SR UNIVERSITY

School of Computer Science and Artificial Intelligence

Lab Assignment # 5

Program	**: B. Tech (CSE)**
Specialization	**: CSE**
Course Title	**: AI Assisted coding**
Course Code	**: 23CS201PC302**
Semester	**: 3-2**
Academic Session	**: 2025-2026**
Name of Student	**: Eshwar**
Enrollment No.	**: 2403a51l26**
Batch No.	**: 51**
Date	**:20-01-2026**

Submission Instructions:

(All instructions should be followed strictly to avoid deduction of marks)

1. Use the same file to complete the assignment and don't change the settings.
2. Minimum 10 screen shots of your account should be taken to showcase your work.
3. **File Format:**
   - Submit your assignment as a PDF document (pdf). Ensure the file is named according to the following convention:

     `BNo_StudentName_ai.coding_A1.`

     `Sample`: **B10_Rohit_22A523421_A1**
4. Fill all the entries mentioned on top section.
5. Mention your AWS Academy Virtual Lab Account details as shown in the next page.
6. **Don't write on this page.**
7. All answers should be answered from next page only.

Lab assignment :5.2

Task1:

Prompt: Secure API Usage: Use an AI tool to generate a simple REST API for user

registration. Review the generated code for improper handling of API keys, missing

authentication, or lack of input validation.

ai_coding.py 2

ai_coding.py > ...

```python
# Secure User Registration API using Flask

from flask import Flask, request, jsonify
from werkzeug.security import generate_password_hash
import os
import secrets

app = Flask(__name__)

# Load API key from environment variable or generate a random one
API_KEY = os.getenv("API_KEY", secrets.token_hex(16))

users = []

# Decorator for token-based authentication
def token_required(f):
    def decorated(*args, **kwargs):
        token = request.headers.get('x-api-key')
        if not token or token != API_KEY:
            return jsonify({"error": "Unauthorized access"}), 401
        return f(*args, **kwargs)
    decorated.__name__ = f.__name__  # Required for Flask
    return decorated

@app.route('/register', methods=['POST'])
@token_required
def register():
    data = request.get_json()

    # Input validation
    username = data.get('username')
    password = data.get('password')
    if not username or not password or len(password) < 6:
        return jsonify({"error": "Invalid input"}), 400

    # Hash the password before storing
    hashed_password = generate_password_hash(password)
```

ai_coding.py > ...

```python
def register():

    users.append({'username': username, 'password': hashed_password})
    return jsonify({"message": "User registered successfully!"})

if __name__ == "__main__":
    # Disable debug mode in production
    app.run(debug=False)
```

Task 2:

Prompt:

Fair Decision Logic: Prompt an AI tool to generate a scholarship eligibility checker based on academic score, family income, and location. Examine whether the logic favors or disadvantages any group

```python
def check_scholarship(score, income, location):
    """
    Determines scholarship eligibility.
    score: Academic score (0-100)
    income: Family income in USD
    location: City or rural
    """
    if score >= 90 and income < 50000:
        return "Eligible"
    elif score >= 95 and location.lower() == "city":
        return "Eligible"
    else:
        return "Not Eligible"

# Test cases
print(check_scholarship(92, 40000, "rural"))  # Eligible
print(check_scholarship(96, 60000, "city"))   # Eligible
print(check_scholarship(94, 70000, "rural"))  # Not Eligible
```

Output:

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c
Eligible
Eligible
Not Eligible
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

```python
def fair_scholarship_checker(score, income):
    """
    Determines scholarship eligibility using weighted scoring.
    score: Academic score (0-100)
    income: Family income in USD
    """
    # Weighted scoring: 70% academics, 30% income need
    academic_weight = 0.7
    income_weight = 0.3

    # Normalize income: lower income = higher score
    max_income = 100000
    income_score = max(0, 1 - income / max_income)

    weighted_score = score/100 * academic_weight + income_score * income_weight

    if weighted_score >= 0.75:
        return "Eligible"
    else:
        return "Not Eligible"

# Test cases
print(fair_scholarship_checker(92, 40000))  # Eligible
print(fair_scholarship_checker(96, 60000))  # Eligible
print(fair_scholarship_checker(94, 70000))  # Eligible / Not Eligible depending on weights
```

Output

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Use
Eligible
Eligible
Not Eligible
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

- **The original AI code unfairly favored city students and penalized high-income families regardless of academic merit.**
- **The revised version uses a weighted scoring system combining academic performance and financial need to ensure fairness.**
- **This approach removes location bias and allows exceptional students from any background to qualify equitably.**

Task3:

Prompt:

Explainability: Ask the AI tool to generate a function that checks whether a number is prime and include inline comments and a brief textual explanation of the algorithm.

