

AI Assisted Coding LAB ASSIGNMENT-5.4

NAME: D.Nithin

Batch:13

Hall Ticket:2303A51845

Task Description #1:

- Prompt GitHub Copilot to generate a Python script that collects user data (e.g., name, age, email). Then, ask Copilot to add comments on how to anonymize or protect this data.

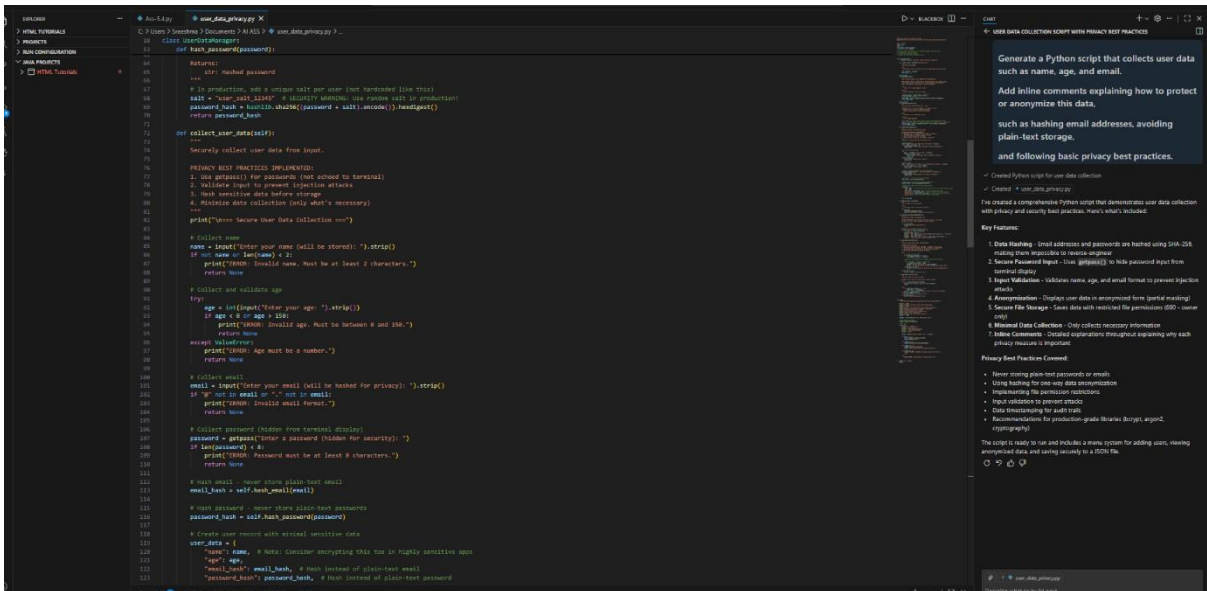
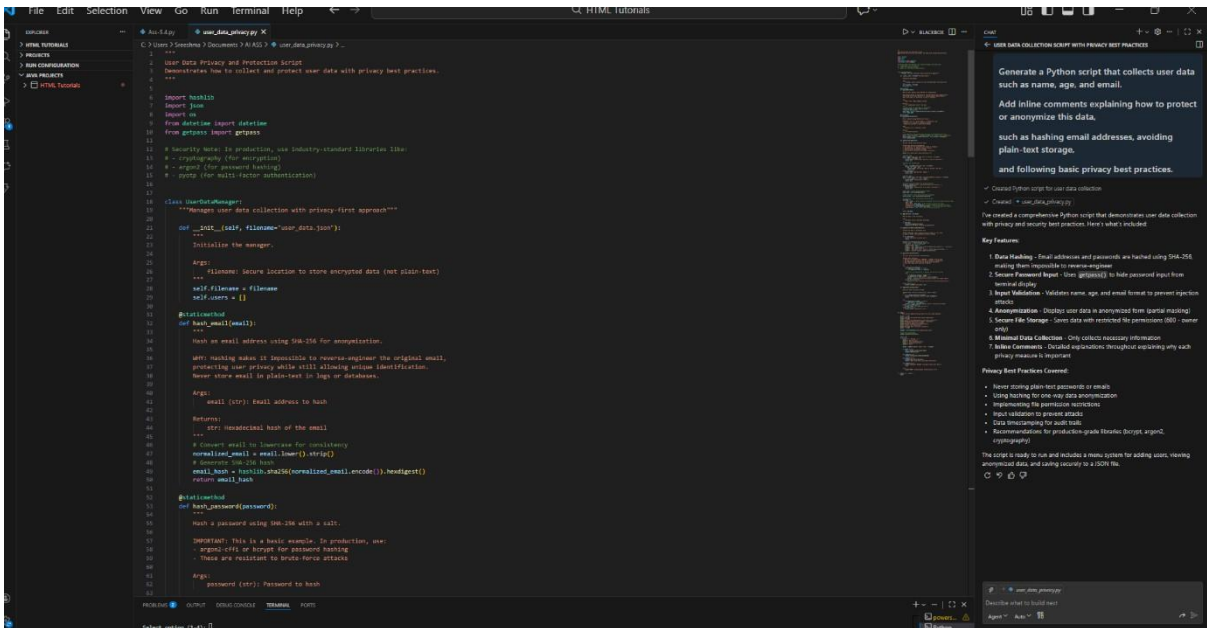
PROMPT

