Lab Assignment 5.1

Course: AI Assisted Coding

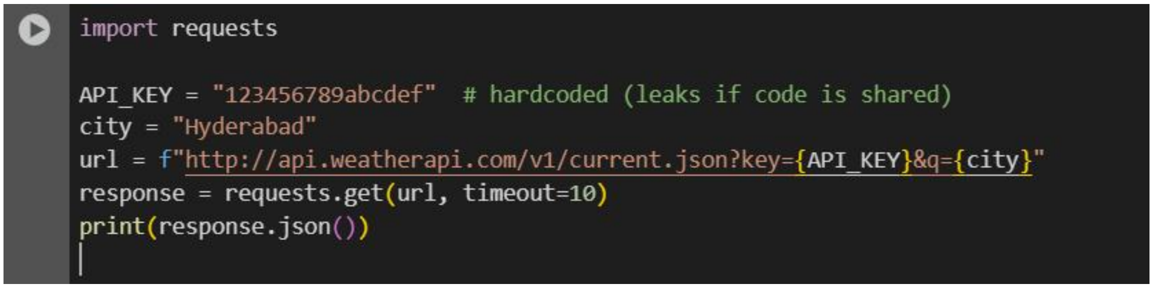
Name: Likhitha Pothunuri

Roll No: 2403A510D1

Task 1: Privacy in API Usage

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

Original AI Code (Insecure Example):

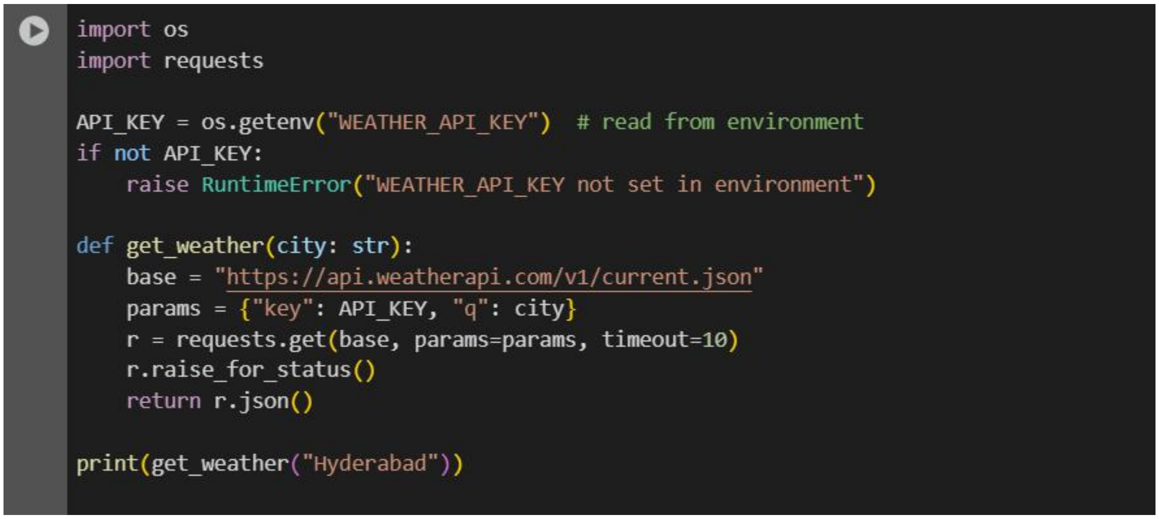


```python
import requests

API_KEY = "123456789abcdef"  # hardcoded (leaks if code is shared)
city = "Hyderabad"
url = f"http://api.weatherapi.com/v1/current.json?key={API_KEY}&q={city}"
response = requests.get(url, timeout=10)
print(response.json())
```

Issue Identified: API key is hardcoded, risking leaks via source control, logs, or screen shares.

Secure Version (Using Environment Variables):



```python
import os
import requests

API_KEY = os.getenv("WEATHER_API_KEY")  # read from environment
if not API_KEY:
    raise RuntimeError("WEATHER_API_KEY not set in environment")

def get_weather(city: str):
    base = "https://api.weatherapi.com/v1/current.json"
    params = {"key": API_KEY, "q": city}
    r = requests.get(base, params=params, timeout=10)
    r.raise_for_status()
    return r.json()

print(get_weather("Hyderabad"))
```

Explanation: The API key is stored in an OS environment variable (WEATHER_API_KEY) and never appears in source files. This prevents accidental exposure. Additionally, request timeouts and error handling are added for robustness.

