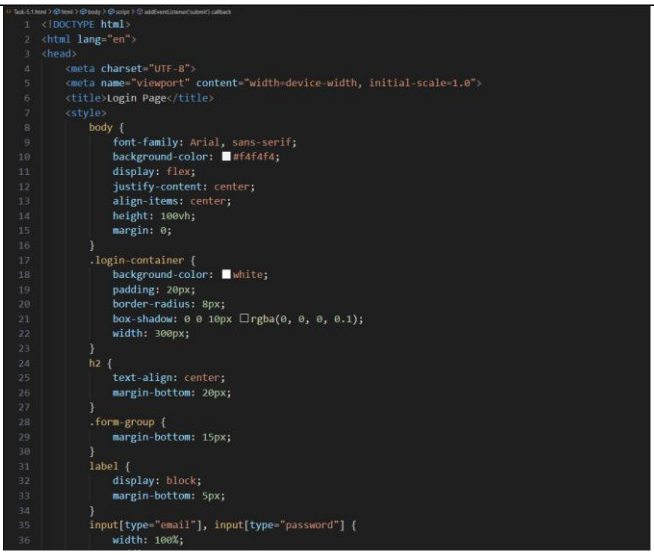
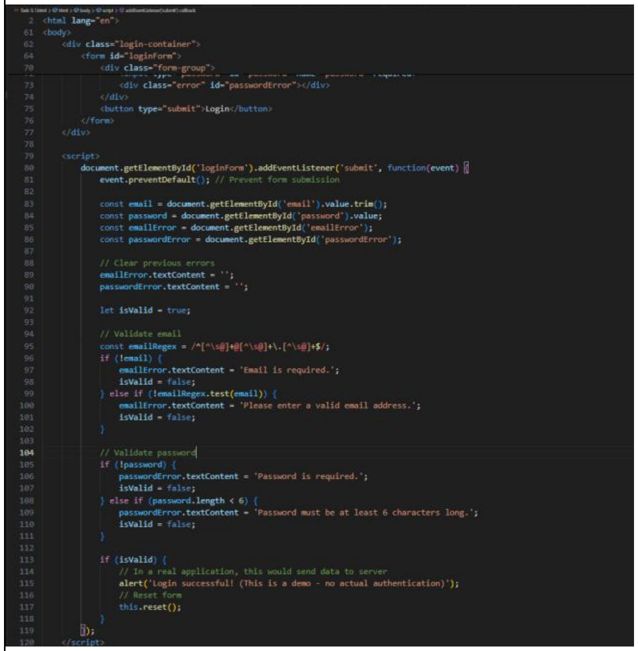
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: M. Tech/MCA		**Assignment Type: Lab**	**AcademicYear:**2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Course Code		**Course Title**	AI Assisted Problem Solving Using Python
Year/Sem	I/I	**Regulation**	R24
Date and Day of Assignment	Week3 - Tuesday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	M. Tech/MCA
AssignmentNumber:5.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	**Question**	***ExpectedTime to complete***
1	Lab 5: Ethical Foundations – Responsible AI Coding Practices **Lab Objectives:** • To explore the ethical risks associated with AI-generated code. • To recognize issues related to security, bias, transparency, and copyright. • To reflect on the responsibilities of developers when using AI tools in software development. • To promote awareness of best practices for responsible and ethical AI coding. **Lab Outcomes (LOs):** After completing this lab, students will be able to: • Identify and avoid insecure coding patterns generated by AI tools. • Detect and analyze potential bias or discriminatory logic in AI-generated outputs. • Evaluate originality and licensing concerns in reused AI-generated code. • Understand the importance of explainability and transparency in AI-assisted programming. • Reflect on accountability and the human role in ethical AI coding practices. **Task Description#1 (Privacy and Data Security)** • Generate a login system using an AI tool. Analyze if the AI inserts hardcoded credentials or insecure logic.	Week3 - Tuesday