Generate a Python script that collects user data such as name, age, and email.

Add inline comments explaining how to protect or anonymize this data,

such as hashing email addresses, avoiding plain-text storage,

and following basic privacy best practices.

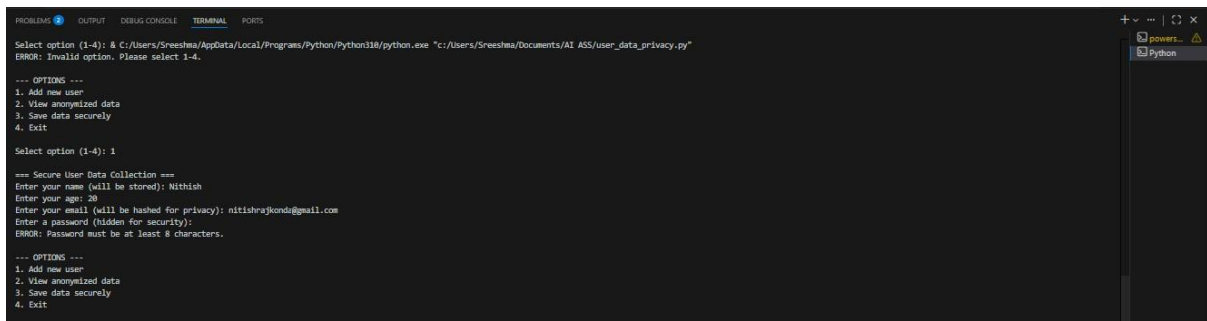


```
1 # user_data_privacy.py
2 """
3 A Python script that collects user data such as name, age, and email.
4 Add inline comments explaining how to protect or anonymize this data,
5 such as hashing email addresses, avoiding plain-text storage,
6 and following basic privacy best practices.
7 """
8
9 # Create a Python script for user data collection
10 # Create a user_data_privacy.py
11
12 # I've created a comprehensive Python script that demonstrates user data collection
13 # with privacy and security best practices. Here's what's included:
14
15 # Key Features:
16
17 1. Data Hashing - Email addresses and passwords are hashed using SHA-256,
18 making them irreversible to prevent exposure.
19 2. Secure Password Input - Uses getpass() to hide password input from
20 terminal display.
21 3. Input Validation - Validates name, age, and email format to prevent injection
22 attacks.
23 4. Anonymization - Displays user data in anonymized form (partial masking).
24 5. Secure File Storage - Saves user data with restricted file permissions (0600 - owner
25 only).
26 6. Minimal Data Collection - Only collects necessary information.
27 7. Future Consistency - Detailed explanations throughout explaining why each
28 privacy measure is important.
29
30 # Privacy Best Practices Covered:
31
32 • Never storing plain-text passwords or emails
33 • Using hashing for one-way data anonymization
34 • Implementing the permission restrictions
35 • Input validation to prevent attacks
36 • Data anonymization for audit trails
37 • Recommendations for production-grade libraries (bcrypt, argon2,
38 cryptography)
39
40 The script is ready to run and includes a menu system for adding users, viewing
41 anonymized data, and saving securely to a JSON file.
```

```
1 # user_data_privacy.py
2 """
3 A Python script that collects user data such as name, age, and email.
4 Add inline comments explaining how to protect or anonymize this data,
5 such as hashing email addresses, avoiding plain-text storage,
6 and following basic privacy best practices.
7 """
8
9 # Create a Python script for user data collection
10 # Create a user_data_privacy.py
11
12 # I've created a comprehensive Python script that demonstrates user data collection
13 # with privacy and security best practices. Here's what's included:
14
15 # Key Features:
16
17 1. Data Hashing - Email addresses and passwords are hashed using SHA-256,
18 making them irreversible to prevent exposure.
19 2. Secure Password Input - Uses getpass() to hide password input from
20 terminal display.
21 3. Input Validation - Validates name, age, and email format to prevent injection
22 attacks.
23 4. Anonymization - Displays user data in anonymized form (partial masking).
24 5. Secure File Storage - Saves user data with restricted file permissions (0600 - owner
25 only).
26 6. Minimal Data Collection - Only collects necessary information.
27 7. Future Consistency - Detailed explanations throughout explaining why each
28 privacy measure is important.
29
30 # Privacy Best Practices Covered:
31
32 • Never storing plain-text passwords or emails
33 • Using hashing for one-way data anonymization
34 • Implementing the permission restrictions
35 • Input validation to prevent attacks
36 • Data anonymization for audit trails
37 • Recommendations for production-grade libraries (bcrypt, argon2,
38 cryptography)
39
40 The script is ready to run and includes a menu system for adding users, viewing
41 anonymized data, and saving securely to a JSON file.
```

