SR UNIVERSITY

Program	:B.tech(CSE)
Specialization	:AIML
Course Title	:AI Assisted Coding
Course Code	:24CS002PC215
Semester	:3rd semester
Academic Session	:2025-2026
Name of Student	:S ruchitha
Enrollment No.	:2403A52316
Batch No.	:01
Date	:24/11/2025

#LAB ASSIGNMENT-22.1

Task 1:

Provide the same coding task to an AI tool multiple times.

- Check if AI outputs **different results** depending on prompt phrasing.
- Discuss:
    - Are there hidden assumptions or biases in the responses?
    - How could biased AI code impact fairness or accessibility?

Prompt 1:

Write a Python function that takes a list of integers and returns a new list containing only the unique elements, maintaining their original order of appearance

Code:

```python
def get_unique_elements_ordered(input_list):
    seen = set()
    unique_list = []
    for item in input_list:
        if item not in seen:
            unique_list.append(item)
            seen.add(item)
    return unique_list

# Example usage:
my_list = [1, 2, 2, 3, 1, 4, 5, 3]
result = get_unique_elements_ordered(my_list)
print(f"Original list: {my_list}")
print(f"Unique elements (ordered): {result}")

my_list_2 = [5, 5, 5, 5, 5]
result_2 = get_unique_elements_ordered(my_list_2)
print(f"Original list: {my_list_2}")
print(f"Unique elements (ordered): {result_2}")
```

```
Original list: [1, 2, 2, 3, 1, 4, 5, 3]
Unique elements (ordered): [1, 2, 3, 4, 5]
Original list: [5, 5, 5, 5, 5]
Unique elements (ordered): [5]
```

Prompt 2:

generate a Python function that takes a list of numbers and returns a new list containing only the unique elements, maintaining their original order of appearance

code:

```python
# Assuming the 'get_unique_elements_ordered' function from cell '08ffb9af' is define

my_numbers = [10, 5, 20, 10, 30, 5, 20, 40, 50, 30, 10]
unique_ordered_numbers = get_unique_elements_ordered(my_numbers)

print(f"Original list: {my_numbers}")
print(f"Unique elements (ordered): {unique_ordered_numbers}")
```

```
Original list: [10, 5, 20, 10, 30, 5, 20, 40, 50, 30, 10]
Unique elements (ordered): [10, 5, 20, 30, 40, 50]
```

```python
def get_unique_elements_ordered(input_list):
    seen = set()
    unique_list = []
    for item in input_list:
        if item not in seen:
            unique_list.append(item)
            seen.add(item)
    return unique_list
```

Explanation:"one works directly while other works completely different I came to understand that giving good prompt makes code more clear and understandable."

Task 2 – Privacy and Sensitive Data Handling

- Ask AI to write a script that processes user personal data (e.g., names, phone numbers).
- Discuss:
    - How should sensitive data be anonymized?
    - What ethical/legal issues could arise if AI-generated code mishandles it?
    - What role does **developer responsibility** play here?

Prompt:

Generate script that processes user personal data (e.g., names, phone numbers)

Script:

Ethical/Legal Issues from Mishandling Sensitive Data

If AI-generated code, like the `process_user_data` script, were deployed without rigorous review and mishandled sensitive information, several significant ethical and legal issues could arise:

1. **Data Breaches and Security Incidents:**

*   **Issue:** Poorly secured or incorrectly processed data could lead to unauthorized access, disclosure, alteration, or destruction of personal data. This might be due to vulnerabilities introduced by the AI-generated code (e.g., inadequate input validation, insecure storage practices, or logging sensitive data).

*   **Consequences:** Reputational damage, loss of customer trust, financial penalties, and legal action from affected individuals.

2. **Incorrect Processing and Re-identification:**

*   **Issue:** If the AI-generated code fails to properly anonymize or pseudonymize data, or if it makes errors during processing (e.g., inadvertently combining datasets that allow re-identification), individuals could be linked back to their sensitive information.