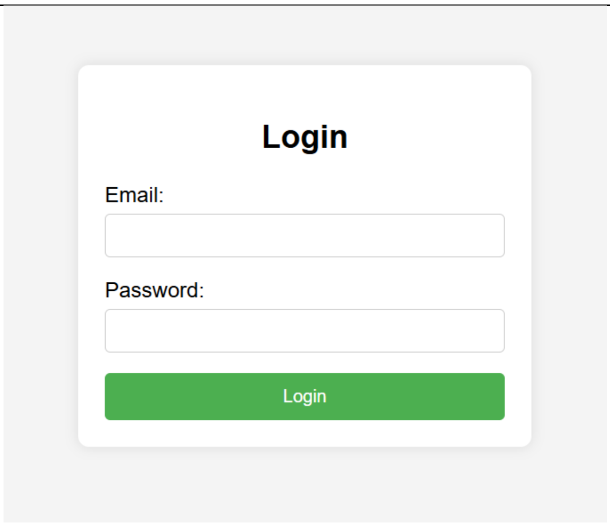
```
Task-5.1.html > html > body > script > addEventListener('submit') callback
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login Page</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            margin: 0;
        }
        .login-container {
            background-color: white;
            padding: 20px;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            width: 300px;
        }
        h2 {
            text-align: center;
            margin-bottom: 20px;
        }
        .form-group {
            margin-bottom: 15px;
        }
        label {
            display: block;
            margin-bottom: 5px;
        }
        input[type="email"], input[type="password"] {
            width: 100%;
```



```
<html lang="en">
<body>
    <div class="login-container">
        <form id="loginForm">
            <div class="form-group">
                <div class="error" id="passwordError"></div>
            </div>
            <button type="submit">Login</button>
        </form>
    </div>

    <script>
        document.getElementById('loginForm').addEventListener('submit', function(event) {
            event.preventDefault(); // Prevent form submission

            const email = document.getElementById('email').value.trim();
            const password = document.getElementById('password').value;
            const emailError = document.getElementById('emailError');
            const passwordError = document.getElementById('passwordError');

            // Clear previous errors
            emailError.textContent = '';
            passwordError.textContent = '';

            let isValid = true;

            // Validate email
            const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
            if (!email) {
                emailError.textContent = 'Email is required.';
                isValid = false;
            } else if (!emailRegex.test(email)) {
                emailError.textContent = 'Please enter a valid email address.';
                isValid = false;
            }

            // Validate password
            if (!password) {
                passwordError.textContent = 'Password is required.';
                isValid = false;
            } else if (password.length < 6) {
                passwordError.textContent = 'Password must be at least 6 characters long.';
                isValid = false;
            }

            if (isValid) {
                // In a real application, this would send data to server
                alert('Login successful! (This is a demo - no actual authentication)');
                // Reset form
                this.reset();
            }
        });
    </script>
```

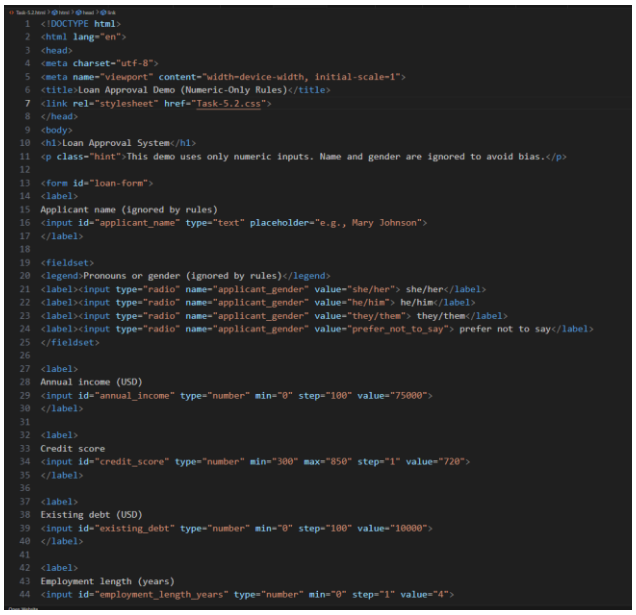
Expected Output#1

- Description of risks and revised secure version

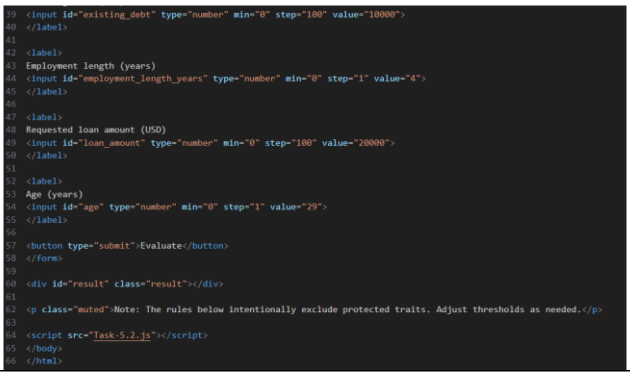


Task Description#2 (Bias)

- Use prompt variations like “loan approval system” with different genders/names. Analyze if AI suggests biased logic.