Expected Output #1:

- A script with inline Copilot-suggested code and comments explaining how to safeguard or anonymize user information (e.g., hashing emails, not storing data unencrypted).



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Select option (1-4): & C:/Users/Sreeshma/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Sreeshma/Documents/AI ASS/user_data_privacy.py"
ERROR: Invalid option. Please select 1-4.

--- OPTIONS ---
1. Add new user
2. View anonymized data
3. Save data securely
4. Exit

Select option (1-4): 1

=== Secure User Data Collection ===
Enter your name (will be stored): Nithish
Enter your age: 20
Enter your email (will be hashed for privacy): nitishrajkond@gmail.com
Enter a password (hidden for security):
ERROR: Password must be at least 8 characters.

--- OPTIONS ---
1. Add new user
2. View anonymized data
3. Save data securely
4. Exit
```

Task Description #2:

- Ask Copilot to generate a Python function for sentiment analysis.

Then prompt Copilot to identify and handle potential biases in the data.

PROMPT: # Generate a Python function for sentiment analysis.

Add comments or code to identify and reduce potential biases in the data,

such as removing offensive terms, balancing positive and negative samples,

and avoiding biased language in predictions.

Task Description #3:

- Use Copilot to write a Python program that recommends products based on user history. Ask it to follow ethical guidelines

like transparency and fairness

PROMPT: # Generate a Python program that recommends products based on user purchase history.

Follow ethical AI guidelines such as transparency, fairness, and user control.

Add comments explaining how recommendations are generated,

avoid favoritism toward only popular products,

and allow users to give feedback or opt out of recommendations.

```
class RecommendationSystem:
    """Product recommendation with fairness and user control"""

    def __init__(self):
        self.user_purchases = {} # {user_id: [products]}
        self.products = {} # {product_id: [category]}
        self.user_opt_out = {} # {user_id: bool}
        self.feedback = {} # {user_id: float}

    def add_purchase(self, user_id, product_id, category):
        """Record a user purchase"""
        if user_id not in self.user_purchases:
            self.user_purchases[user_id] = []
        self.user_purchases[user_id].append(product_id)
        self.products[product_id] = {'category': category, 'count': 0}

    def recommend(self, user_id, num=5):
        """
        TRANSPARENCY: show why each recommendation is made
        FAIRNESS: don't only recommend popular products
        USER CONTROL: respect opt-out preference
        """
        # ETHICAL CHECK: Respect user opt-out
        if user_id in self.user_opt_out:
            return {'status': 'user opted out', 'recommendations': []}

        if user_id not in self.user_purchases:
            return {'status': 'New user', 'recommendations': []}

        user_history = self.user_purchases[user_id]
        user_categories = [self.products[p].get('category') for p in user_history if p in self.products]

