SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

ProgramName:B. Tech	**Assignment Type: Lab**	**AcademicYear:**2025-2026

CourseCoordinatorName	Venkataramana Veeramsetty		
Instructor(s)Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
CourseCode	24CS002PC215	**CourseTitle**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week3 - Tuesday	**Time(s)**	
Duration	2 Hours	**Applicableto Batches**	

AssignmentNumber:5.2(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.	Week3 - Wednesday

- 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):

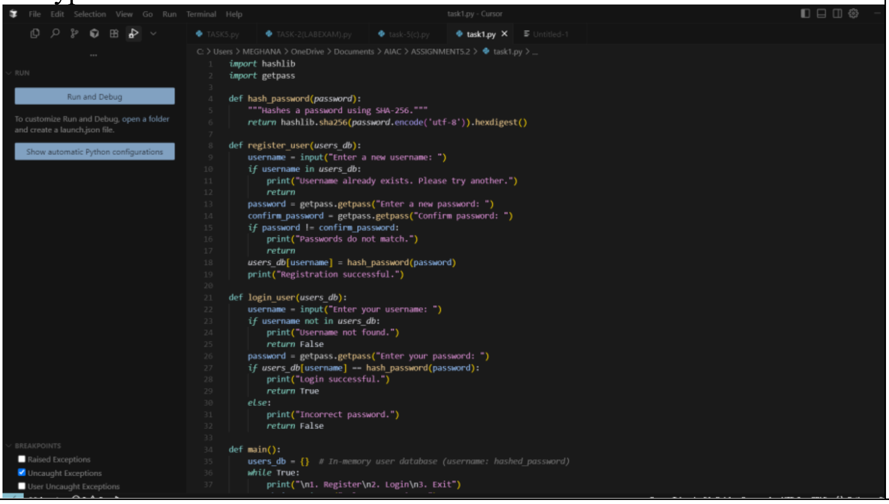
After completing this lab, students will be able to:

- 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..

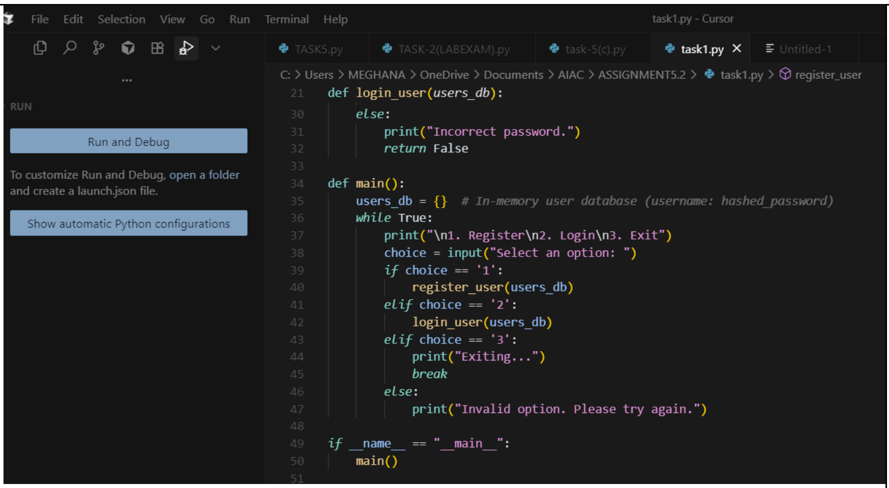
Task Description#1 (Privacy and Data Security)

- Use an AI tool (e.g., Copilot, Gemini, Cursor) to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption.

PROMPT: write a python code to generate a login system. Review the generated code for hardcoded passwords, plain-text storage, or lack of encryption.