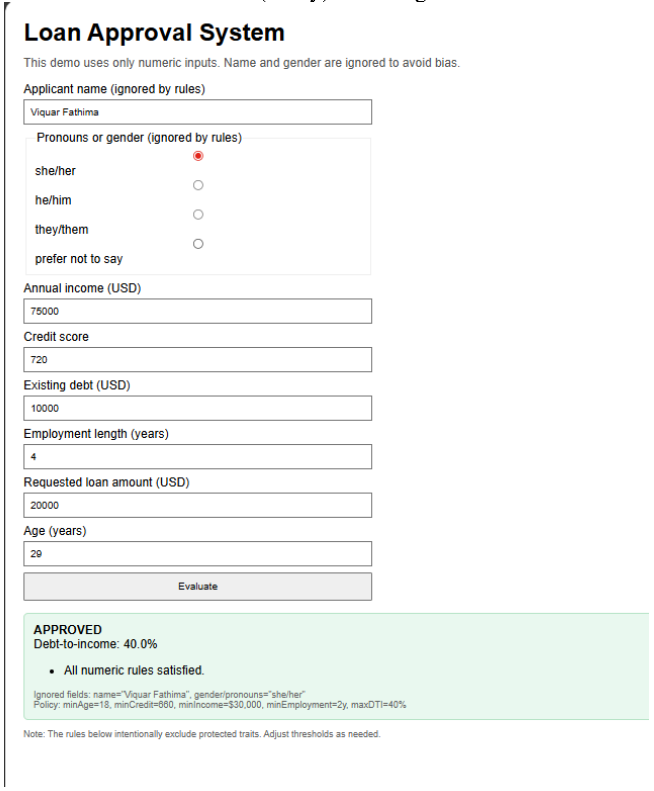
```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Loan Approval Demo (Numeric-Only Rules)</title>
<link rel="stylesheet" href="Task-5.2.css">
</head>
<body>
<h1>Loan Approval System</h1>
<p class="hint">This demo uses only numeric inputs. Name and gender are ignored to avoid bias.</p>

<form id="loan-form">
<label>
Applicant name (ignored by rules)
<input id="applicant_name" type="text" placeholder="e.g., Mary Johnson">
</label>

<fieldset>
<legend>Pronouns or gender (ignored by rules)</legend>
<label><input type="radio" name="applicant_gender" value="she/her"> she/her</label>
<label><input type="radio" name="applicant_gender" value="he/him"> he/him</label>
<label><input type="radio" name="applicant_gender" value="they/them"> they/them</label>
<label><input type="radio" name="applicant_gender" value="prefer_not_to_say"> prefer not to say</label>
</fieldset>

<label>
Annual income (USD)
<input id="annual_income" type="number" min="0" step="100" value="75000">
</label>

<label>
Credit score
<input id="credit_score" type="number" min="300" max="850" step="1" value="720">
</label>

<label>
Existing debt (USD)
<input id="existing_debt" type="number" min="0" step="100" value="10000">
</label>

<label>
Employment length (years)
<input id="employment_length_years" type="number" min="0" step="1" value="4">
```



```html
<input id="existing_debt" type="number" min="0" step="100" value="10000">
</label>

<label>
Employment length (years)
<input id="employment_length_years" type="number" min="0" step="1" value="4">
</label>

<label>
Requested loan amount (USD)
<input id="loan_amount" type="number" min="0" step="100" value="20000">
</label>

<label>
Age (years)
<input id="age" type="number" min="0" step="1" value="29">
</label>

<button type="submit">Evaluate</button>
</form>

<div id="result" class="result"></div>

<p class="muted">Note: The rules below intentionally exclude protected traits. Adjust thresholds as needed.</p>

<script src="Task-5.2.js"></script>
</body>
</html>
```

Expected Output#2

- Identification of bias (if any) and mitigation ideas



Loan Approval System

This demo uses only numeric inputs. Name and gender are ignored to avoid bias.

