secrets.")
except Exception as e:
    print(f"Could not load API key from Colab secrets: {e}")
    print("Please ensure 'OPENWEATHER_API_KEY' is set in Colab's secrets and notebook access is enabled.")

# Retrieve API key from environment variables
# This is more secure as the key is not hardcoded in the script.
API_KEY = os.getenv("OPENWEATHER_API_KEY")

# Short comment: Using os.getenv() prevents the API key from being directly visible in the code
# This reduces the risk of accidental exposure if the code is shared publicly.

if not API_KEY:
    print("Error: OPENWEATHER_API_KEY environment variable is not set.")
    print("Please set the environment variable before running this script.")
else:
    CITY = "Paris" # Changed city for demonstration
    BASE_URL = "http://api.openweathermap.org/data/2.5/weather"
    url = f"{BASE_URL}q={CITY}&appid={API_KEY}&units=metric"

    trv:
```

```
try:
    response = requests.get(url)
    response.raise_for_status() # Raises an HTTPError for bad responses (4xx or 5xx)
    data = response.json()

    print(f"\n--- Weather in {CITY} (Secure Version) ---")
    print(f"Description: {data['weather'][0]['description'].capitalize()}")
    print(f"Temperature: {data['main']['temp']}°C")
    print(f"Feels Like: {data['main'][0]['feels_like']}°C")
    print(f"Humidity: {data['main'][0]['humidity']}%")
    print(f"Wind Speed: {data['wind'][0]['speed']} m/s")
except requests.exceptions.RequestException as e:
    print(f"Error fetching weather data: {e}")
except KeyError:
    print("Could not parse weather data. Check city name or API key validity.")

...
... Could not load API key from Colab secrets: Secret OPENWEATHER_API_KEY does not exist.
Please ensure 'OPENWEATHER_API_KEY' is set in colab's secrets and notebook access is enabled.
Error: OPENWEATHER_API_KEY environment variable is not set.
Please set the environment variable before running this script.

Start coding or generate with AI.
```

Prompt:

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

Task2:

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.

Expected Output:

- Identified privacy risks.
- Revised version with encrypted password storage (e.g., hashing).

Ai generated code:

```
import hashlib
import os
import getpass
import base64

USER_DATA_FILE = "user_data_secure.txt"

# -----
# Step 1: Hash the password with SHA-256 and salt
# -----
def hash_password(password: str, salt: bytes = None) -> str:
    if salt is None:
        salt = os.urandom(16) # 16-byte random salt
    pwd_hash = hashlib.pbkdf2_hmac('sha256', password.encode(), salt, 100_000)
    # store salt + hash in base64 for saving
    return base64.b64encode(salt + pwd_hash).decode()

# -----
# Step 2: Save user data
# -----
def save_user_data(name, email, password):
    hashed_password = hash_password(password)
    data_to_save = f"Name: {name}\nEmail: {email}\nPassword: {hashed_password}\n---\n"

    # append user data to file
    with open(USER_DATA_FILE, "a") as file:
        file.write(data_to_save)

print("[INFO] User data stored securely!")
```

```

68  # -----
69  # Step 3: Read all user data
70  # -----
71  def read_all_user_data():
72      if not os.path.exists(USER_DATA_FILE):
73          return "[INFO] No user data found."
74
75      with open(USER_DATA_FILE, "r") as file:
76          return file.read()
77
78  # -----
79  # Step 4: Collect user data
80  # -----
81  def main():
82      print("==> Secure User Registration ==>")
83      name = input("Enter your name: ")
84      email = input("Enter your email: ")
85      password = getpass.getpass("Enter your password: ")
86
87      save_user_data(name, email, password)
88
89      print("\n==> Stored Content for Verification ==>")
90      print(read_all_user_data())
91
92
93 if __name__ == "__main__":
94     main()
95

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
 === Stored Content for Verification ===
 Name: udayan
 Email: udayanduddukuri@gmail.com
 Password: xnPKsE2Wx+zSynz+ZRTPM5bh6R2JVsro3BzD9UlKYRgzpluchtnXPNrtFsYPh5n

Prompt:

“Generate a python code that stores the user data like(name,email and password)”

Task3:

Task Description #3 (Transparency in Algorithm Design)

Objective: Use AI to generate an Armstrong number checking function with comments and explanations.

Instructions:

1. Ask AI to explain the code line-by-line.
2. Compare the explanation with code functionality.

Expected Output:

- Transparent, commented code.
- Correct, easy-to-understand explanation.

Ai generated code:

```
97 def is_armstrong(number):
103     Example: 153 -> 1^3 + 5^3 + 3^3 = 153
104     """
105
106     # Convert the number to string to easily iterate through digits
107     digits = str(number)
108     num_digits = len(digits) # Number of digits in the number
109
110     # Initialize sum to store sum of digits raised to power of num_digits
111     sum_of_powers = 0
112
113     # Loop through each digit
114     for digit in digits:
115         sum_of_powers += int(digit) ** num_digits # Raise digit to power and add
116
117     # Compare calculated sum with the original number
118     if sum_of_powers == number:
119         return True # It's an Armstrong number
120     else:
121         return False # Not an Armstrong number
122
123 # -----
124 # Example Usage
125 #
126 num = int(input("Enter a number to check if it's an Armstrong number: "))
127 if is_armstrong(num):
128     print(f"{num} is an Armstrong number.")
129 else:
130     print(f"{num} is NOT an Armstrong number.")
131
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/Users/MYPC/OneDrive/Desktop/ai assisted coding/ai_ass.5.1.py"
153 is an Armstrong number.
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding> & C:\Users\MYPC\AppData\Local\Programs\Python\Python313\python.exe
/Users/MYPC/OneDrive/Desktop/ai assisted coding/ai_ass.5.1.py"
Enter a number to check if it's an Armstrong number: 234
234 is NOT an Armstrong number.
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding>
```

Prompt:

“Generate an python code for Armstrong number checking function”

Task4:

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."