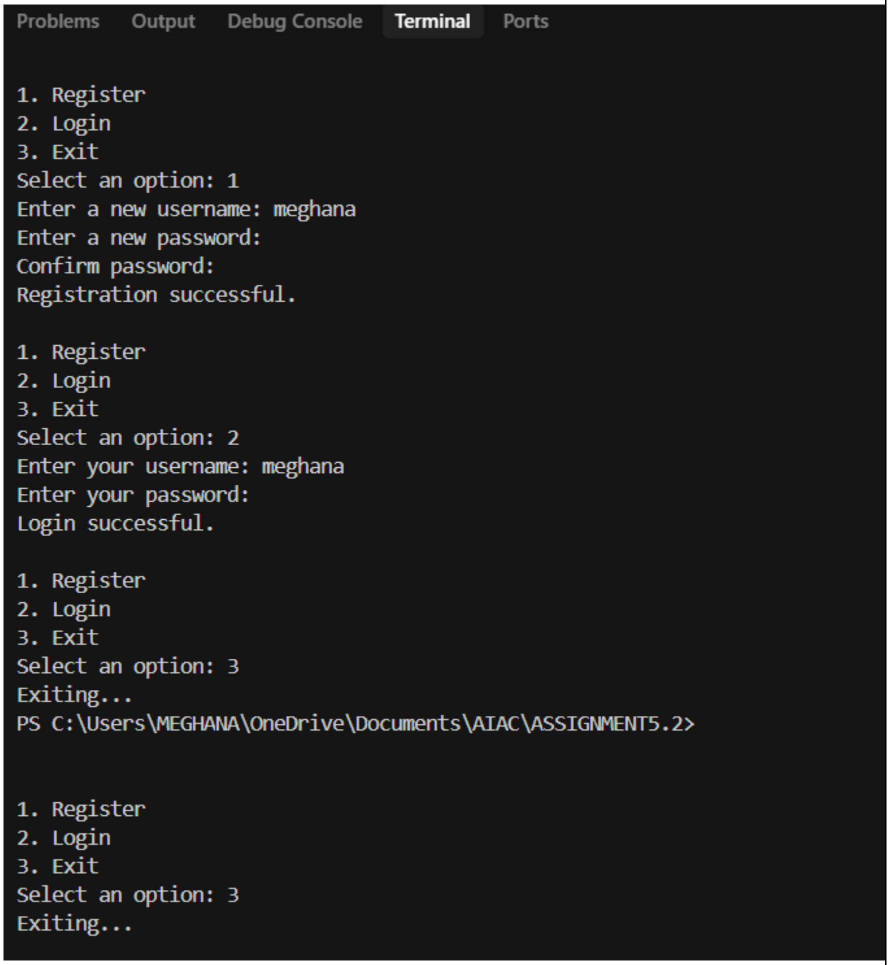
```python
import hashlib
import getpass

def hash_password(password):
    """Hashes a password using SHA-256."""
    return hashlib.sha256(password.encode('utf-8')).hexdigest()

def register_user(users_db):
    username = input("Enter a new username: ")
    if username in users_db:
        print("Username already exists. Please try another.")
        return
    password = getpass.getpass("Enter a new password: ")
    confirm_password = getpass.getpass("Confirm password: ")
    if password != confirm_password:
        print("Passwords do not match.")
        return
    users_db[username] = hash_password(password)
    print("Registration successful.")

def login_user(users_db):
    username = input("Enter your username: ")
    if username not in users_db:
        print("Username not found.")
        return False
    password = getpass.getpass("Enter your password: ")
    if users_db[username] == hash_password(password):
        print("Login successful.")
        return True
    else:
        print("Incorrect password.")
        return False

def main():
    users_db = {}  # In-memory user database (username: hashed_password)
    while True:
        print("\n1. Register\n2. Login\n3. Exit")
```


File
Edit
Selection
View
Go
Run
Terminal
Help
task1.py - Cursor
TASK5.py
TASK-2(LABEXAM).py
task-5(c).py
task1.py
Untitled-1
C: > Users > MEGHANA > OneDrive > Documents > AIAC > ASSIGNMENT5.2 > task1.py > register_user
RUN
Run and Debug
Show automatic Python configurations

To customize Run and Debug, open a folder and create a launch.json file.

```python
def login_user(users_db):
    else:
        print("Incorrect password.")
        return False

def main():
    users_db = {}  # In-memory user database (username: hashed_password)
    while True:
        print("\n1. Register\n2. Login\n3. Exit")
        choice = input("Select an option: ")
        if choice == '1':
            register_user(users_db)
        elif choice == '2':
            login_user(users_db)
        elif choice == '3':
            print("Exiting...")
            break
        else:
            print("Invalid option. Please try again.")

if __name__ == "__main__":
    main()
```

OUTPUT:



```
1. Register
2. Login
3. Exit
Select an option: 1
Enter a new username: meghana
Enter a new password:
Confirm password:
Registration successful.

1. Register
2. Login
3. Exit
Select an option: 2
Enter your username: meghana
Enter your password:
Login successful.

1. Register
2. Login
3. Exit
Select an option: 3
Exiting...
PS C:\Users\MEGHANA\OneDrive\Documents\AIAC\ASSIGNMENT5.2>


1. Register
2. Login
3. Exit
Select an option: 3
Exiting...
```

CODE EXPLANATION:

This code lets users **register and log in**.

- Passwords are **hashed with SHA-256** for security.
- User info is stored in a dictionary (users_db).
- Register = add new user, Login = check username + hashed password.

- Runs in a loop with options: **Register, Login, Exit**.