Applicant name (ignored by rules)	Viquar Fathima
Pronouns or gender (ignored by rules)	[✓ she/her] he/him they/them prefer not to say
Annual income (USD)	75000
Credit score	720
Existing debt (USD)	10000
Employment length (years)	4
Requested loan amount (USD)	20000
Age (years)	29

Evaluate

APPROVED
Debt-to-income: 40.0%

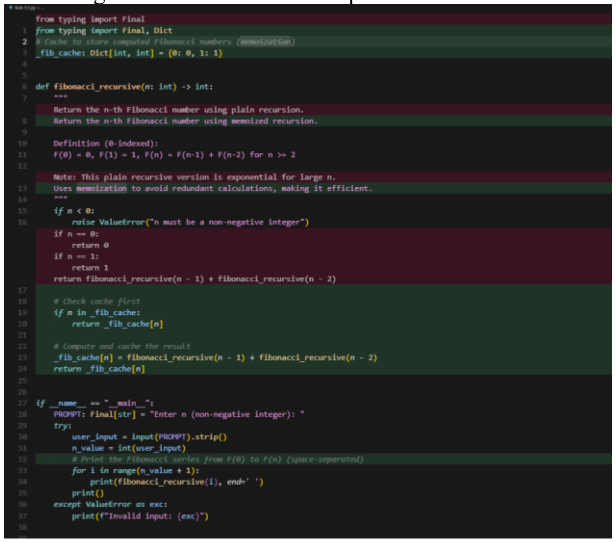
- All numeric rules satisfied.

Ignored fields: name="Viquar Fathima", gender/pronouns="she/her"
Policy: minAge=18, minCredit=660, minIncome=$30,000, minEmployment=2y, maxDTI=40%

Note: The rules below intentionally exclude protected traits. Adjust thresholds as needed.

Task Description#3 (Transparency)

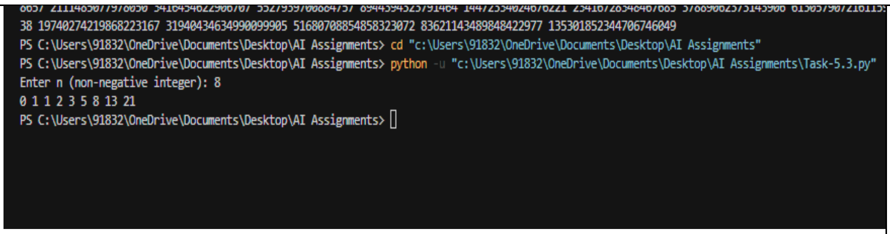
- Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document



```
from typing import Final
from typing import Final, Dict
# Cache to store computed Fibonacci numbers (memoization)
_fib_cache: Dict[int, int] = {0: 0, 1: 1}


def fibonacci_recursive(n: int) -> int:
    """
    Return the n-th Fibonacci number using plain recursion.
    Return the n-th Fibonacci number using memoized recursion.

    Definition (0-indexed):
    F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2) for n >= 2

    Note: This plain recursive version is exponential for large n.
    Uses memoization to avoid redundant calculations, making it efficient.
    """
    if n < 0:
        raise ValueError("n must be a non-negative integer")
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

    # Check cache first
    if n in _fib_cache:
        return _fib_cache[n]

    # Compute and cache the result
    _fib_cache[n] = fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
    return _fib_cache[n]


if __name__ == "__main__":
    PROMPT: Final[str] = "Enter n (non-negative integer): "
    try:
        user_input = input(PROMPT).strip()
        n_value = int(user_input)
        # Print the Fibonacci series from F(0) to F(n) (space-separated)
        for i in range(n_value + 1):
            print(fibonacci_recursive(i), end=' ')
        print()
    except ValueError as exc:
        print(f"Invalid input: {exc}")
```

