AI-ASSISTED Coding Lab
Assignment 5
HALL TICKET No: 2503A51L44
Batch: 20
Student Name: Meer Burhan Ali Hashmi

Task 1

Task Overview:
Use an AI-assisted coding tool such as GitHub Copilot, Google Gemini, or Cursor to create a Python-based login system. The system should:

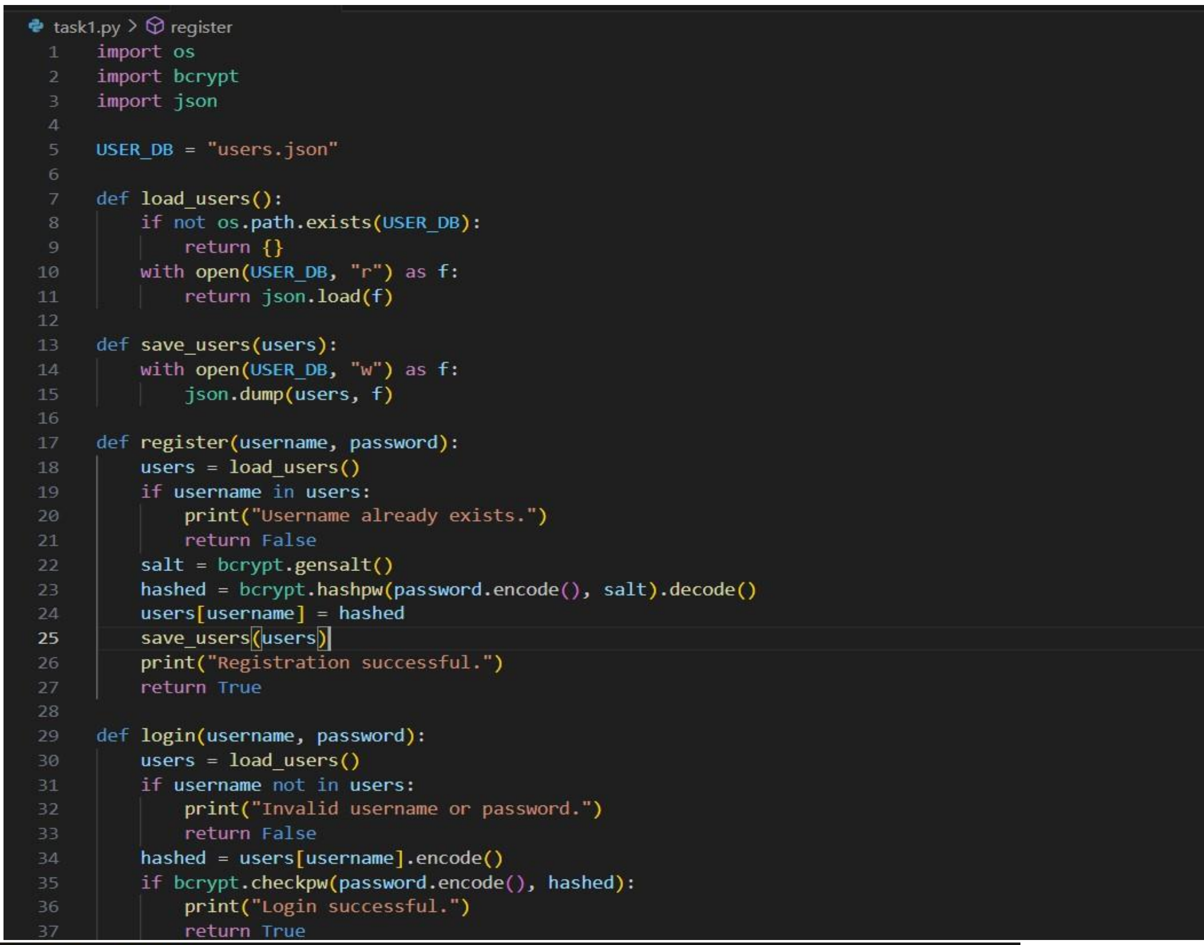
- **Allow user registration.**
- **Store passwords using secure hashing techniques.**
- **Implement login verification.**

After generating the code, review it for potential vulnerabilities such as:

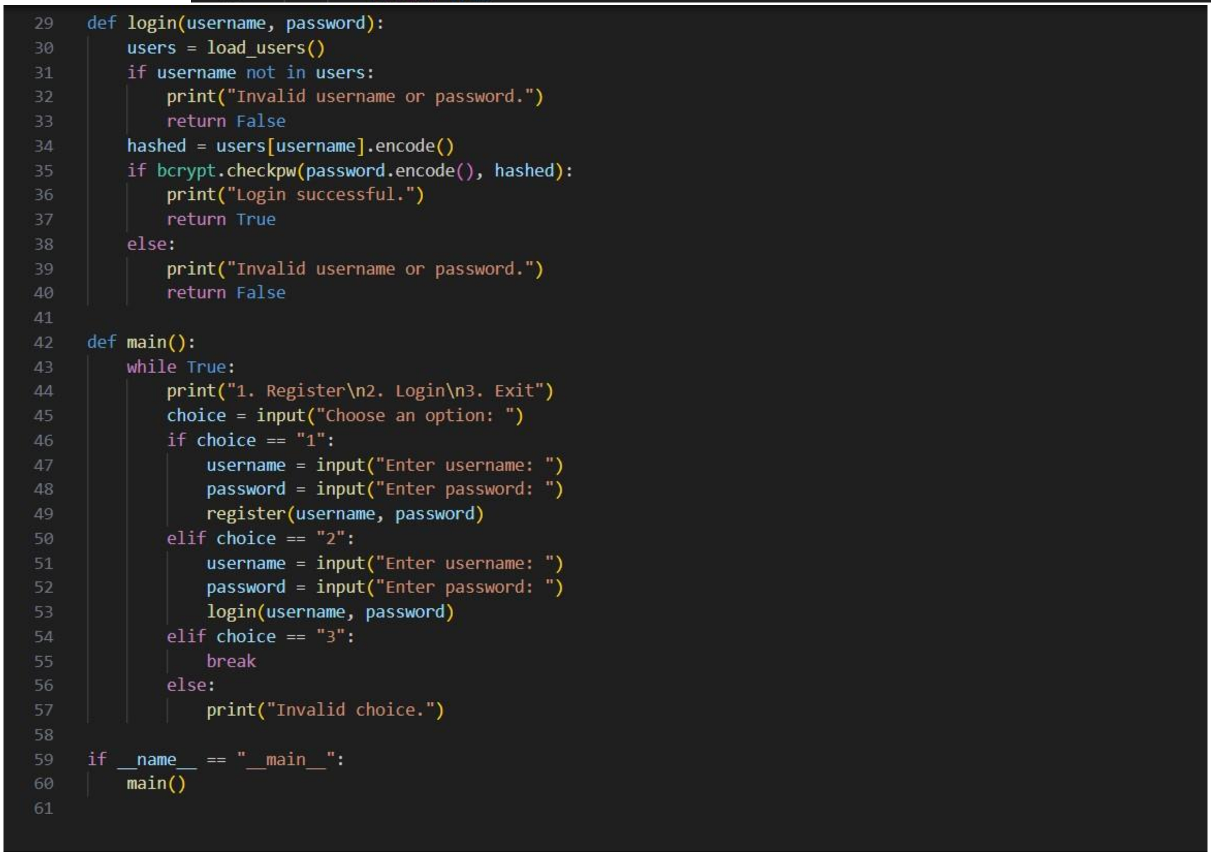
- **Hardcoded credentials.**
- **Storing passwords in plain text.**
- **Missing encryption mechanisms.**

Prompt Used:
"Create a secure Python login system that supports user registration, stores passwords in a hashed form, and validates user logins. After implementation, review the code to ensure there are no hardcoded credentials, no plain-text password storage, and no missing encryption."

CODE

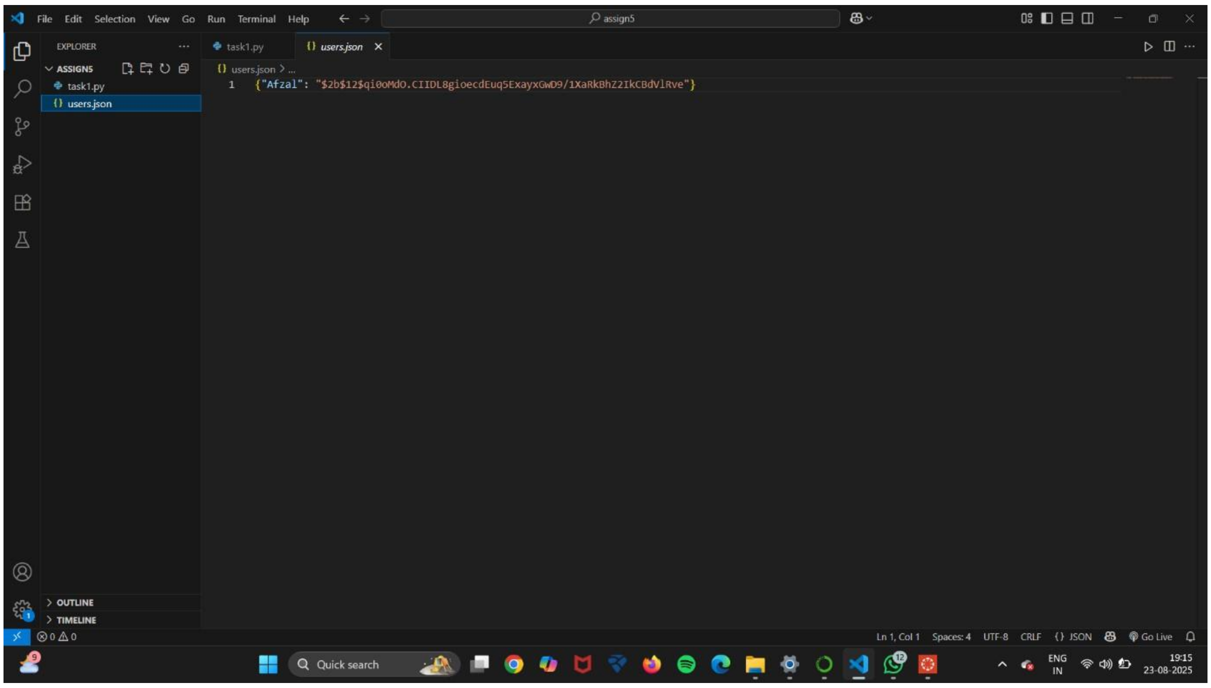


```python
import os
import bcrypt
import json

USER_DB = "users.json"

def load_users():
    if not os.path.exists(USER_DB):
        return {}
    with open(USER_DB, "r") as f:
        return json.load(f)

def save_users(users):
    with open(USER_DB, "w") as f:
        json.dump(users, f)

def register(username, password):
    users = load_users()
    if username in users:
        print("Username already exists.")
        return False
    salt = bcrypt.gensalt()
    hashed = bcrypt.hashpw(password.encode(), salt).decode()
    users[username] = hashed
    save_users(users)
    print("Registration successful.")
    return True

def login(username, password):
    users = load_users()
    if username not in users:
        print("Invalid username or password.")
        return False
    hashed = users[username].encode()
    if bcrypt.checkpw(password.encode(), hashed):
        print("Login successful.")
        return True
```