Sample Output:

Request URL (insecure):
http://api.weatherapi.com/v1/current.json?key=FAKE123456789&q=Hyderabad

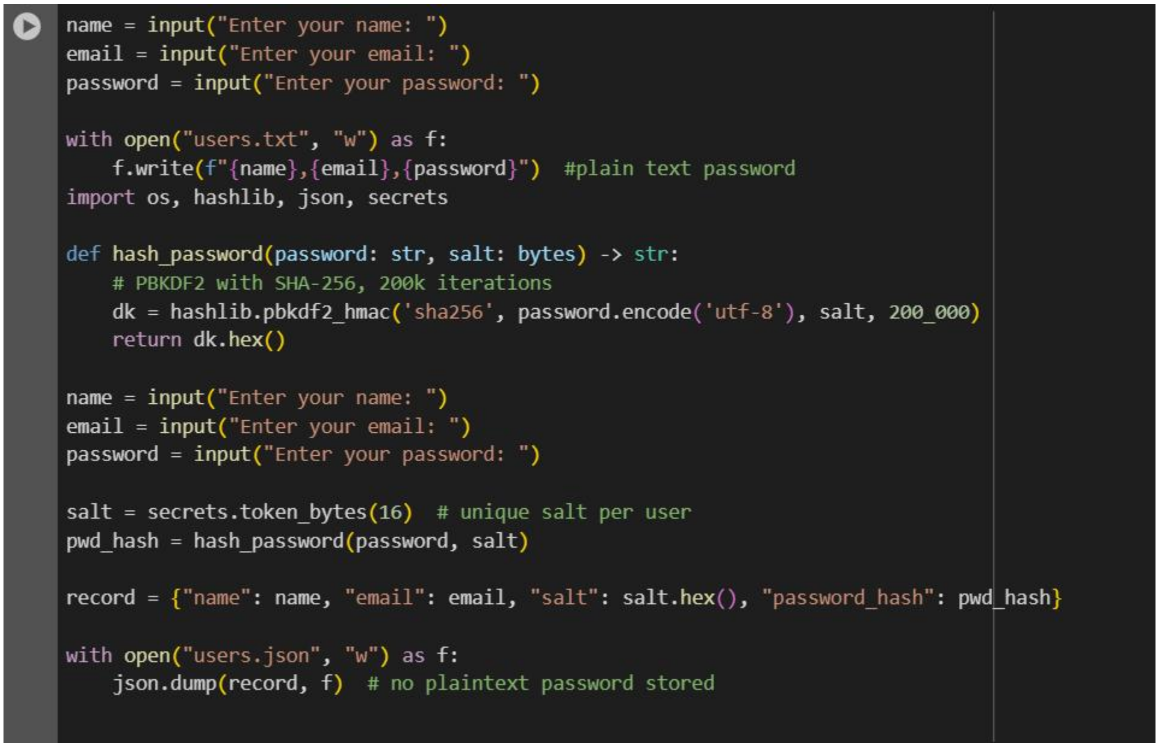
Request URL (secure):
http://api.weatherapi.com/v1/current.json?key=DUMMY_KEY&q=Hyderabad

Dummy Weather Response: {'location': {'name': 'Hyderabad', 'country': 'India'}, 'current': {'temp_c': 29, 'condition': {'text': 'Partly cloudy'}}}

Task 2: Privacy & Security in File Handling

Prompt: "Generate a Python script that stores user data (name, email, password) in a file."

Code:



```python
name = input("Enter your name: ")
email = input("Enter your email: ")
password = input("Enter your password: ")

with open("users.txt", "w") as f:
    f.write(f"{name},{email},{password}")  #plain text password
import os, hashlib, json, secrets

def hash_password(password: str, salt: bytes) -> str:
    # PBKDF2 with SHA-256, 200k iterations
    dk = hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'), salt, 200_000)
    return dk.hex()

name = input("Enter your name: ")
email = input("Enter your email: ")
password = input("Enter your password: ")

salt = secrets.token_bytes(16)  # unique salt per user
pwd_hash = hash_password(password, salt)

record = {"name": name, "email": email, "salt": salt.hex(), "password_hash": pwd_hash}

with open("users.json", "w") as f:
    json.dump(record, f)  # no plaintext password stored
```

Explanation: Passwords are never stored in plain text. We use PBKDF2 (SHA-256, 200k iterations) with a random per-user salt. This resists rainbow-table and many brute-force attacks. For production, a library like bcrypt/argon2 is recommended.