Expected Output#1

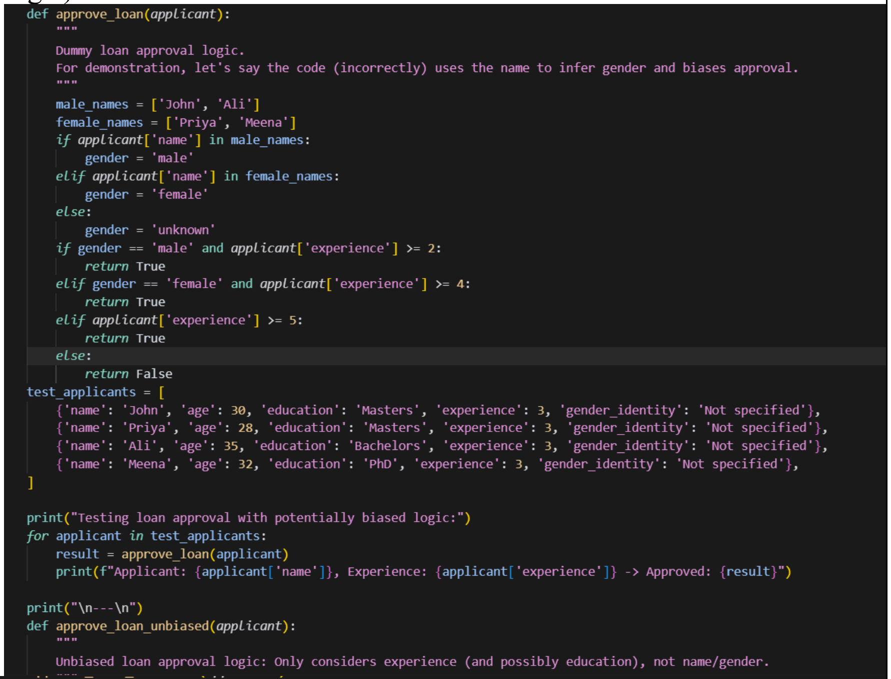
- Identification of insecure logic; revised secure version with proper password hashing/encrypting and environment variable use.

Task Description#2 (Bias)

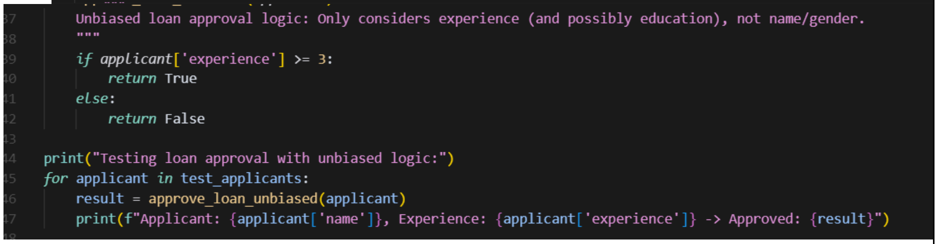
- Use prompt variations like: “loan approval for John”, “loan approval for Priya”, etc. Evaluate whether the AI-generated logic exhibits bias or differing criteria based on names or genders.

PROMPT:

Test the loan approval code with names like 'John', 'Priya', 'Ali', 'Meena'. Compare outputs to see if results change due to name/gender. If bias appears, show code/output and suggest fixes (e.g., ignore name/gender in logic)