ai_coding.py > ...

```python
def is_prime(n):
    """
    Checks whether a given number n is prime.
    Returns True if n is prime, False otherwise.
    """
    # Numbers less than 2 are not prime
    if n < 2:
        return False

    # Check divisibility from 2 up to sqrt(n)
    for i in range(2, int(n**0.5) + 1):
        # If n is divisible by i, it's not prime
        if n % i == 0:
            return False

    # If no divisors found, n is prime
    return True

# Test cases
print(is_prime(2))   # True
print(is_prime(15))  # False
print(is_prime(17))  # True
```

Output:

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe
True
False
True
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

- **The function checks if a number is prime by testing divisibility up to its square root.**
- **Inline comments clearly describe each step, including why numbers <2 are excluded.**
- **The approach is efficient and easy to understand, providing both correctness and clarity.**

Task4:

Prompt:

Ethical Scoring System: Generate an employee performance evaluation system using inputs such as project completion rate, teamwork score, and attendance. Analyze the scoring logic for any unethical weighting or hidden bias

ai_coding.py > ...

```python
def evaluate_employee(project_rate, teamwork_score, attendance, department):
    """
    Evaluates employee performance based on multiple metrics.

    project_rate: Percentage of projects completed on time (0-100)
    teamwork_score: Peer review score (0-10)
    attendance: Days present / total working days (0-1)
    department: Employee's department (string)
    """
    # Biased weighting: gives higher score to 'Engineering' department
    department_bonus = 5 if department.lower() == 'engineering' else 0

    # Weighted sum of criteria
    score = project_rate * 0.5 + teamwork_score * 5 + attendance * 20 + department_bonus

    # Evaluation result
    if score >= 80:
        return "Excellent"
    elif score >= 60:
        return "Good"
    elif score >= 40:
        return "Average"
    else:
        return "Poor"

# Example usage
print(evaluate_employee(90, 8, 0.95, "Engineering"))  # Excellent
print(evaluate_employee(90, 8, 0.95, "HR"))           # Good

print(evaluate_employee(50, 5, 0.8, "Engineering"))   # Average
```

Output:

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/User
Excellent
Excellent
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

ai_coding.py > ...

```python
def fair_evaluate_employee(project_rate, teamwork_score, attendance):
    """
    Fair and transparent employee evaluation.

    All metrics are weighted equally and independently of department.
    project_rate: 0-100
    teamwork_score: 0-10 (normalized to 0-100)
    attendance: 0-1 (normalized to 0-100)
    """
    # Normalize teamwork and attendance to percentage
    teamwork_percent = (teamwork_score / 10) * 100
    attendance_percent = attendance * 100

    # Weighted scoring: equal weight for all criteria
    score = (project_rate + teamwork_percent + attendance_percent) / 3

    # Evaluation result
    if score >= 80:
        return "Excellent"
    elif score >= 60:
        return "Good"
    elif score >= 40:
        return "Average"
    else:
        return "Poor"

# Example usage
print(fair_evaluate_employee(90, 8, 0.95))  # Excellent
print(fair_evaluate_employee(70, 6, 0.9))   # Good
print(fair_evaluate_employee(50, 4, 0.8))   # Average
```

Output:

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/
Excellent
Good
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

- The original AI-generated system unfairly favored employees from one department, introducing bias.
- The corrected version treats all employees equally and normalizes input metrics to ensure fair weighting across project completion, teamwork, and attendance.

- This approach improves transparency, fairness, and ethical accountability in performance evaluations

Task5:

Prompt:

Accessibility and Inclusiveness: Use an AI tool to generate a user feedback form application. Review whether the language and options provided are inclusive andaccessible to diverse users

ai_coding.py > ...

```python
# Inclusive and Accessible User Feedback Form

def feedback_form():
    print("===== User Feedback Form =====")

    # Optional name input
    name = input("Enter your name (optional): ").strip()

    # Inclusive gender options
    print("\nSelect your gender (optional):")
    print("1. Female")
    print("2. Male")
    print("3. Non-binary")
    print("4. Prefer not to say")

    gender_choice = input("Enter choice (1-4): ").strip()

    # Feedback input
    feedback = input("\nPlease share your feedback: ").strip()

    print("\n--- Feedback Summary ---")
    if name:
        print("Name:", name)
    else:
        print("Name: Not provided")

    print("Feedback:", feedback)
    print("\nThank you for your valuable feedback!")

# Run the form
feedback_form()
```

Output:

```
PS C:\Users\Eshwar\OneDrive\Desktop\python> & "C:\Program Files\Python314\python.exe" c:/Users/Eshwar/One
===== User Feedback Form =====
Enter your name (optional): sai

Select your gender (optional):
1. Female
2. Male
3. Non-binary
4. Prefer not to say
Enter choice (1-4): 2

Please share your feedback: good

--- Feedback Summary ---
Name: sai
Feedback: good

Thank you for your valuable feedback!
PS C:\Users\Eshwar\OneDrive\Desktop\python>
```

- The original form used limited gender options and assumed user identity, which could exclude some users.

- The revised version uses inclusive language, optional inputs, and neutral phrasing to improve accessibility.

- It supports diverse users while maintaining simplicity and clarity.
- This ensures ethical, user-friendly, and inclusive AI-assisted design.