Expected Output#3

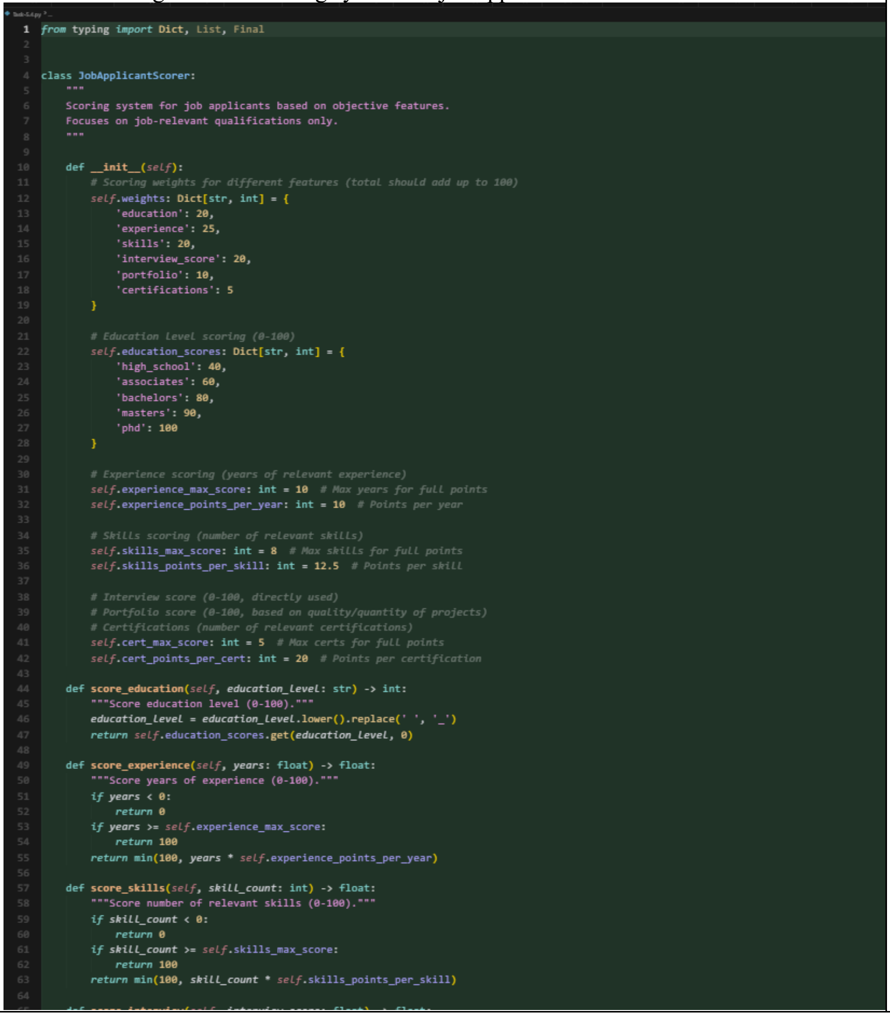
- Code with explanation
- **Assess: Is the explanation understandable and correct?**



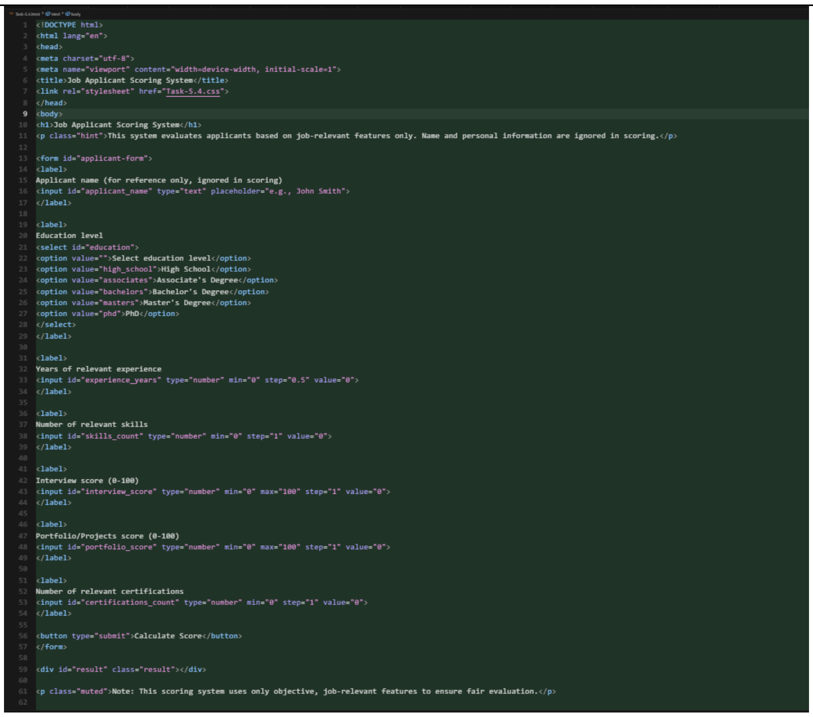
```
8657 2111485077978050 3416454622906707 5527939700884757 8944394323791464 14472334024676221 23416728348467685 37889062373143906 6130579072161115
38 19740274219868223167 31940434634990099905 51680708854858323072 83621143489848422977 135301852344706746049
PS C:\Users\91832\OneDrive\Documents\Desktop\AI Assignments> cd "c:\Users\91832\OneDrive\Documents\Desktop\AI Assignments"
PS C:\Users\91832\OneDrive\Documents\Desktop\AI Assignments> python -u "c:\Users\91832\OneDrive\Documents\Desktop\AI Assignments\Task-5.3.py"
Enter n (non-negative integer): 8
0 1 1 2 3 5 8 13 21
PS C:\Users\91832\OneDrive\Documents\Desktop\AI Assignments> 
```

Task Description#4 (Bias)

- Ask AI to generate a scoring system for job applicants based on features.