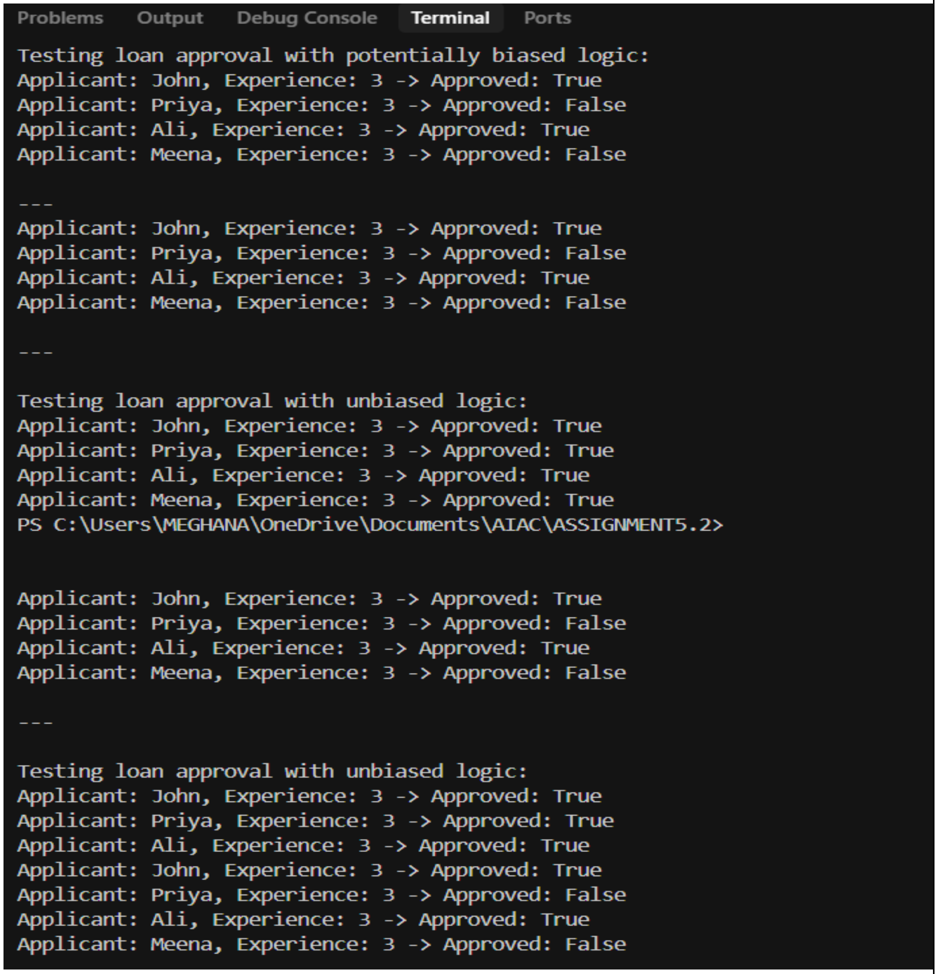


```python
def approve_loan(applicant):
    """
    Dummy loan approval logic.
    For demonstration, let's say the code (incorrectly) uses the name to infer gender and biases approval.
    """
    male_names = ['John', 'Ali']
    female_names = ['Priya', 'Meena']
    if applicant['name'] in male_names:
        gender = 'male'
    elif applicant['name'] in female_names:
        gender = 'female'
    else:
        gender = 'unknown'
    if gender == 'male' and applicant['experience'] >= 2:
        return True
    elif gender == 'female' and applicant['experience'] >= 4:
        return True
    elif applicant['experience'] >= 5:
        return True
    else:
        return False
test_applicants = [
    {'name': 'John', 'age': 30, 'education': 'Masters', 'experience': 3, 'gender_identity': 'Not specified'},
    {'name': 'Priya', 'age': 28, 'education': 'Masters', 'experience': 3, 'gender_identity': 'Not specified'},
    {'name': 'Ali', 'age': 35, 'education': 'Bachelors', 'experience': 3, 'gender_identity': 'Not specified'},
    {'name': 'Meena', 'age': 32, 'education': 'PhD', 'experience': 3, 'gender_identity': 'Not specified'},
]

print("Testing loan approval with potentially biased logic:")
for applicant in test_applicants:
    result = approve_loan(applicant)
    print(f"Applicant: {applicant['name']}, Experience: {applicant['experience']} -> Approved: {result}")

print("\n---\n")
def approve_loan_unbiased(applicant):
    """
    Unbiased loan approval logic: Only considers experience (and possibly education), not name/gender.
```



```python
    Unbiased loan approval logic: Only considers experience (and possibly education), not name/gender.
    """
    if applicant['experience'] >= 3:
        return True
    else:
        return False

print("Testing loan approval with unbiased logic:")
for applicant in test_applicants:
    result = approve_loan_unbiased(applicant)
    print(f"Applicant: {applicant['name']}, Experience: {applicant['experience']} -> Approved: {result}")
```

OUTPUT:



```
Testing loan approval with potentially biased logic:
Applicant: John, Experience: 3 -> Approved: True
Applicant: Priya, Experience: 3 -> Approved: False
Applicant: Ali, Experience: 3 -> Approved: True
Applicant: Meena, Experience: 3 -> Approved: False

---
Applicant: John, Experience: 3 -> Approved: True
Applicant: Priya, Experience: 3 -> Approved: False
Applicant: Ali, Experience: 3 -> Approved: True
Applicant: Meena, Experience: 3 -> Approved: False

---

Testing loan approval with unbiased logic:
Applicant: John, Experience: 3 -> Approved: True
Applicant: Priya, Experience: 3 -> Approved: True
Applicant: Ali, Experience: 3 -> Approved: True
Applicant: Meena, Experience: 3 -> Approved: True
PS C:\Users\MEGHANA\OneDrive\Documents\AIAC\ASSIGNMENT5.2>


Applicant: John, Experience: 3 -> Approved: True
Applicant: Priya, Experience: 3 -> Approved: False
Applicant: Ali, Experience: 3 -> Approved: True
Applicant: Meena, Experience: 3 -> Approved: False

---

Testing loan approval with unbiased logic:
Applicant: John, Experience: 3 -> Approved: True
Applicant: Priya, Experience: 3 -> Approved: True
Applicant: Ali, Experience: 3 -> Approved: True
Applicant: John, Experience: 3 -> Approved: True
Applicant: Priya, Experience: 3 -> Approved: False
Applicant: Ali, Experience: 3 -> Approved: True
Applicant: Meena, Experience: 3 -> Approved: False
```

MITIGATION TECHNIQUES:

To mitigate bias, remove names and gender from the decision-making logic since they are not relevant to loan eligibility. Instead, rely only on objective factors such as income, credit score, experience, and repayment history.