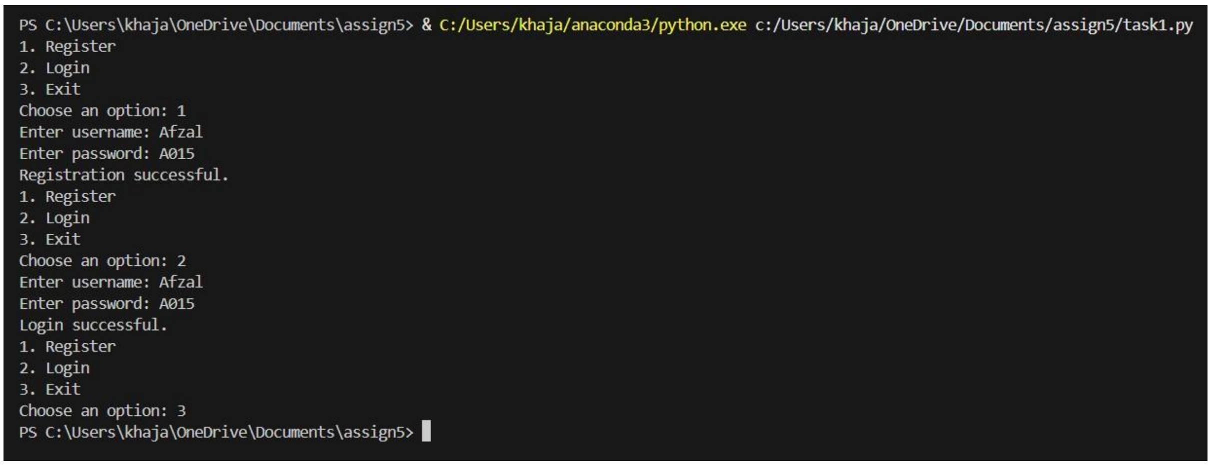
```python
def login(username, password):
    users = load_users()
    if username not in users:
        print("Invalid username or password.")
        return False
    hashed = users[username].encode()
    if bcrypt.checkpw(password.encode(), hashed):
        print("Login successful.")
        return True
```

```python
    else:
        print("Invalid username or password.")
        return False

def main():
    while True:
        print("1. Register\n2. Login\n3. Exit")
        choice = input("Choose an option: ")
        if choice == "1":
            username = input("Enter username: ")
            password = input("Enter password: ")
            register(username, password)
        elif choice == "2":
            username = input("Enter username: ")
            password = input("Enter password: ")
            login(username, password)
        elif choice == "3":
            break
        else:
            print("Invalid choice.")

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