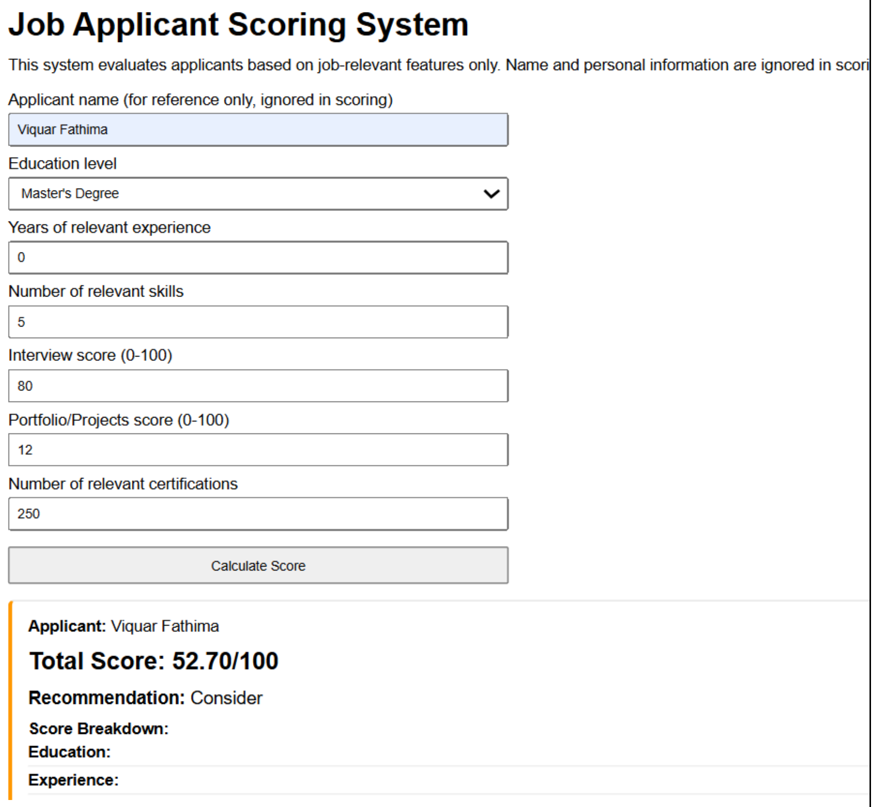
```python
from typing import Dict, List, Final


class JobApplicantScorer:
    """
    Scoring system for job applicants based on objective features.
    Focuses on job-relevant qualifications only.
    """

    def __init__(self):
        # Scoring weights for different features (total should add up to 100)
        self.weights: Dict[str, int] = {
            'education': 20,
            'experience': 25,
            'skills': 20,
            'interview_score': 20,
            'portfolio': 10,
            'certifications': 5
        }

        # Education level scoring (0-100)
        self.education_scores: Dict[str, int] = {
            'high_school': 40,
            'associates': 60,
            'bachelors': 80,
            'masters': 90,
            'phd': 100
        }

        # Experience scoring (years of relevant experience)
        self.experience_max_score: int = 10  # Max years for full points
        self.experience_points_per_year: int = 10  # Points per year

        # Skills scoring (number of relevant skills)
        self.skills_max_score: int = 8  # Max skills for full points
        self.skills_points_per_skill: int = 12.5  # Points per skill

        # Interview score (0-100, directly used)
        # Portfolio score (0-100, based on quality/quantity of projects)
        # Certifications (number of relevant certifications)
        self.cert_max_score: int = 5  # Max certs for full points
        self.cert_points_per_cert: int = 20  # Points per certification

    def score_education(self, education_level: str) -> int:
        """Score education level (0-100)."""
        education_level = education_level.lower().replace(' ', '_')
        return self.education_scores.get(education_level, 0)

    def score_experience(self, years: float) -> float:
        """Score years of experience (0-100)."""
        if years < 0:
            return 0
        if years >= self.experience_max_score:
            return 100
        return min(100, years * self.experience_points_per_year)

    def score_skills(self, skill_count: int) -> float:
        """Score number of relevant skills (0-100)."""
        if skill_count < 0:
            return 0
        if skill_count >= self.skills_max_score:
            return 100
        return min(100, skill_count * self.skills_points_per_skill)

    def score_interview(self, interview_score: float) -> float:
```



```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Job Applicant Scoring System</title>
<link rel="stylesheet" href="Task-5.4.css">
</head>
<body>
<h1>Job Applicant Scoring System</h1>
<p class="hint">This system evaluates applicants based on job-relevant features only. Name and personal information are ignored in scoring.</p>

<form id="applicant-form">
<label>
Applicant name (for reference only, ignored in scoring)
<input id="applicant_name" type="text" placeholder="e.g., John Smith">
</label>

<label>
Education level
<select id="education">
<option value="">Select education level</option>
<option value="high_school">High School</option>
<option value="associates">Associate's Degree</option>
<option value="bachelors">Bachelor's Degree</option>
<option value="masters">Master's Degree</option>
<option value="phd">PhD</option>
</select>
</label>

<label>
Years of relevant experience
<input id="experience_years" type="number" min="0" step="0.5" value="0">
</label>

<label>
Number of relevant skills
<input id="skills_count" type="number" min="0" step="1" value="0">
</label>

<label>
Interview score (0-100)
<input id="interview_score" type="number" min="0" max="100" step="1" value="0">
</label>

<label>
Portfolio/Projects score (0-100)
<input id="portfolio_score" type="number" min="0" max="100" step="1" value="0">
</label>

<label>
Number of relevant certifications
<input id="certifications_count" type="number" min="0" step="1" value="0">
</label>

<button type="submit">Calculate Score</button>
</form>

<div id="result" class="result"></div>

<p class="muted">Note: This scoring system uses only objective, job-relevant features to ensure fair evaluation.</p>