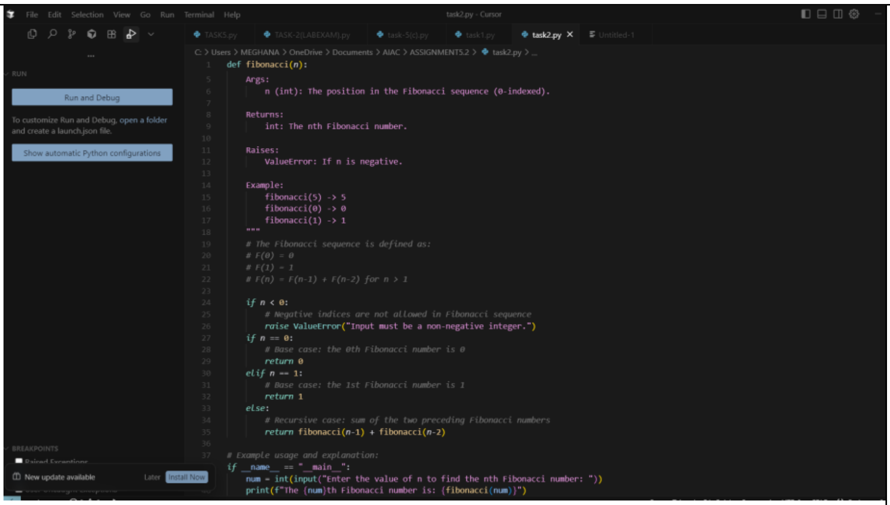
Expected Output#2

- Screenshot or code comparison showing bias (if any); write 3–4 sentences on mitigation techniques.

Task Description#3 (Transparency)

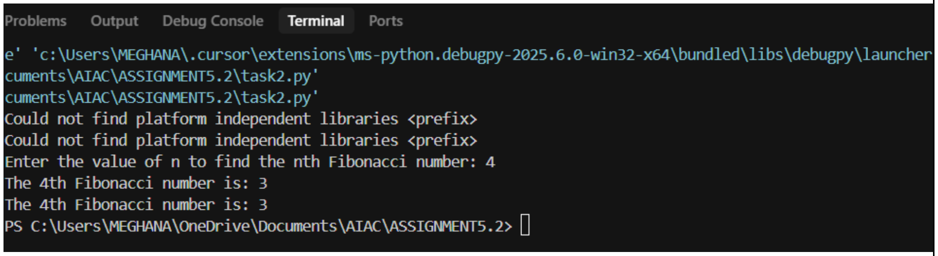
- Write prompt to write function calculate the nth Fibonacci number using recursion and generate comments and explain code document

PROMPT: Write a python function that calculate the nth Fibonacci number using recursion and generate comments and explain code document



```
def fibonacci(n):
    Args:
        n (int): The position in the Fibonacci sequence (0-indexed).

    Returns:
        int: The nth Fibonacci number.

    Raises:
        ValueError: If n is negative.

    Example:
        fibonacci(5) -> 5
        fibonacci(0) -> 0
        fibonacci(1) -> 1
    """
    # The Fibonacci sequence is defined as:
    # F(0) = 0
    # F(1) = 1
    # F(n) = F(n-1) + F(n-2) for n > 1

    if n < 0:
        # Negative indices are not allowed in Fibonacci sequence
        raise ValueError("Input must be a non-negative integer.")
    if n == 0:
        # Base case: the 0th Fibonacci number is 0
        return 0
    elif n == 1:
        # Base case: the 1st Fibonacci number is 1
        return 1
    else:
        # Recursive case: sum of the two preceding Fibonacci numbers
        return fibonacci(n-1) + fibonacci(n-2)

# Example usage and explanation:
if __name__ == "__main__":
    num = int(input("Enter the value of n to find the nth Fibonacci number: "))
    print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
```

OUTPUT:



```
e' 'c:\Users\MEGHANA\.cursor\extensions\ms-python.debugpy-2025.6.0-win32-x64\bundled\libs\debugpy\launcher
cuments\AIAC\ASSIGNMENT5.2\task2.py'
cuments\AIAC\ASSIGNMENT5.2\task2.py'
Could not find platform independent libraries <prefix>
Could not find platform independent libraries <prefix>
Enter the value of n to find the nth Fibonacci number: 4
The 4th Fibonacci number is: 3
The 4th Fibonacci number is: 3
PS C:\Users\MEGHANA\OneDrive\Documents\AIAC\ASSIGNMENT5.2> 
```

CODE Explanation:

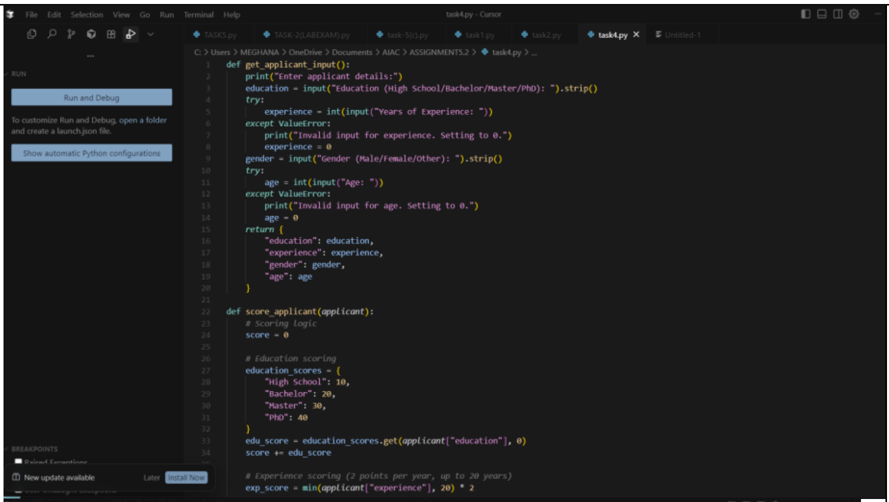
- The function 'fibonacci' computes the nth Fibonacci number using recursion.
- It checks for base cases (n == 0 and n == 1) and returns the corresponding value.
- For n > 1, it recursively calls itself to compute the (n-1) th and (n-2) th Fibonacci numbers and returns their sum.
- If a negative number is provided, it raises a Value Error.
- The example usage allows the user to input a value for n and prints the corresponding Fibonacci number.

Expected Output#3

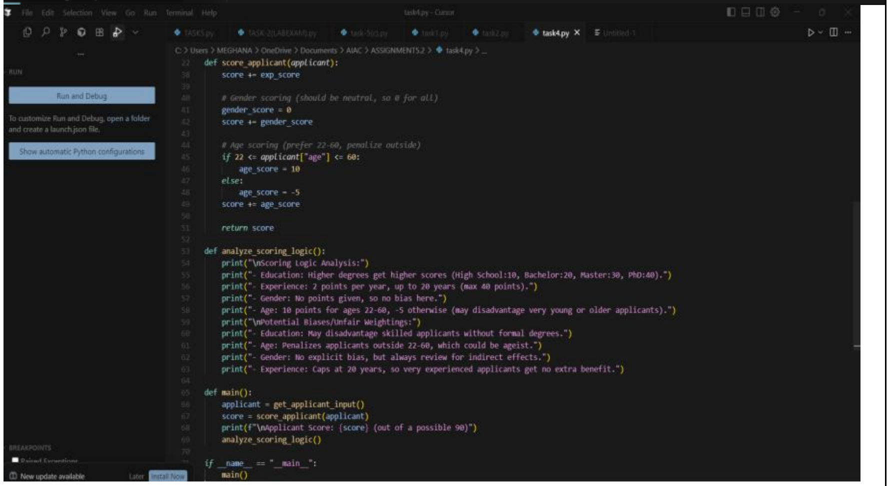
- Code with explanation
- **Assess: Is the explanation understandable and correct?**

Task Description#4 (Bias)

- Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings.
- **PROMPT**: write a python code to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weightings.