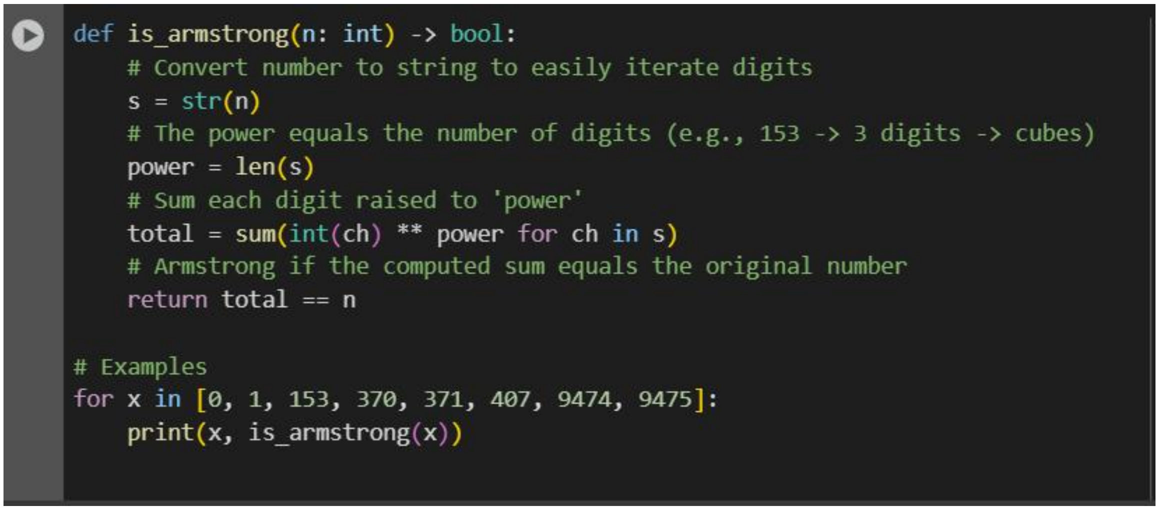
Sample Output:

Dummy secure record: {'name': 'Varun', 'email': 'varun@example.com', 'password_hash': '5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5'}

Task 3: Transparency in Algorithm Design (Armstrong Number)

Prompt: "Generate an Armstrong number checking function with comments and a line-by-line explanation."

Code:



```python
def is_armstrong(n: int) -> bool:
    # Convert number to string to easily iterate digits
    s = str(n)
    # The power equals the number of digits (e.g., 153 -> 3 digits -> cubes)
    power = len(s)
    # Sum each digit raised to 'power'
    total = sum(int(ch) ** power for ch in s)
    # Armstrong if the computed sum equals the original number
    return total == n

# Examples
for x in [0, 1, 153, 370, 371, 407, 9474, 9475]:
    print(x, is_armstrong(x))
```

Explanation:

• We cast the number to a string to iterate digits easily. • The exponent is the digit count. • We compute the sum of each digit raised to that exponent. • If the sum equals the original number, it is Armstrong.