        # Find candidates
        candidates = []
        for prod_id, prod_data in self.products.items():
            if prod_id not in user_history: # skip already purchased
                category = prod_data.get('category')

                # FAIRNESS: Score based on relevance + diversity
                if category in user_categories:
                    score = 0.5 # Bias toward user's interests
                else:
                    score = 0.5 # Explore new category

                # TRANSPARENCY: Explain why
                reason = f"Similar to your {category} purchases" if category in user_categories else f"Try new: {category}"

                candidates.append({
                    'product': prod_id,
                    'score': score,
                    'reason': reason
                })

        # Sort by score and return top N
        top_recs = sorted(candidates, key=lambda x: x['score'], reverse=True)[:num]

        return {
            'status': 'Success',
            'user_id': user_id,
            'history': user_history,
            'recommendations': top_recs
        }
```

```

1 # ethical_recommendation_system.py
2 """
3     Recommendation System
4 """
5
6 class RecommendationSystem:
7     def __init__(self, user_id, name):
8         self.user_id = user_id
9         self.name = name
10        self.history = []
11        self.recommendations = []
12
13    def recommend(self, user_id, name):
14        """
15            Recommend products to the user based on their history.
16            Returns a list of recommended products.
17        """
18        return self.recommendations
19
20    def give_feedback(self, user_id, product_id, liked):
21        """
22            Give feedback on a product.
23            Returns a message.
24        """
25        self.history.append(product_id)
26        return f"Thanks for feedback on {product_id}"
27
28    def opt_out(self, user_id):
29        """
30            Opt out of recommendations.
31            Returns a message.
32        """
33        return f"{user_id} opted out of recommendations"
34
35    def opt_in(self, user_id):
36        """
37            Opt in to recommendations.
38            Returns a message.
39        """
40        return f"{user_id} opted in to recommendations"
41
42 # Example Usage
43 if __name__ == "__main__":
44     system = RecommendationSystem(1, "John")
45
46     # Add purchases
47     print("Adding purchases")
48     system.add_purchase("user1", "laptop", "electronics")
49     system.add_purchase("user1", "mouse", "electronics")
50     system.add_purchase("user1", "phone", "phone")
51     print(f"Purchases recorded")
52
53     # Add products
54     system.products["keyboard"] = {"category": "electronics"}
55     system.products["monitor"] = {"category": "electronics"}
56     system.products["headset"] = {"category": "phone"}
57
58     # Get recommendations
59     print("Recommendations for user1")
60     result = system.recommend("user1", name=1)
61     for rec in result["recommendations"]:
62         print(f"Product: {rec['product']}, Score: {rec['score']}, Reason: {rec['reason']}")
63
64     # User feedback
65     print("Give user feedback")
66     system.give_feedback("user1", "keyboard", True)
67
68     # Opt out
69     print("User Control")
70     print(system.opt_out("user1"))
71     result2 = system.recommend("user1")
72     print(f"User Control: {result2}")
73
74     # Opt in
75     print(system.opt_in("user1"))

```

Expected Output #3:

- Copilot suggestions that include explanations, fairness checks

(e.g., avoiding favoritism), and user feedback options in the code.

```

-- Adding Purchases --
# Purchases recorded
PS C:\Users\joshua\Downloads\WPN_Tutorials> C:\Users\joshua\AppData\Local\Programs\Python\Python310\python.exe "C:\Users\joshua\Documents\AI_ASE\ethical_recommendation_system.py"
-- Adding Purchases --
# Purchases recorded
PS C:\Users\joshua\Downloads\WPN_Tutorials> C:\Users\joshua\AppData\Local\Programs\Python\Python310\python.exe "C:\Users\joshua\Documents\AI_ASE\ethical_recommendation_system.py"
-- Adding Purchases --
# Purchases recorded
PS C:\Users\joshua\Downloads\WPN_Tutorials>