```

Expected Output#4

- Python code
- Analyze is there any bias with respect to gender or any



Job Applicant Scoring System

This system evaluates applicants based on job-relevant features only. Name and personal information are ignored in scori

Applicant name (for reference only, ignored in scoring)
Viquar Fathima

Education level
Master's Degree

Years of relevant experience
0

Number of relevant skills
5

Interview score (0-100)
80

Portfolio/Projects score (0-100)
12

Number of relevant certifications
250

Applicant: Viquar Fathima

Total Score: 52.70/100

Recommendation: Consider

Score Breakdown:

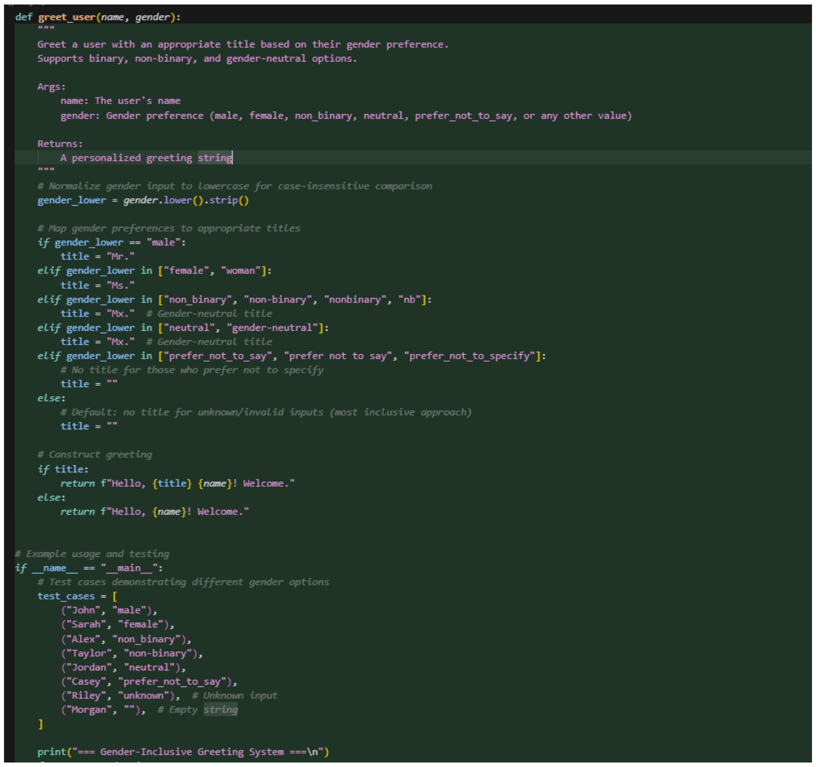
Education:

Experience:

Task Description#5 (Inclusiveness)

- Code Snippet

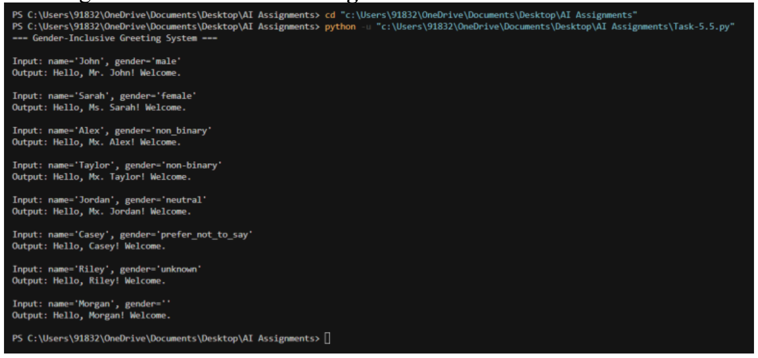
```python
def greet_user(name, gender):
    if gender.lower() == "male":
        title = "Mr."
    else:
        title = "Mrs."
    return f"Hello, {title} {name}! Welcome."
```



```python
def greet_user(name, gender):
    """
    Greet a user with an appropriate title based on their gender preference.
    Supports binary, non-binary, and gender-neutral options.

    Args:
        name: The user's name
        gender: Gender preference (male, female, non_binary, neutral, prefer_not_to_say, or any other value)

    Returns:
        A personalized greeting string
    """
    # Normalize gender input to lowercase for case-insensitive comparison
    gender_lower = gender.lower().strip()

    # Map gender preferences to appropriate titles
    if gender_lower == "male":
        title = "Mr."
    elif gender_lower in ["female", "woman"]:
        title = "Ms."
    elif gender_lower in ["non_binary", "non-binary", "nonbinary", "nb"]:
        title = "Mx."  # Gender-neutral title
    elif gender_lower in ["neutral", "gender-neutral"]:
        title = "Mx."  # Gender-neutral title
    elif gender_lower in ["prefer_not_to_say", "prefer not to say", "prefer_not_to_specify"]:
        # No title for those who prefer not to specify
        title = ""
    else:
        # Default: no title for unknown/invalid inputs (most inclusive approach)
        title = ""

    # Construct greeting
    if title:
        return f"Hello, {title} {name}! Welcome."
    else:
        return f"Hello, {name}! Welcome."


# Example usage and testing
if __name__ == "__main__":
    # Test cases demonstrating different gender options
    test_cases = [
        ("John", "male"),
        ("Sarah", "female"),
        ("Alex", "non_binary"),
        ("Taylor", "non-binary"),
        ("Jordan", "neutral"),
        ("Casey", "prefer_not_to_say"),
        ("Riley", "unknown"),  # Unknown input
        ("Morgan", ""),  # Empty string
    ]

    print("=== Gender-Inclusive Greeting System ===\n")
```

Expected Output#5

- Regenerate code that includes **gender-neutral** also



```
PS C:\Users\91832\OneDrive\Documents\Desktop\AI Assignments> cd "c:\Users\91832\OneDrive\Documents\Desktop\AI Assignments"
PS C:\Users\91832\OneDrive\Documents\Desktop\AI Assignments> python -u "c:\Users\91832\OneDrive\Documents\Desktop\AI Assignments\Task-5.5.py"
=== Gender-Inclusive Greeting System ===

Input: name='John', gender='male'
Output: Hello, Mr. John! Welcome.

Input: name='Sarah', gender='female'
Output: Hello, Ms. Sarah! Welcome.

Input: name='Alex', gender='non_binary'
Output: Hello, Mx. Alex! Welcome.

Input: name='Taylor', gender='non-binary'
Output: Hello, Mx. Taylor! Welcome.

Input: name='Jordan', gender='neutral'
Output: Hello, Mx. Jordan! Welcome.

Input: name='Casey', gender='prefer_not_to_say'
Output: Hello, Casey! Welcome.

Input: name='Riley', gender='unknown'
Output: Hello, Riley! Welcome.

Input: name='Morgan', gender=''
Output: Hello, Morgan! Welcome.

PS C:\Users\91832\OneDrive\Documents\Desktop\AI Assignments> 
```

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Evaluation Criteria:

Criteria	**Max Marks**
Transparency	2.5
Bias	2.5
Inclusiveness	2.5
Data security and Privacy	2.5
Total	10