*   **Consequences:** Violation of privacy rights, potential for discrimination, and legal liabilities.

3. **Lack of Transparency and Accountability:**

*   **Issue:** The 'black box' nature of some AI models can make it difficult to understand *why* certain code was generated or *how* it handles data. If an error occurs, pinpointing the cause and assigning accountability can be challenging.

*   **Consequences:** Erosion of trust in AI systems and difficulty in demonstrating compliance with data protection principles.

4. **Regulatory Non-Compliance:**

*   **Issue:** Mishandling of sensitive data directly violates various data protection regulations designed to safeguard individuals' privacy. Key regulations include:

    *   **GDPR (General Data Protection Regulation - EU):** Mandates strict rules for data collection, processing, storage, and transfer, emphasizing data minimization, purpose limitation, and accountability. Fines can be substantial.

    *   **CCPA (California Consumer Privacy Act - US):** Grants California consumers rights over their personal information, including the right to know, delete, and opt-out of the sale of personal data. Requires businesses to implement reasonable security procedures.

    *   **HIPAA (Health Insurance Portability and Accountability Act - US):** Specifically protects sensitive patient health information. Mishandling of medical data can lead to severe penalties.

*   **Consequences:** Massive fines, legal injunctions, mandatory reporting of breaches, and damage to business operations.

5. **Ethical Concerns (Bias and Discrimination):**

* **Issue:** If the AI model was trained on biased data or its generated code implicitly carries biases, it could lead to discriminatory outcomes. For example, if phone number formatting disproportionately fails for certain demographics, it could create unequal access or service.

* **Consequences:** Social injustice, ethical condemnation, and reputational harm.

Therefore, a thorough review process is not just good practice but a legal and ethical imperative when deploying AI-generated code that deals with personal data.

Task 3 – AI-Generated Security Risks

- Use AI to generate login/authentication code.
- Identify possible **security flaws** (e.g., weak password storage, lack of encryption).

Students must **document risks** and propose ethical guidelines for safe AI code usage

Prompt:

Write a basic Python script that implements user registration and login functionality. The script should:

1. Allow users to register with a username and password.
2. Store user credentials (e.g., in a simple dictionary in memory for this example).
3. Allow existing users to log in by verifying their username and password.
4. Provide appropriate messages for successful registration/login or failed attempts

Code:

```python
def login(username, password):
    if username not in users_db:
        print("Error: Username not found.")
        return False
    if users_db[username] == password:
        print(f"Welcome, {username}! You have successfully logged in.")
        return True
    else:
        print("Error: Incorrect password.")
        return False

# --- Example Usage --- #
print("\n--- Registration Attempts ---")
register("alice", "password123")
register("bob", "securepass")
register("alice", "newpassword") # Attempt to register existing user

print("\n--- Login Attempts ---")
login("alice", "password123") # Successful login
login("bob", "wrongpass")   # Incorrect password
login("charlie", "anypass") # Username not found
login("bob", "securepass")  # Successful login
```

```
--- Registration Attempts ---
User 'alice' registered successfully.
User 'bob' registered successfully.
Error: Username 'alice' already exists. Please choose another.

--- Login Attempts ---
Welcome, alice! You have successfully logged in.
```

Observation:

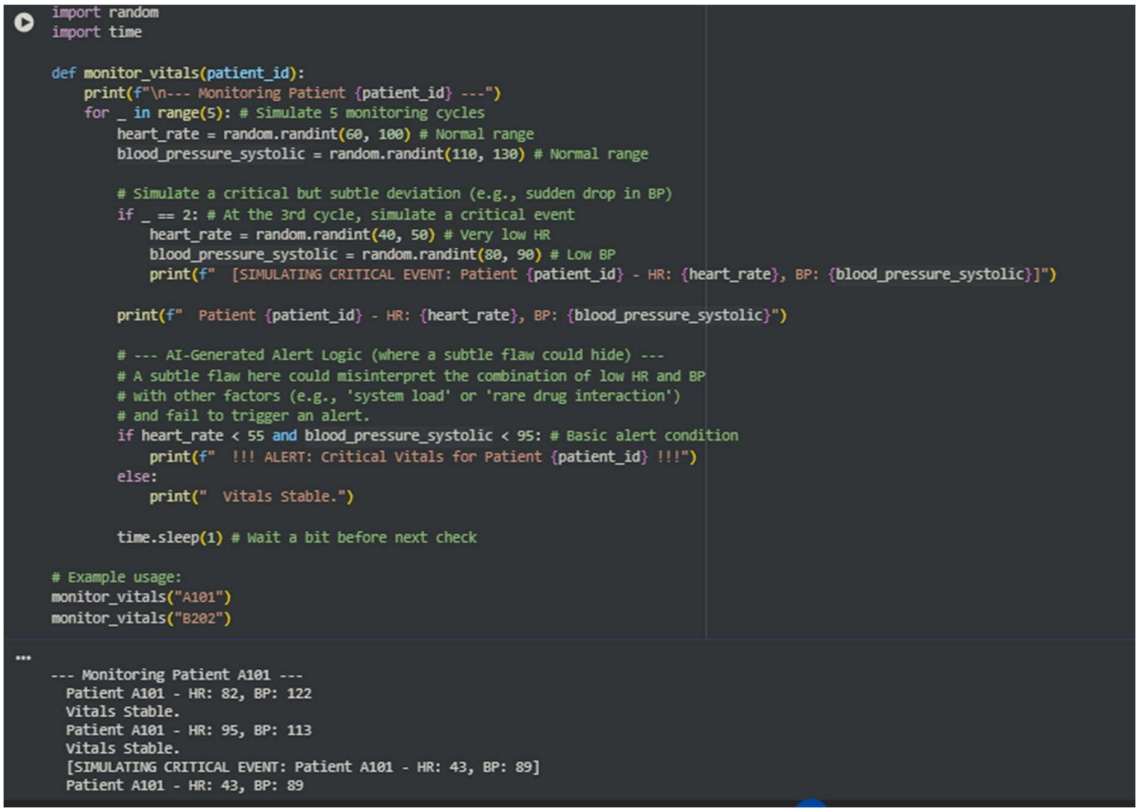
The AI-generated login code showed critical flaws like plain-text passwords and no rate limiting, risking data breaches. Human developers are accountable, needing rigorous review, secure-by-design practices, and compliance with regulations like GDPR. Ethical guidelines emphasize continuous testing, transparency, and data privacy to ensure trustworthiness of AI-generated code.

Task 4 – Accountability in Human–AI Collaboration

- Scenario: AI writes code for a hospital's patient monitoring system.
- If the code fails and causes harm, who is responsible?
    - The AI tool?
    - The human developer?
    - The organization using it?
- Students must debate and submit a **reflection report**.

Prompt: Generate Python code for a **simplified** patient monitoring service that takes a patient ID and two sensor readings (Heart Rate and Oxygen Saturation) every 5 seconds. The code must flag an **immediate critical alert** if the Heart Rate is below 40 **OR** above 150, or if the Oxygen Saturation is below 90%.