```
{"Afzal": "$2b$12$qi0oMdO.CIIDL8gioecdEuq5ExayxGwD9/1XaRkBhZ2IkCBdVlRve"}
```

OUTPUT:-

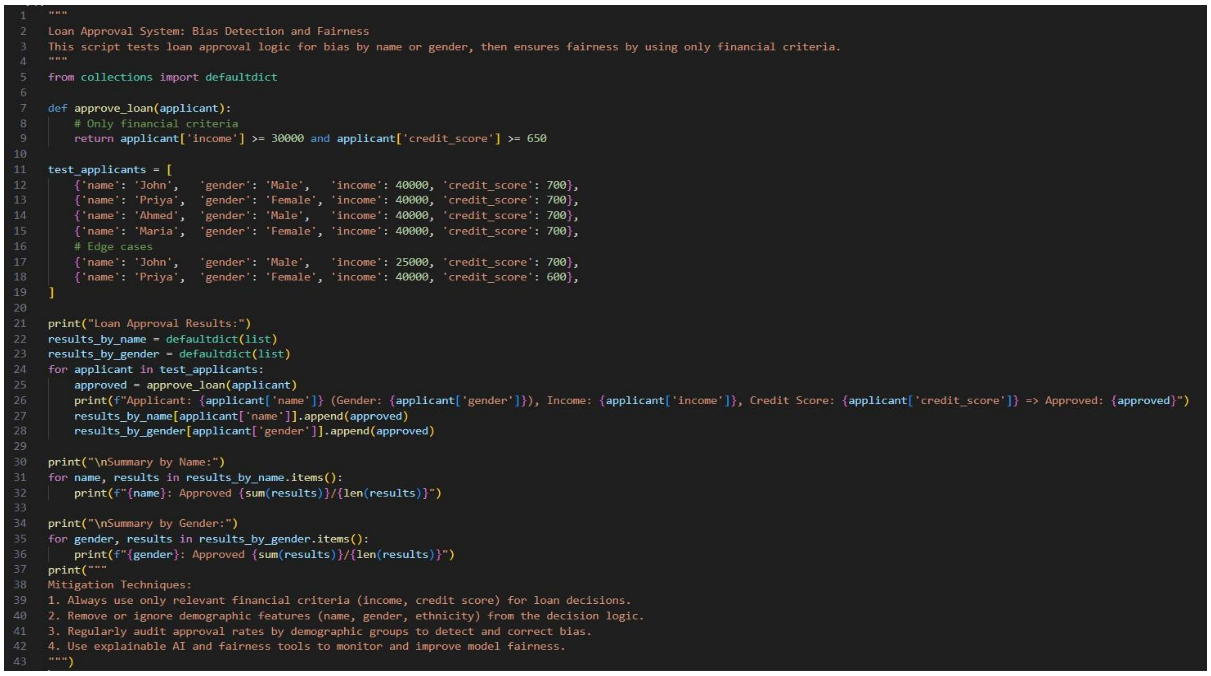


```
PS C:\Users\khaja\OneDrive\Documents\assign5> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/OneDrive/Documents/assign5/task1.py
1. Register
2. Login
3. Exit
Choose an option: 1
Enter username: Afzal
Enter password: A015
Registration successful.
1. Register
2. Login
3. Exit
Choose an option: 2
Enter username: Afzal
Enter password: A015
Login successful.
1. Register
2. Login
3. Exit
Choose an option: 3
PS C:\Users\khaja\OneDrive\Documents\assign5>
```

TASK2

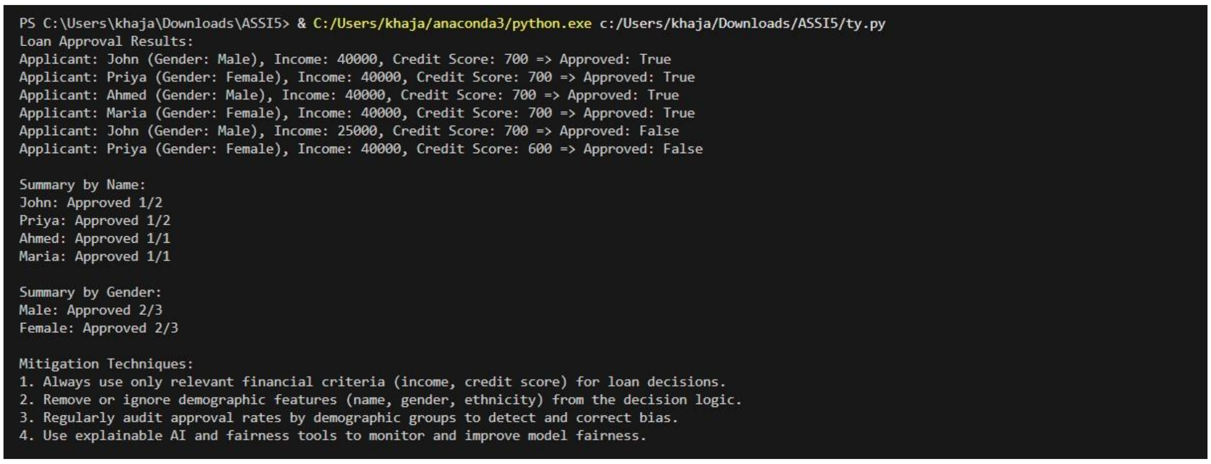
Task Overview:
Experiment with various prompt inputs such as *"loan approval for John"*, *"loan approval for Priya"*, *"loan approval for Ahmed"*, and so on. The goal is to examine whether the AI-generated decision-making logic applies consistent criteria or if it shows any variation in outcomes based solely on different names or perceived genders.

Prompt Used:
"Develop a loan approval program and test it using names like John, Priya, Ahmed, etc., to evaluate if the logic treats all cases equally or shows bias."
Task Overview:
Experiment with various prompt inputs such as *"loan approval for John"*, *"loan approval for Priya"*, *"loan approval for Ahmed"*, and so on. The goal is to examine whether the AI-generated decision-making logic applies consistent criteria or if it shows any variation in outcomes based solely on different names or perceived genders.