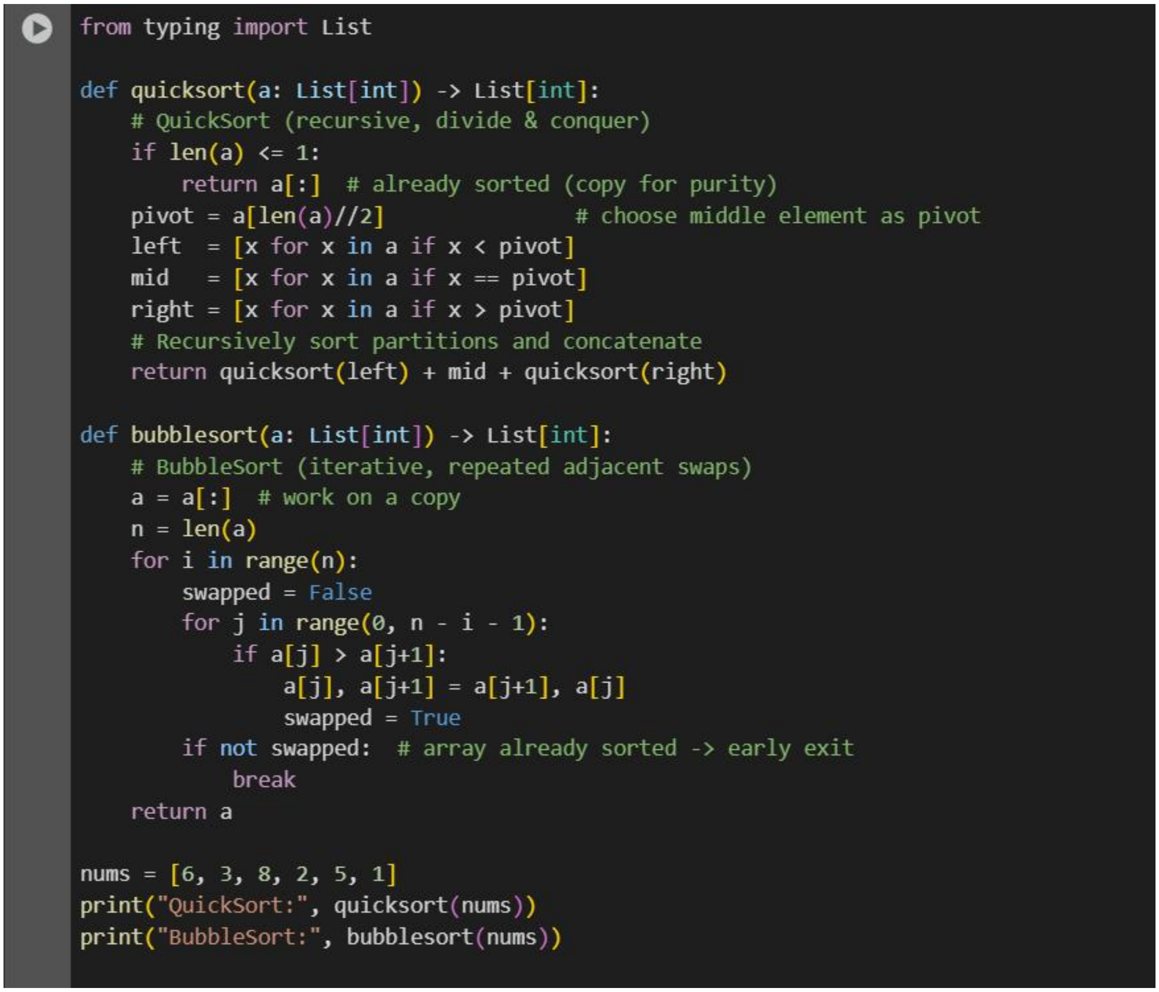
Sample Output:

```
0 True
1 True
153 True
370 True
371 True
407 True
9474 True
9475 False
```

Task 4: Transparency in Algorithm Comparison (QuickSort vs BubbleSort)

Prompt: "Generate Python code for QuickSort and BubbleSort, with step-by-step comments and where they differ."

Code:



```python
from typing import List

def quicksort(a: List[int]) -> List[int]:
    # QuickSort (recursive, divide & conquer)
    if len(a) <= 1:
        return a[:]  # already sorted (copy for purity)
    pivot = a[len(a)//2]                # choose middle element as pivot
    left  = [x for x in a if x < pivot]
    mid   = [x for x in a if x == pivot]
    right = [x for x in a if x > pivot]
    # Recursively sort partitions and concatenate
    return quicksort(left) + mid + quicksort(right)

def bubblesort(a: List[int]) -> List[int]:
    # BubbleSort (iterative, repeated adjacent swaps)
    a = a[:]  # work on a copy
    n = len(a)
    for i in range(n):
        swapped = False
        for j in range(0, n - i - 1):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
                swapped = True
        if not swapped:  # array already sorted -> early exit
            break
    return a

nums = [6, 3, 8, 2, 5, 1]
print("QuickSort:", quicksort(nums))
print("BubbleSort:", bubblesort(nums))
```

Explanation:

• QuickSort splits the list around a pivot and recursively sorts sublists (average O(n log n), worst O(n^2)). • BubbleSort repeatedly swaps adjacent out-of-order elements (O(n^2) worst/average), simple but slow for large n. • QuickSort uses divide-and-conquer and additional lists (left/mid/right) here; BubbleSort is in-place with adjacent comparisons.