-- Recommendations for user1 --
Product: keyboard, Score: 0.8, Reason: Similar to your Electronics purchases
Product: monitor, Score: 0.8, Reason: Similar to your Electronics purchases

-- User feedback --
Thanks for feedback on keyboard

-- User Control --
User1 opted out of recommendations
After opt-out: User1 opted out
User1 opted in to recommendations
PS C:\Users\joshua\Downloads\WPN_Tutorials>

```

Task Description #4:

- Prompt Copilot to generate logging functionality in a Python web

application. Then, ask it to ensure the logs do not record sensitive information.

PROMPT: # Generate logging functionality for a Python web application.

Ensure logs do NOT store sensitive information such as passwords,

emails, or personal identifiers.

Add comments explaining ethical logging practices and privacy protection.

```
1 #!/usr/bin/env python3
2 """Simple ethical logging for web applications"""
3
4 import logging
5
6
7 class PrivacyFilter(logging.Filter):
8     """Remove sensitive data from logs"""
9
10     def filter(self, record):
11         """Mask passwords, emails, tokens, cards before logging"""
12         msg = record.getMessage()
13
14         # PRIVACY: Mask passwords
15         msg = re.sub('password=.*', 'password=***REDACTED***', msg, flags=re.IGNORECASE)
16
17         # PRIVACY: Mask emails (show domain only)
18         msg = re.sub('[-.\w+@]{1,25}@[a-z0-9]{2,5}\.([a-z]{2,5})', '[REDACTED]', msg)
19
20         # PRIVACY: Mask API keys and tokens
21         msg = re.sub('([a-zA-Z0-9]{32}|[a-zA-Z0-9]{40}|[a-zA-Z0-9]{64})', '[REDACTED]', msg, flags=re.IGNORECASE)
22
23         # PRIVACY: Mask credit cards (show last 4 digits)
24         msg = re.sub('(?:[0-9]{4}-){3}[0-9]{4}', '[REDACTED]', msg)
25
26         # PRIVACY: Mask phone numbers (show last 4 digits)
27         msg = re.sub('(\d{3}|\d{4})-[0-9]{3,4}', '[REDACTED]', msg)
28
29         record.msg = msg
30         return True
31
32
33 def setup_logger(name, log_filename):
34     """Set up logger with privacy protection"""
35     logger = logging.getLogger(name)
36     logger.setLevel(logging.INFO)
37
38     # Add privacy filter
39     privacy_filter = PrivacyFilter()
40
41     # Console handler
42     console_handler = logging.StreamHandler()
43     console_handler.addFilter(privacy_filter)
44     formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
45     console_handler.setFormatter(formatter)
46     logger.addHandler(console_handler)
47
48     # File handler
49     if log_filename:
50         file_handler = logging.FileHandler(log_filename)
51         file_handler.addFilter(privacy_filter)
52         file_handler.setFormatter(formatter)
53         logger.addHandler(file_handler)
54
55     # PRIVACY: Restrict file permissions (never read/write only)
56     import os
57     os.chmod(log_filename, 0o600)
58
59     return logger
60
61
62 def log_user_action(logger, action, user_id, **safe_details):
63     """Log user action with only safe fields"""
64     msg = f'ACTION: {action} | user: {user_id}'
65     if safe_details:
66         msg += ' | ' + (safe_details)
67     logger.info(msg)
68
69
70 # Example usage
71 if __name__ == '__main__':
72     print("--- Simple Ethical Logging Demo ---")
73
74     logger = setup_logger("app", log_file="app.log")
75
76     print("Test 1: Password Masking")
77     logger.info("login with password=SecurePass123!")
78
79     print("Test 2: Email Masking")
80     logger.info("send email to user@example.com")
81
82     print("Test 3: API Key Masking")
83     logger.info("API key: sk_live_1234567890")
84
85     print("Test 4: Credit Card Masking")
86     logger.info("Payment with card 6011 1234 5678 9010")
87
88     print("Test 5: User Action Logging")
89     log_user_action(logger, "purchase", "user_123", status="success", amount=99.99)
90
91     print("\n" + "=" * 50)
92     print("ETHICAL LOGGING PRACTICES")
93     print("=" * 50)
94
95     # 1. PRIVACY FILTER: Mask passwords, emails, tokens, cards
96     # 2. MINIMAL DATA: Only log necessary information
97     # 3. SECURE FILES: Set permissions to 600 (never write only)
98     # 4. CLEAR ACTIONS: Log for auditing and debugging
99     # 5. NO SENSITIVE: Never store sensitive data in logs
100
101     print("\n")
```