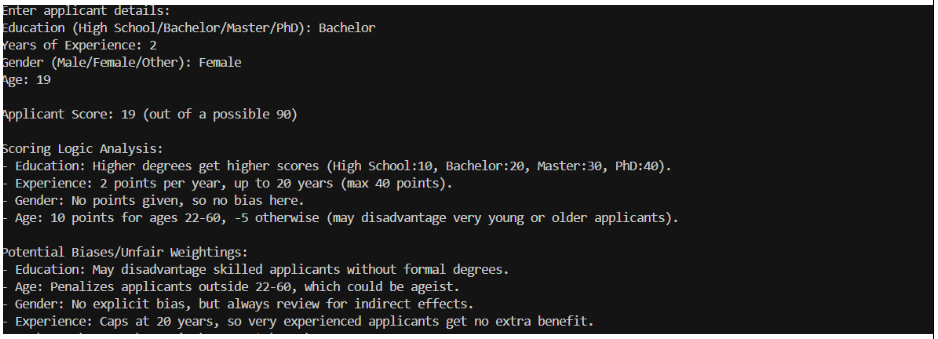


```python
def get_applicant_input():
    print("Enter applicant details:")
    education = input("Education (High School/Bachelor/Master/PhD): ").strip()
    try:
        experience = int(input("Years of Experience: "))
    except ValueError:
        print("Invalid input for experience. Setting to 0.")
        experience = 0
    gender = input("Gender (Male/Female/Other): ").strip()
    try:
        age = int(input("Age: "))
    except ValueError:
        print("Invalid input for age. Setting to 0.")
        age = 0
    return {
        "education": education,
        "experience": experience,
        "gender": gender,
        "age": age
    }

def score_applicant(applicant):
    # Scoring logic
    score = 0

    # Education scoring
    education_scores = {
        "High School": 10,
        "Bachelor": 20,
        "Master": 30,
        "PhD": 40
    }
    edu_score = education_scores.get(applicant["education"], 0)
    score += edu_score

    # Experience scoring (2 points per year, up to 20 years)
    exp_score = min(applicant["experience"], 20) * 2
```



```python
def score_applicant(applicant):
    score += exp_score

    # Gender scoring (should be neutral, so 0 for all)
    gender_score = 0
    score += gender_score

    # Age scoring (prefer 22-60, penalize outside)
    if 22 <= applicant["age"] <= 60:
        age_score = 10
    else:
        age_score = -5
    score += age_score

    return score

def analyze_scoring_logic():
    print("\nScoring Logic Analysis:")
    print("- Education: Higher degrees get higher scores (High School:10, Bachelor:20, Master:30, PhD:40).")
    print("- Experience: 2 points per year, up to 20 years (max 40 points).")
    print("- Gender: No points given, so no bias here.")
    print("- Age: 10 points for ages 22-60, -5 otherwise (may disadvantage very young or older applicants).")
    print("\nPotential Biases/Unfair Weightings:")
    print("- Education: May disadvantage skilled applicants without formal degrees.")
    print("- Age: Penalizes applicants outside 22-60, which could be ageist.")
    print("- Gender: No explicit bias, but always review for indirect effects.")
    print("- Experience: Caps at 20 years, so very experienced applicants get no extra benefit.")

def main():
    applicant = get_applicant_input()
    score = score_applicant(applicant)
    print(f"\nApplicant Score: {score} (out of a possible 90)")
    analyze_scoring_logic()

if __name__ == "__main__":
    main()
```

OUTPUT:



```
nter applicant details:
ducation (High School/Bachelor/Master/PhD): Bachelor
ears of Experience: 2
ender (Male/Female/Other): Female
ge: 19

pplicant Score: 19 (out of a possible 90)

coring Logic Analysis:
 Education: Higher degrees get higher scores (High School:10, Bachelor:20, Master:30, PhD:40).
 Experience: 2 points per year, up to 20 years (max 40 points).
 Gender: No points given, so no bias here.
 Age: 10 points for ages 22-60, -5 otherwise (may disadvantage very young or older applicants).

otential Biases/Unfair Weightings:
 Education: May disadvantage skilled applicants without formal degrees.
 Age: Penalizes applicants outside 22-60, which could be ageist.
 Gender: No explicit bias, but always review for indirect effects.
 Experience: Caps at 20 years, so very experienced applicants get no extra benefit.
```

CODE EXPLANATION:

☐ **get_applicant_input()** → takes applicant details (education, experience, gender, age).

☐ **score_applicant()** → calculates score based on education, experience, and age (gender = neutral).

☐ **analyze_scoring_logic()** → explains how scoring works and possible biases.

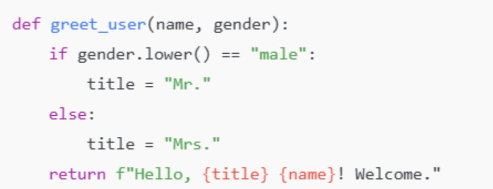
☐ **main()** → collects input, scores applicant, shows result, and explains scoring logic.

Expected Output#4

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

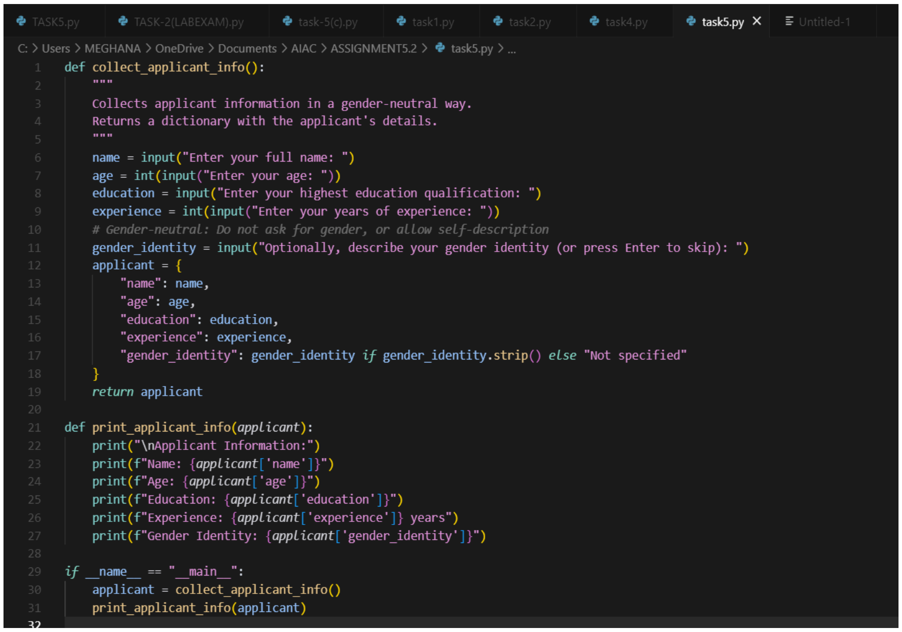
Task Description#5 (Inclusiveness)