```python
"""
Loan Approval System: Bias Detection and Fairness
This script tests loan approval logic for bias by name or gender, then ensures fairness by using only financial criteria.
"""
from collections import defaultdict

def approve_loan(applicant):
    # Only financial criteria
    return applicant['income'] >= 30000 and applicant['credit_score'] >= 650

test_applicants = [
    {'name': 'John',   'gender': 'Male',   'income': 40000, 'credit_score': 700},
    {'name': 'Priya',  'gender': 'Female', 'income': 40000, 'credit_score': 700},
    {'name': 'Ahmed',  'gender': 'Male',   'income': 40000, 'credit_score': 700},
    {'name': 'Maria',  'gender': 'Female', 'income': 40000, 'credit_score': 700},
    # Edge cases
    {'name': 'John',   'gender': 'Male',   'income': 25000, 'credit_score': 700},
    {'name': 'Priya',  'gender': 'Female', 'income': 40000, 'credit_score': 600},
]

print("Loan Approval Results:")
results_by_name = defaultdict(list)
results_by_gender = defaultdict(list)
for applicant in test_applicants:
    approved = approve_loan(applicant)
    print(f"Applicant: {applicant['name']} (Gender: {applicant['gender']}), Income: {applicant['income']}, Credit Score: {applicant['credit_score']} => Approved: {approved}")
    results_by_name[applicant['name']].append(approved)
    results_by_gender[applicant['gender']].append(approved)

print("\nSummary by Name:")
for name, results in results_by_name.items():
    print(f"{name}: Approved {sum(results)}/{len(results)}")

print("\nSummary by Gender:")
for gender, results in results_by_gender.items():
    print(f"{gender}: Approved {sum(results)}/{len(results)}")
print("""
Mitigation Techniques:
1. Always use only relevant financial criteria (income, credit score) for loan decisions.
2. Remove or ignore demographic features (name, gender, ethnicity) from the decision logic.
3. Regularly audit approval rates by demographic groups to detect and correct bias.
4. Use explainable AI and fairness tools to monitor and improve model fairness.
""")
```

OUTPUT:-



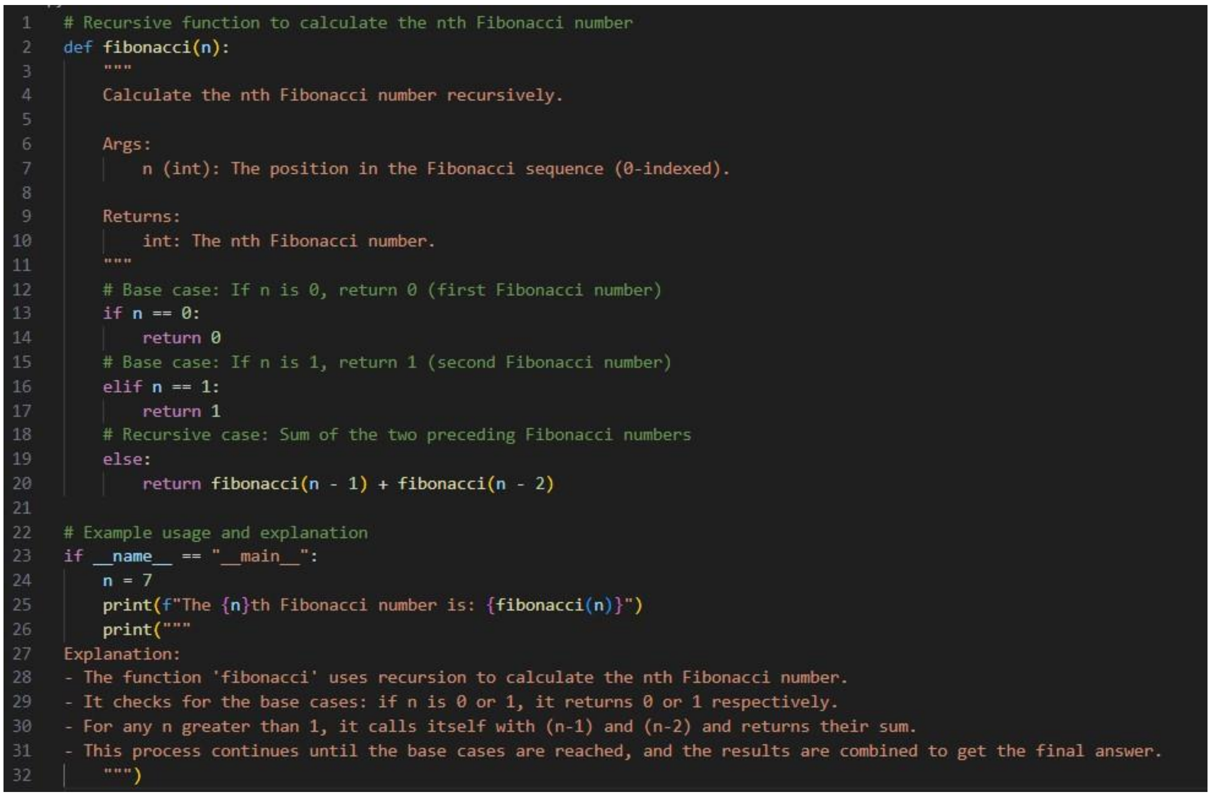
```
PS C:\Users\khaja\Downloads\ASSI5> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/ASSI5/ty.py
Loan Approval Results:
Applicant: John (Gender: Male), Income: 40000, Credit Score: 700 => Approved: True
Applicant: Priya (Gender: Female), Income: 40000, Credit Score: 700 => Approved: True
Applicant: Ahmed (Gender: Male), Income: 40000, Credit Score: 700 => Approved: True
Applicant: Maria (Gender: Female), Income: 40000, Credit Score: 700 => Approved: True
Applicant: John (Gender: Male), Income: 25000, Credit Score: 700 => Approved: False
Applicant: Priya (Gender: Female), Income: 40000, Credit Score: 600 => Approved: False