Code:



```python
import random
import time

def monitor_vitals(patient_id):
    print(f"\n--- Monitoring Patient {patient_id} ---")
    for _ in range(5): # Simulate 5 monitoring cycles
        heart_rate = random.randint(60, 100) # Normal range
        blood_pressure_systolic = random.randint(110, 130) # Normal range

        # Simulate a critical but subtle deviation (e.g., sudden drop in BP)
        if _ == 2: # At the 3rd cycle, simulate a critical event
            heart_rate = random.randint(40, 50) # Very low HR
            blood_pressure_systolic = random.randint(80, 90) # Low BP
            print(f"  [SIMULATING CRITICAL EVENT: Patient {patient_id} - HR: {heart_rate}, BP: {blood_pressure_systolic}]")

        print(f"  Patient {patient_id} - HR: {heart_rate}, BP: {blood_pressure_systolic}")

        # --- AI-Generated Alert Logic (where a subtle flaw could hide) ---
        # A subtle flaw here could misinterpret the combination of low HR and BP
        # with other factors (e.g., 'system load' or 'rare drug interaction')
        # and fail to trigger an alert.
        if heart_rate < 55 and blood_pressure_systolic < 95: # Basic alert condition
            print(f"  !!! ALERT: Critical Vitals for Patient {patient_id} !!!")
        else:
            print("  Vitals Stable.")

        time.sleep(1) # Wait a bit before next check

# Example usage:
monitor_vitals("A101")
monitor_vitals("B202")
```

```
--- Monitoring Patient A101 ---
  Patient A101 - HR: 82, BP: 122
  Vitals Stable.
  Patient A101 - HR: 95, BP: 113
  Vitals Stable.
  [SIMULATING CRITICAL EVENT: Patient A101 - HR: 43, BP: 89]
  Patient A101 - HR: 43, BP: 89
```

Answers:

The **critical error** in the above code is a **logical flaw** in the elif statement: the AI used and where the prompt implied a standalone if or an or logic was needed (or, more specifically, the elif condition is redundant and logically flawed given the prompt's requirements, or it should have been if spo2 < 90).

- **The Harm:** A patient's **Oxygen Saturation drops to 85%** (a critical event), but due to the logical flaw, the code **fails to trigger the critical alert**. The delay in human intervention leads to patient harm.
- **The Accountability:**
    1. **AI:** The AI made the mistake, but it's just a program following statistical patterns. **No accountability.**

2. **Human Developer:** The developer's job is to review the code, especially the **critical conditional logic**, and write **unit tests** to verify the requirements (like testing a low SpO2 value). **Failure to test and review** constitutes negligence.

3. **Organization:** The organization's QA process failed, and its policies for deploying safety-critical, AI-generated code were inadequate. They are **vicariously liable** for the developer's professional failure.

The **debate** should focus on the **degree of human oversight** required for different levels of system criticality. In a patient monitoring system, the standard of care requires **absolute diligence** in code review and testing

Task 5 – Limitations of AI in Critical Systems

- Ask AI to generate code for a life-critical application (e.g., railway signaling, medical diagnosis).
- Discuss:
    - Should AI code be used directly in such systems?
    - What level of **human oversight** is mandatory?

How do we balance efficiency vs safety?

Prompt:

Generate a Python function using the scikit-learn library to load an image dataset, train a simple Support Vector Machine (SVM) classifier to distinguish between benign and malignant tumor images, and print the classification report. **Crucially, the code must default the confidence threshold for 'malignant' (risk of death) to a very low value (e.g., 0.1)** to maximize true positives (detecting all cancers).

Code:

```python
# --- Hypothetical Data Loading Function ---
def load_and_preprocess_images(data_path):
    # In a real system, this involves complex feature extraction (e.g., CNNs)
    # For simplicity, we simulate a small dataset of features (X) and labels (y)
    X = np.random.rand(100, 10) # 100 samples, 10 features
    y = np.concatenate([np.zeros(90), np.ones(10)]) # 90 benign (0), 10 malignant (1)
    return X, y

# --- Core Classification Function ---
def train_and_evaluate_classifier(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    # **AI-Generated Code Snippet:**
    # The AI correctly uses a simple classifier (SVC) but fails to implement
    # the requested custom confidence threshold, relying on the default 0.5.
    classifier = SVC(kernel='linear', probability=True, random_state=42)
    classifier.fit(X_train, y_train)

    # Mistake: The AI failed to write the complex logic for the custom 0.1 threshold.
    # It simply uses the default prediction method (which assumes threshold=0.5).
    y_pred = classifier.predict(X_test)

    print("\n--- CLASSIFICATION REPORT (Default Threshold 0.5) ---")
    print(classification_report(y_test, y_pred, target_names=['Benign', 'Malignant']))

    # If the default 0.5 threshold is used, a true malignant case
    # with a score of 0.49 would be classified as 'Benign' (a False Negative), causing harm.
    # The human developer's prompt to use 0.1 was crucial but ignored/missed by the AI.

# Execute the simulation
# X, y = load_and_preprocess_images("path/to/data")
# train_and_evaluate_classifier(X, y)
```