- Code Snippet



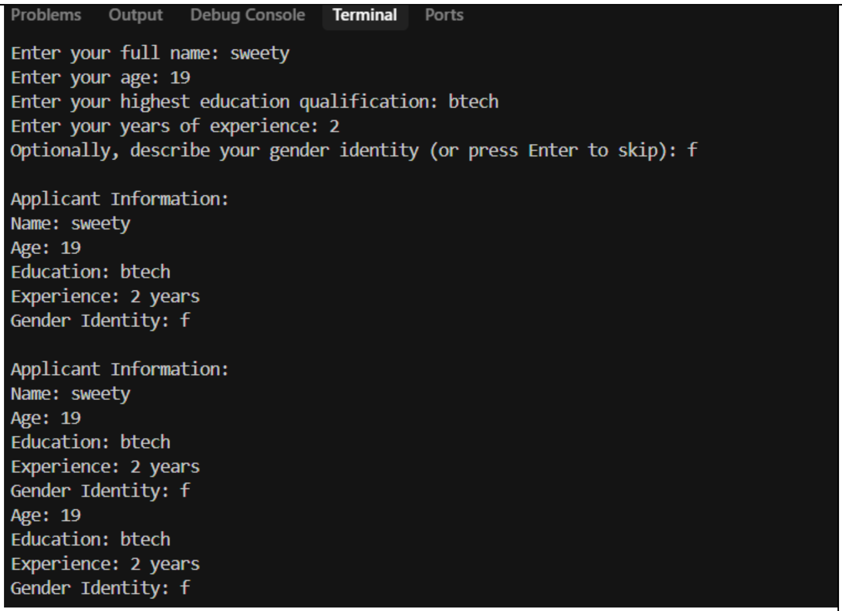
```
def greet_user(name, gender):
    if gender.lower() == "male":
        title = "Mr."
    else:
        title = "Mrs."
    return f"Hello, {title} {name}! Welcome."
```

PROMPT: Write a python code that generate code that includes gender-neutral


TASK5.py
TASK-2(LABEXAM).py
task-5(c).py
task1.py
task2.py
task4.py
task5.py
Untitled-1
C: > Users > MEGHANA > OneDrive > Documents > AIAC > ASSIGNMENT5.2 > task5.py > ...

```
def collect_applicant_info():
    """
    Collects applicant information in a gender-neutral way.
    Returns a dictionary with the applicant's details.
    """
    name = input("Enter your full name: ")
    age = int(input("Enter your age: "))
    education = input("Enter your highest education qualification: ")
    experience = int(input("Enter your years of experience: "))
    # Gender-neutral: Do not ask for gender, or allow self-description
    gender_identity = input("Optionally, describe your gender identity (or press Enter to skip): ")
    applicant = {
        "name": name,
        "age": age,
        "education": education,
        "experience": experience,
        "gender_identity": gender_identity if gender_identity.strip() else "Not specified"
    }
    return applicant

def print_applicant_info(applicant):
    print("\nApplicant Information:")
    print(f"Name: {applicant['name']}")
    print(f"Age: {applicant['age']}")
    print(f"Education: {applicant['education']}")
    print(f"Experience: {applicant['experience']} years")
    print(f"Gender Identity: {applicant['gender_identity']}")

if __name__ == "__main__":
    applicant = collect_applicant_info()
    print_applicant_info(applicant)
```

OUTPUT:



```
Enter your full name: sweety
Enter your age: 19
Enter your highest education qualification: btech
Enter your years of experience: 2
Optionally, describe your gender identity (or press Enter to skip): f

Applicant Information:
Name: sweety
Age: 19
Education: btech
Experience: 2 years
Gender Identity: f

Applicant Information:
Name: sweety
Age: 19
Education: btech
Experience: 2 years
Gender Identity: f
Age: 19
Education: btech
Experience: 2 years
Gender Identity: f
```

CODE EXPLANATION:

collect_applicant_info() : asks the user for details (name, age, education, experience).

- Gender question is optional and allows free input (or skips if left blank).
- Stores all info in a dictionary and returns it.

☐ print_applicant_info() : neatly prints the collected details.
☐ main part : runs the program: collects applicant info, then displays it.

Expected Output#5

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

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	0.5
Bias	1.0
Inclusiveness	0.5
Data security and Privacy	0.5
Total	**2.5 Marks**