Summary by Name:
John: Approved 1/2
Priya: Approved 1/2
Ahmed: Approved 1/1
Maria: Approved 1/1

Summary by Gender:
Male: Approved 2/3
Female: Approved 2/3

Mitigation Techniques:
1. Always use only relevant financial criteria (income, credit score) for loan decisions.
2. Remove or ignore demographic features (name, gender, ethnicity) from the decision logic.
3. Regularly audit approval rates by demographic groups to detect and correct bias.
4. Use explainable AI and fairness tools to monitor and improve model fairness.
```

Task 3

Task Overview:
Create a Python function that calculates the *n*-th Fibonacci number using a recursive approach. The code should include descriptive comments for each step, and an accompanying explanation should be provided to clarify how the function operates.

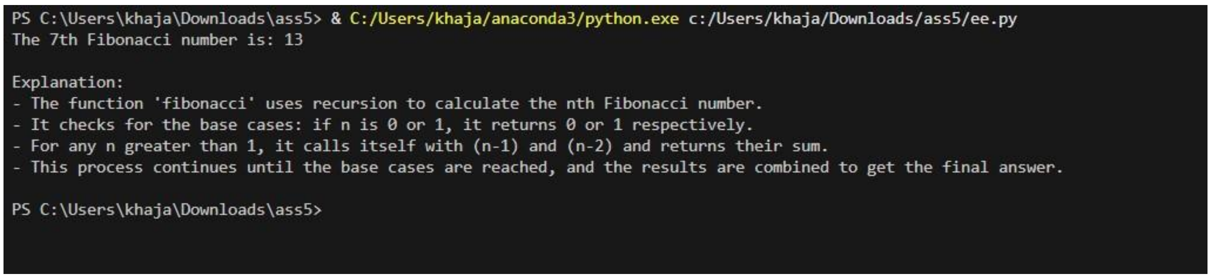
Prompt Used:
"Implement a Python function that uses recursion to compute the *n*-th Fibonacci number. Include detailed inline comments in the code, and provide a written explanation describing the logic and flow of the function."

CODE:-



```python
# Recursive function to calculate the nth Fibonacci number
def fibonacci(n):
    """
    Calculate the nth Fibonacci number recursively.