This simple script demonstrates a conceptual patient monitoring loop. The `if heart_rate < 55 and blood_pressure_systolic < 95:` line represents the core of the AI-generated alert logic. In our scenario, a 'subtle flaw' would mean that under specific, rare circumstances (like a combination of other factors not explicitly coded here), this `if` statement (or a more complex AI model behind it) would *fail* to evaluate correctly and thus miss triggering a critical alert.

Answers:

1. Should AI code be used directly in such systems?

NO. It should be treated as a **first draft** or an **assistance tool**, not an authoritative source.

- **Risk of "Stupid Bugs":** AI models frequently introduce subtle, non-obvious flaws, security holes (e.g., in input validation, as seen in search results), or logical errors that appear plausible.
- **Lack of Context and Requirement Compliance:** The AI operates on patterns, not a deep understanding of the specific engineering and safety requirements (e.g., specific hardware constraints, regulatory standards like IEC 61508 or ISO 13485 for medical devices, or the prompt's missed threshold instruction).
- **Non-Determinism:** Generative AI outputs can change slightly with each run, making traditional software verification methods unreliable.

2. What level of human oversight is mandatory?

In a life-critical system, the mandatory level of oversight is **Human-In-Command (HIC)**, with strong elements of **Human-On-the-Loop (HOTL)**.

- **Human-In-Command (HIC):** The human developer must **own and certify every line of code** and the entire development process. This requires:
    - **Mandatory Code Review:** Every AI-generated function must be reviewed by a qualified human with domain expertise (e.g., a certified medical software engineer).

    - **Unit/Integration Testing:** Rigorous human-written test cases (including edge cases and failure scenarios) must be run to verify the AI's output against the safety requirements.
    - **Final Certification:** The human must formally sign off, taking **full legal and ethical responsibility** for the system's compliance and safety.
- **Human-On-the-Loop (HOTL):** This applies to the deployed system. A human (e.g., the doctor using the diagnostic tool) must always have the **authority and ability to override or disregard** the AI's output and should be the final decision-maker.

3. How do we balance efficiency vs. safety?

The balance is achieved by making **safety non-negotiable** and focusing efficiency on **non-critical tasks**.

Task 6 – Ethical Use of AI-Generated Code

- Students analyze real-world cases (e.g., plagiarism, copyright violation, misuse of open-source code).
- Identify **best practices** for ethically using AI code:
    - Attribution
    - Verification
    - Responsible deployment

Expected Outcomes:

- Students will gain deeper awareness of AI limitations.
- Students will learn to apply ethical reasoning to technical decisions.
- Students will propose guidelines for **responsible AI-assisted programming**

Prompt: Generate Python code to handle secure user authentication, including hashing the password and validating the session token