```
102
103
104 def setup_logger(name, log_filename):
105     """Set up logger with privacy protection"""
106     logger = logging.getLogger(name)
107     os.chmod(log_filename, 0o600)
108
109     return logger
110
111
112 def log_user_action(logger, action, user_id, **safe_details):
113     """Log user action with only safe fields"""
114     msg = f'ACTION: {action} | user: {user_id}'
115     if safe_details:
116         msg += ' | ' + (safe_details)
117     logger.info(msg)
118
119
120 # Example usage
121 if __name__ == '__main__':
122     print("--- Simple Ethical Logging Demo ---")
123
124     logger = setup_logger("app", log_file="app.log")
125
126     print("Test 1: Password Masking")
127     logger.info("login with password=SecurePass123!")
128
129     print("Test 2: Email Masking")
130     logger.info("send email to user@example.com")
131
132     print("Test 3: API Key Masking")
133     logger.info("API key: sk_live_1234567890")
134
135     print("Test 4: Credit Card Masking")
136     logger.info("Payment with card 6011 1234 5678 9010")
137
138     print("Test 5: User Action Logging")
139     log_user_action(logger, "purchase", "user_123", status="success", amount=99.99)
140
141     print("\n" + "=" * 50)
142     print("ETHICAL LOGGING PRACTICES")
143     print("=" * 50)
144
145     # 1. PRIVACY FILTER: Mask passwords, emails, tokens, cards
146     # 2. MINIMAL DATA: Only log necessary information
147     # 3. SECURE FILES: Set permissions to 600 (never write only)
148     # 4. CLEAR ACTIONS: Log for auditing and debugging
149     # 5. NO SENSITIVE: Never store sensitive data in logs
150
151     print("\n")
```

Expected Output #4:

- Logging code that avoids saving personal identifiers (e.g., passwords, emails), and includes comments about ethical logging practices.

```

Python3 1/2/2024 10:00:00 AM 10.0.0.100 10.0.0.100
Test 5: User Action Logging
2024-01-20 10:20:15,606 - app - INFO - ACTION: purchase | user: user_123 | {'status': 'success', 'amount': 99.99}

-----
ETHICAL LOGGING PRACTICES:
-----
1. PRIVACY FILTER: Mask passwords, email, tokens, cards
2. MINIMAL DATA: Only log necessary information
3. SECURE FILES: Set permissions to 600 (owner only)
4. USER ACTIONS: Log for auditing and debugging
5. NO SENSITIVE: Never store sensitive data in logs
2024-01-20 10:20:15,606 - app - INFO - ACTION: purchase | user: user_123 | {'status': 'success', 'amount': 99.99}

-----
ETHICAL LOGGING PRACTICES:
-----
1. PRIVACY FILTER: Mask passwords, email, tokens, cards
2. MINIMAL DATA: Only log necessary information
3. SECURE FILES: Set permissions to 600 (owner only)
4. USER ACTIONS: Log for auditing and debugging
5. NO SENSITIVE: Never store sensitive data in logs

-----
1. PRIVACY FILTER: Mask passwords, email, tokens, cards
2. MINIMAL DATA: Only log necessary information
3. SECURE FILES: Set permissions to 600 (owner only)
4. USER ACTIONS: Log for auditing and debugging
5. NO SENSITIVE: Never store sensitive data in logs

```

Task Description #5:

- Ask Copilot to generate a machine learning model.

Then, prompt

it to add documentation on how to use the model responsibly (e.g., explainability, accuracy limits).

PROMPT: Generate a Python machine learning model (including data loading, training, and prediction steps).

Add inline documentation or a README-style comment section explaining how to use the model responsibly, including accuracy limitations, explainability considerations, fairness concerns, and appropriate use cases and restrictions.