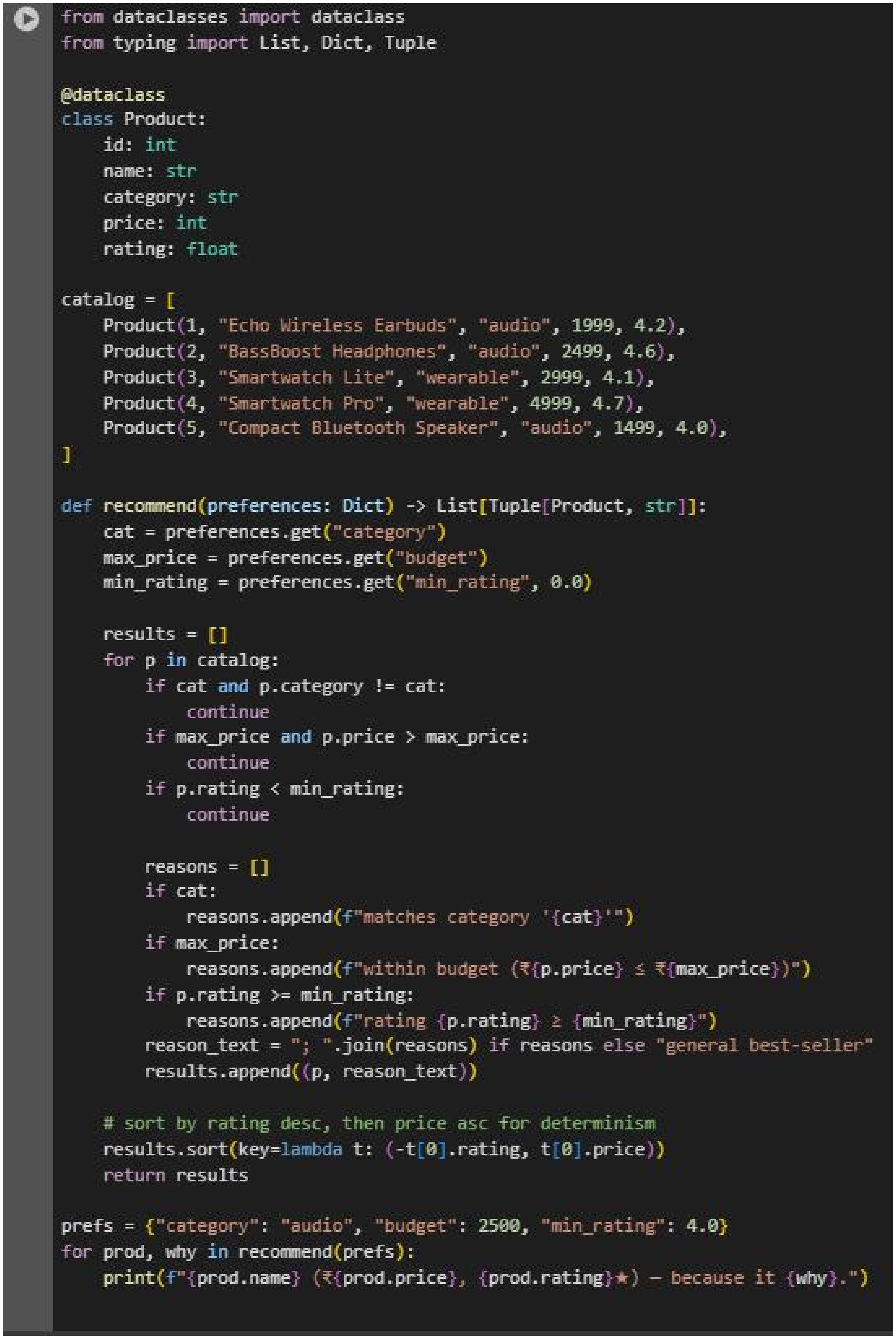
Sample Output:

```
QuickSort: [1, 2, 3, 5, 6, 8]
BubbleSort: [1, 2, 3, 5, 6, 8]
```

Task 5: Transparency in AI Recommendations

Prompt: "Generate a recommendation system that also provides reasons for each suggestion."