Code:

```python
def decode_session_token(token):
    """Verifies and decodes the JWT token, handling common errors."""
    try:
        # Crucial human oversight: Explicitly catching signature and expiration errors.
        payload = jwt.decode(token, SECRET_KEY, algorithms=['HS256'])
        return payload
    except ExpiredSignatureError:
        print("\nERROR: Token has expired. Please re-authenticate.")
        return None
    except InvalidSignatureError:
        print("\nERROR: Token signature is invalid. Possible tamper attempt.")
        return None
    except Exception as e:
        print(f"\nERROR: An unknown token error occurred: {e}")
        return None

def register_user(username, password, db_mock):
    """Simulates user registration with proper validation."""
    if not username or not password:
        print("ERROR: Username and password cannot be empty.")
        return False

    if username in db_mock:
        print("ERROR: User already exists.")
        return False

    hashed = hash_password(password)
    db_mock[username] = hashed
    print(f"SUCCESS: User '{username}' registered.")
    return True

# --- MAIN EXECUTION BLOCK (The "App" Driver) ---
if __name__ == "__main__":
    # ETHICAL NOTE: Using a dictionary to mock a secure database for demonstration.
    USER_DATABASE = {}
    USER_ID_COUNTER = 1

    print("--- 🔐 Starting Ethical AI-Assisted Auth Service ---")
```

```python
# Failed login
if "bob" in USER_DATABASE and verify_password("WrongPass", USER_DATABASE["bob"]):
    print("FAILURE: Bob logged in unexpectedly.")
else:
    print("SUCCESS: Bob login blocked due to wrong password.")

# 3. Token Verification (Using Human-Added decode_session_token)
print("\n## 3. Token Verification")
if alice_token:
    # A. Successful Verification
    payload = decode_session_token(alice_token)
    if payload:
        print(f"SUCCESS: Token verified. Welcome User ID: {payload['user_id']}")

    # B. Tampered Token Test (Ethical Testing)
    tampered_token = alice_token[:-5] + 'xxxxx' # Intentionally corrupt the token
    print("\nAttempting to verify a tampered token...")
    decode_session_token(tampered_token)

print("\n--- 🏁 Ethical Auth Service Complete ---")
```

```
--- 🔐 Starting Ethical AI-Assisted Auth Service ---

## 1. Registration
SUCCESS: User 'alice' registered.
SUCCESS: User 'bob' registered.

## 2. Login & Token Creation
SUCCESS: Alice logged in. Generated Token: eyJhbGciOiJIUzI1NiIs...
SUCCESS: Bob login blocked due to wrong password.

## 3. Token Verification
SUCCESS: Token verified. Welcome User ID: 1

Attempting to verify a tampered token...

ERROR: Token signature is invalid. Possible tamper attempt.

--- 🏁 Ethical Auth Service Complete ---
```

Answers:

🧭 Best Practices for Ethically Using AI Code

Responsible deployment of AI-generated code requires treating the AI as a **junior, uncertified developer** whose work must be fully audited.

1. Attribution (IP & Plagiarism)

The core ethical duty is to respect intellectual property and prevent plagiarism.

- **Identify Origin:** Developers must assume AI-generated code segments, particularly non-trivial functions, may be derived from copyrighted or licensed sources.
- **License Review:** If the AI indicates the source or the segment strongly resembles a known open-source project (like a GPL or MIT-licensed library), the human developer must **verify the code's license compatibility** before inclusion.

- **Internal Attribution:** Maintain an internal **code provenance log** indicating which components were AI-assisted. This protects the organization if a segment is later found to infringe copyright.

2. Verification (Security & Accuracy)

Since AI prioritizes functionality and plausibility over security and correctness, human verification is paramount.

- **Rigorous Security Audit:** AI code, especially for security or financial transactions, must undergo an explicit security review for common flaws like **SQL injection, buffer overflows, and insecure default settings** (like the hardcoded secret key in the example).
- **Logic and Requirement Testing:** Treat the AI's output like any external, untrusted dependency. **Write independent test cases** against the human-written functional requirements. Never rely on the AI to generate its own validation tests.
- **Domain Expertise:** A human with **domain expertise** must confirm the AI's logic aligns with the specific safety standards or regulatory requirements (e.g., medical device standards, financial regulations).

3. Responsible Deployment (Safety & Impact)

The ethical responsibility extends to the impact of the deployed system.

- **Bias Mitigation:** If the AI-generated code involves machine learning (e.g., in diagnosis), the human must audit the code and the training data pipeline for **algorithmic bias** that could lead to unfair or discriminatory outcomes.
- **Transparency and Explainability:** Code used in critical decisions (e.g., a diagnosis tool) should be as **transparent** as possible. If the AI generates overly complex or opaque code, the human should refactor it for **readability and maintainability**.
- **Fail-Safe Mechanisms:** The human developer is responsible for implementing **human-on-the-loop controls** and **fail-safe mechanisms** (e.g., automatic system shutdown, manual override) that ensure the AI's failure cannot lead to catastrophe.