Expected Output:

- Code for both algorithms.
- Transparent, comparative explanation of their logic and efficiency.

Ai generated code:

```
def bubble_sort(arr):
    """
    Bubble Sort repeatedly compares adjacent elements and swaps them
    if they are in the wrong order. This process is repeated until
    the array is sorted.

    Time Complexity: O(n^2) average and worst case
    Space Complexity: O(1) (in-place)
    Stable: Yes
    """
    n = len(arr)
    for i in range(n):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Compare adjacent elements
            if arr[j] > arr[j+1]:
                # Swap if elements are in wrong order
                arr[j], arr[j+1] = arr[j+1], arr[j]
                # Debug: print current array state after swap
                # print(f"Swapped {arr[j+1]} and {arr[j]}: {arr}")
    return arr

# =====
# Quick Sort Implementation
# =====

def quick_sort(arr):
    """
    QuickSort is a divide-and-conquer algorithm. It works by:
    1. Choosing a 'pivot' element.
    2. Partitioning the array into elements less than pivot (left)
       and greater than pivot (right).
    """

```

```

163     def quick_sort(arr):
164         """
165             Stable: No
166             """
167         if len(arr) <= 1:
168             return arr # Base case: already sorted
169
170         pivot = arr[len(arr) // 2] # Choose middle element as pivot
171         left = [x for x in arr if x < pivot] # Elements less than pivot
172         middle = [x for x in arr if x == pivot] # Elements equal to pivot
173         right = [x for x in arr if x > pivot] # Elements greater than pivot
174
175         # Recursively sort left and right, then combine
176         return quick_sort(left) + middle + quick_sort(right)
177
178
179
180
181
182
183
184
185
186
187     # =====
188     # Example usage and comparison
189     # =====
190
191     arr = [64, 34, 25, 12, 22, 11, 90]
192
193     print("Original Array:", arr)
194
195     # Using Bubble Sort
196     bubble_sorted = bubble_sort(arr.copy())
197     print("Bubble Sorted:", bubble_sorted)
198
199     # Using Quick Sort
200     quick_sorted = quick_sort(arr.copy())
201     print("Quick Sorted:", quick_sorted)
202

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

234 is NOT an Armstrong number.
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding> & C:\Users\MYPC\AppData\Local\Programs\Python\Python311\python.exe "C:/Users/MYPC/OneDrive/Desktop/ai assisted coding/ai_ass.5.1.py"
Original Array: [64, 34, 25, 12, 22, 11, 90]
Bubble Sorted: [11, 12, 22, 25, 34, 64, 90]
Quick Sorted: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding>

```

Prompt:

Generate Python code for QuickSort and BubbleSort, and include comments explaining step-by-step how each works and where they differ."

Tack5:

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."

Expected Output:

- Code with explainable recommendations.
- Evaluation of whether explanations are understandable.

Ai generated code:

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

products = pd.DataFrame({
    "product_id": [1, 2, 3, 4, 5],
    "product_name": [
        "Red Cotton Shirt",
        "Blue Denim Jeans",
        "Black Leather Shoes",
        "White Cotton Hat",
        "Green Winter Jacket"
    ],
    "category": ["Clothing", "Clothing", "Footwear", "Accessories", "Clothing"],
    "features": [
        "red cotton casual",
        "blue denim casual",
        "black leather formal",
        "white cotton casual",
        "green warm winter jacket"
    ]
})

user_preference = "I like casual cotton clothes"

vectorizer = TfidfVectorizer()
product_vectors = vectorizer.fit_transform(products["features"])
user_vector = vectorizer.transform([user_preference])

similarity_scores = cosine_similarity(user_vector, product_vectors).flatten()
products["similarity_score"] = similarity_scores

recommended_products = products.sort_values(
    by="similarity_score", ascending=False
)

print("\n== TRANSPARENT PRODUCT RECOMMENDATIONS ==\n")

for _, row in recommended_products.iterrows():
    print(f"Product: {row['product_name']}")

241 |     print(f"Product: {row['product_name']}")
242 |     print(f"Category: {row['category']}")
243 |     print(f"Similarity Score: {row['similarity_score']:.2f}")
244 |     print(
245 |         f"Reason: This product was recommended because its features "
246 |         f"({row['features']}) closely match your preference "
247 |         f"({user_preference})."
248 |     )
249 |     print("-" * 60)
250 |
251 print("\n== EXPLANATION EVALUATION ==\n")
252
253 evaluation_points = [
254     "Explanations use simple language",
255     "Product features are explicitly shown",
256     "User preference is clearly referenced",
257     "Similarity score supports transparency"
258 ]
259
260
261 for point in evaluation_points:
262     print(f"✓ {point}")
263
264 print(
265     "\nConclusion:\n"
266     "The explanations are understandable because they clearly connect "
267     "user preferences with product features using similarity scores, "
268     "making the recommendations transparent and trustworthy."
269 )

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

== TRANSPARENT PRODUCT RECOMMENDATIONS ==

Product: Red Cotton Shirt
Category: Clothing
Similarity Score: 0.72
Reason: This product was recommended because its features ('red cotton casual') closely match your preference ('I like casual cotton clothes').
-----
Product: White Cotton Hat
Category: Accessories
```

Prompt:

“Generate a recommendation system that also provides reasons for each suggestion”