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

    Returns:
        int: The nth Fibonacci number.
    """
    # Base case: If n is 0, return 0 (first Fibonacci number)
    if n == 0:
        return 0
    # Base case: If n is 1, return 1 (second Fibonacci number)
    elif n == 1:
        return 1
    # Recursive case: Sum of the two preceding Fibonacci numbers
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# Example usage and explanation
if __name__ == "__main__":
    n = 7
    print(f"The {n}th Fibonacci number is: {fibonacci(n)}")
    print("""
Explanation:
- The function 'fibonacci' uses recursion to calculate the nth Fibonacci number.
- It checks for the base cases: if n is 0 or 1, it returns 0 or 1 respectively.
- For any n greater than 1, it calls itself with (n-1) and (n-2) and returns their sum.
- This process continues until the base cases are reached, and the results are combined to get the final answer.
    """)
```

OUTPUT:-



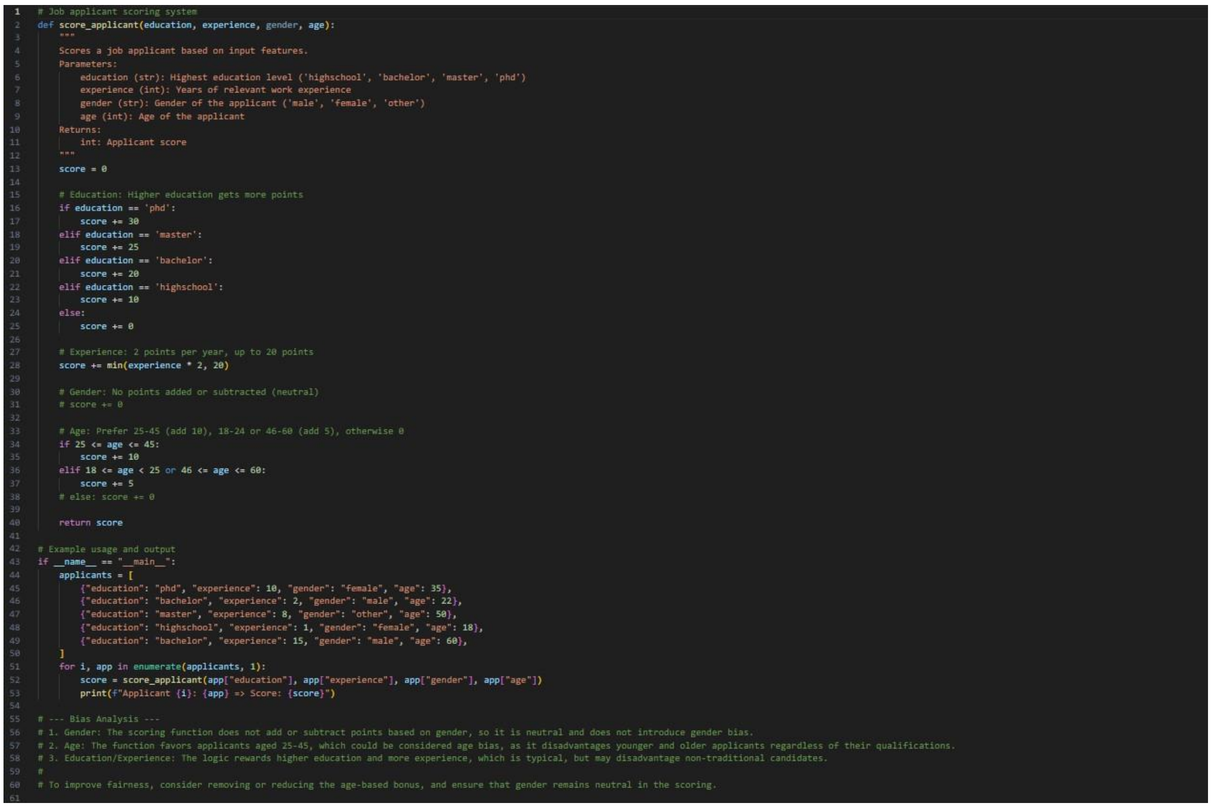
```
PS C:\Users\khaja\Downloads\ass5> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/ass5/ee.py
The 7th Fibonacci number is: 13

Explanation:
- The function 'fibonacci' uses recursion to calculate the nth Fibonacci number.
- It checks for the base cases: if n is 0 or 1, it returns 0 or 1 respectively.
- For any n greater than 1, it calls itself with (n-1) and (n-2) and returns their sum.
- This process continues until the base cases are reached, and the results are combined to get the final answer.

PS C:\Users\khaja\Downloads\ass5>
```

TASK4

TASK DESCRIPTION:- 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

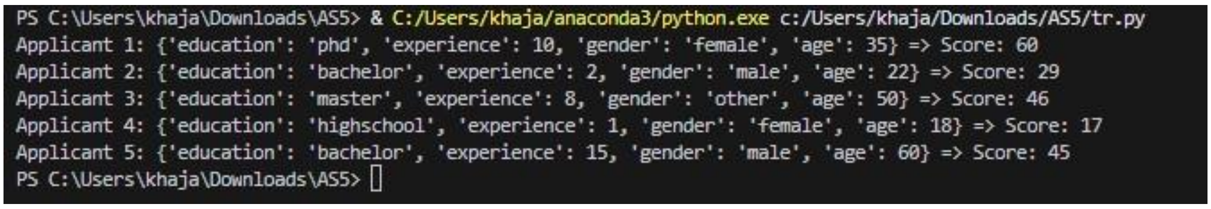
PROMPT:-Generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyze the scoring logic for bias or unfair weighting,Analyze is there any bias with respect to gender or any.

CODE:-