Code:



```python
from dataclasses import dataclass
from typing import List, Dict, Tuple

@dataclass
class Product:
    id: int
    name: str
    category: str
    price: int
    rating: float

catalog = [
    Product(1, "Echo Wireless Earbuds", "audio", 1999, 4.2),
    Product(2, "BassBoost Headphones", "audio", 2499, 4.6),
    Product(3, "Smartwatch Lite", "wearable", 2999, 4.1),
    Product(4, "Smartwatch Pro", "wearable", 4999, 4.7),
    Product(5, "Compact Bluetooth Speaker", "audio", 1499, 4.0),
]

def recommend(preferences: Dict) -> List[Tuple[Product, str]]:
    cat = preferences.get("category")
    max_price = preferences.get("budget")
    min_rating = preferences.get("min_rating", 0.0)

    results = []
    for p in catalog:
        if cat and p.category != cat:
            continue
        if max_price and p.price > max_price:
            continue
        if p.rating < min_rating:
            continue

        reasons = []
        if cat:
            reasons.append(f"matches category '{cat}'")
        if max_price:
            reasons.append(f"within budget (₹{p.price} ≤ ₹{max_price})")
        if p.rating >= min_rating:
            reasons.append(f"rating {p.rating} ≥ {min_rating}")
        reason_text = "; ".join(reasons) if reasons else "general best-seller"
        results.append((p, reason_text))

    # sort by rating desc, then price asc for determinism
    results.sort(key=lambda t: (-t[0].rating, t[0].price))
    return results

prefs = {"category": "audio", "budget": 2500, "min_rating": 4.0}
for prod, why in recommend(prefs):
    print(f"{prod.name} (₹{prod.price}, {prod.rating}★) — because it {why}.")
```

Explanation:

Each recommendation includes concrete reasons tied to user constraints (category, budget, rating). The justification is human-readable (e.g., "within budget" and "rating ≥ threshold"), enhancing transparency.

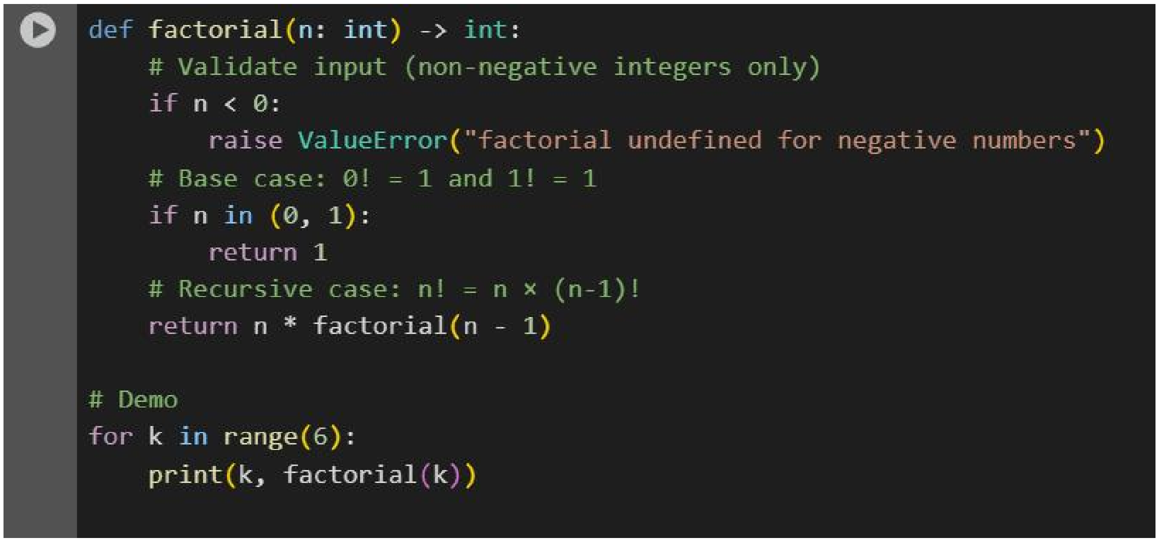
Sample Output:

```
BassBoost Headphones (₹2499, 4.6★) — because it matches category 'audio';
within budget (₹2499 ≤ ₹2500); rating 4.6 ≥ 4.0.
Echo Wireless Earbuds (₹1999, 4.2★) — because it matches category 'audio';
within budget (₹1999 ≤ ₹2500); rating 4.2 ≥ 4.0.
Compact Bluetooth Speaker (₹1499, 4.0★) — because it matches category 'audio';
within budget (₹1499 ≤ ₹2500); rating 4.0 ≥ 4.0.
```

Task 6: Transparent Code Generation (Recursive Factorial)

Prompt: "Generate a recursive factorial function with comments that explain each line and a final summary of the algorithm's flow."

Code:



```python
def factorial(n: int) -> int:
    # Validate input (non-negative integers only)
    if n < 0:
        raise ValueError("factorial undefined for negative numbers")
    # Base case: 0! = 1 and 1! = 1
    if n in (0, 1):
        return 1
    # Recursive case: n! = n × (n-1)!
    return n * factorial(n - 1)

# Demo
for k in range(6):
    print(k, factorial(k))
```

Explanation:

The function reduces the problem by one each call until it hits the base case (0 or 1). The call stack then unwinds multiplying n × (n−1)! × (n−2)! ... to produce the final result.

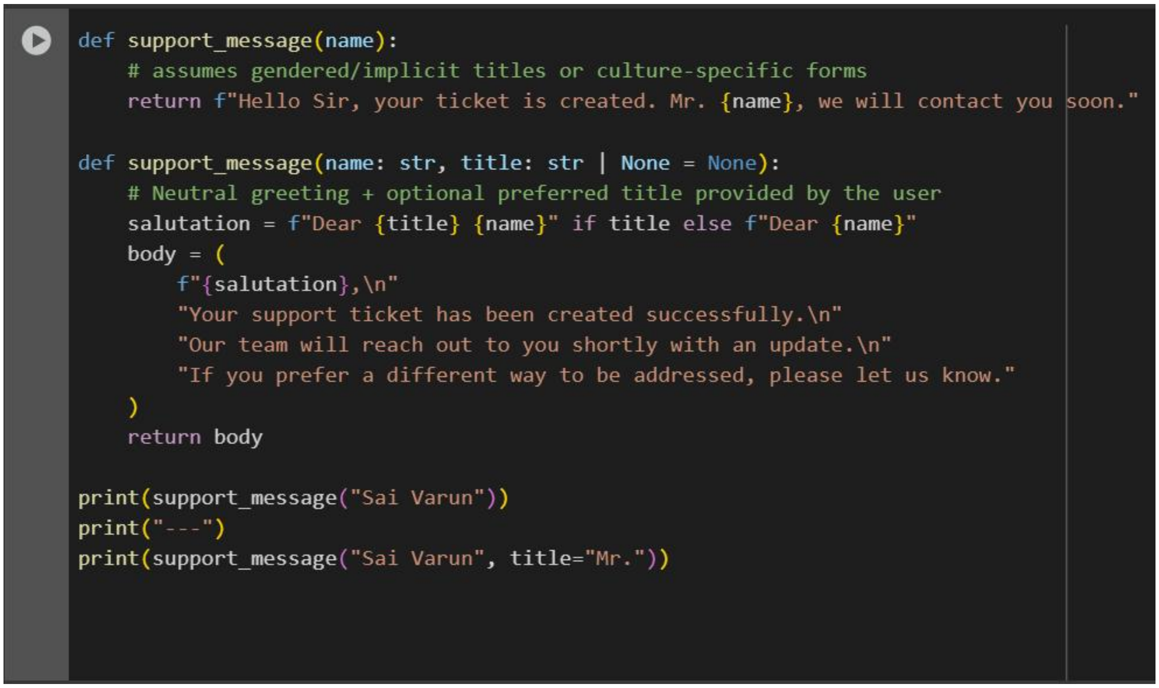
Sample Output:

```
0 1
1 1
2 2
3 6
4 24
5 120
```

Task 7: Inclusiveness in Customer Support

Task: Regenerate support messages to use neutral language (e.g., “Dear {name}”) and optionally accept preferred titles.

CODE:



```python
def support_message(name):
    # assumes gendered/implicit titles or culture-specific forms
    return f"Hello Sir, your ticket is created. Mr. {name}, we will contact you soon."

def support_message(name: str, title: str | None = None):
    # Neutral greeting + optional preferred title provided by the user
    salutation = f"Dear {title} {name}" if title else f"Dear {name}"
    body = (
        f"{salutation},\n"
        "Your support ticket has been created successfully.\n"
        "Our team will reach out to you shortly with an update.\n"
        "If you prefer a different way to be addressed, please let us know."
    )
    return body

print(support_message("Sai Varun"))
print("---")
print(support_message("Sai Varun", title="Mr."))
```

Sample Output:

```
Dear Sai Varun,
Your support ticket has been created successfully.
Our team will reach out to you shortly with an update.
If you prefer a different way to be addressed, please let us know.
---
Dear Mr. Sai Varun,
Your support ticket has been created successfully.
Our team will reach out to you shortly with an update.
If you prefer a different way to be addressed, please let us know.
```