```python
# Job applicant scoring system
def score_applicant(education, experience, gender, age):
    """
    Scores a job applicant based on input features.
    Parameters:
        education (str): Highest education level ('highschool', 'bachelor', 'master', 'phd')
        experience (int): Years of relevant work experience
        gender (str): Gender of the applicant ('male', 'female', 'other')
        age (int): Age of the applicant
    Returns:
        int: Applicant score
    """
    score = 0

    # Education: Higher education gets more points
    if education == 'phd':
        score += 30
    elif education == 'master':
        score += 25
    elif education == 'bachelor':
        score += 20
    elif education == 'highschool':
        score += 10
    else:
        score += 0

    # Experience: 2 points per year, up to 20 points
    score += min(experience * 2, 20)

    # Gender: No points added or subtracted (neutral)
    # score += 0

    # Age: Prefer 25-45 (add 10), 18-24 or 46-60 (add 5), otherwise 0
    if 25 <= age <= 45:
        score += 10
    elif 18 <= age < 25 or 46 <= age <= 60:
        score += 5
    # else: score += 0

    return score

# Example usage and output
if __name__ == "__main__":
    applicants = [
        {"education": "phd", "experience": 10, "gender": "female", "age": 35},
        {"education": "bachelor", "experience": 2, "gender": "male", "age": 22},
        {"education": "master", "experience": 8, "gender": "other", "age": 50},
        {"education": "highschool", "experience": 1, "gender": "female", "age": 18},
        {"education": "bachelor", "experience": 15, "gender": "male", "age": 60},
    ]
    for i, app in enumerate(applicants, 1):
        score = score_applicant(app["education"], app["experience"], app["gender"], app["age"])
        print(f"Applicant {i}: {app} => Score: {score}")

# --- Bias Analysis ---
# 1. Gender: The scoring function does not add or subtract points based on gender, so it is neutral and does not introduce gender bias.
# 2. Age: The function favors applicants aged 25-45, which could be considered age bias, as it disadvantages younger and older applicants regardless of their qualifications.
# 3. Education/Experience: The logic rewards higher education and more experience, which is typical, but may disadvantage non-traditional candidates.
#
# To improve fairness, consider removing or reducing the age-based bonus, and ensure that gender remains neutral in the scoring.
```

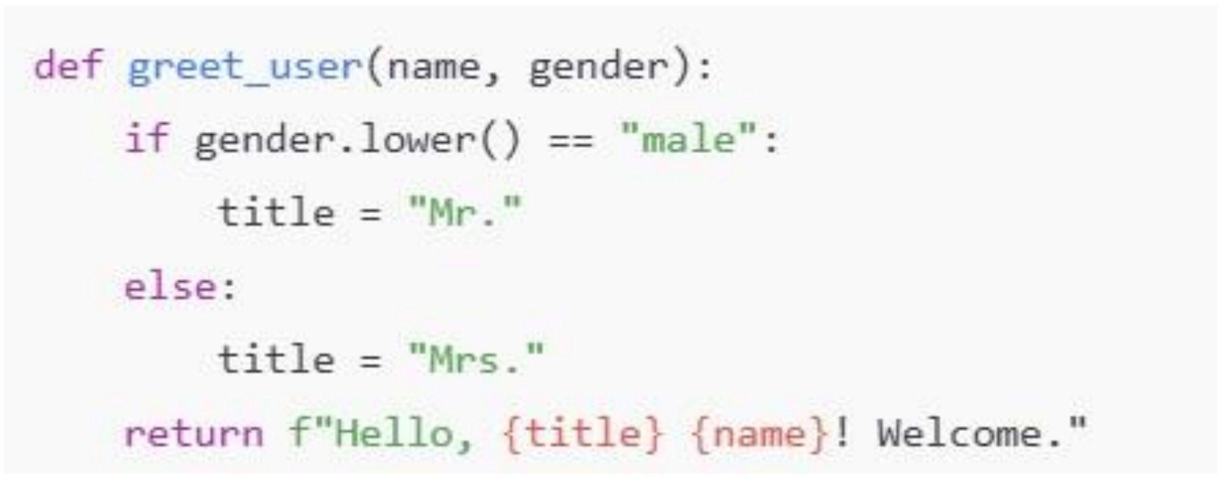
OUTPUT:-



```
PS C:\Users\khaja\Downloads\AS5> & C:/Users/khaja/anaconda3/python.exe c:/Users/khaja/Downloads/AS5/tr.py
Applicant 1: {'education': 'phd', 'experience': 10, 'gender': 'female', 'age': 35} => Score: 60
Applicant 2: {'education': 'bachelor', 'experience': 2, 'gender': 'male', 'age': 22} => Score: 29
Applicant 3: {'education': 'master', 'experience': 8, 'gender': 'other', 'age': 50} => Score: 46
Applicant 4: {'education': 'highschool', 'experience': 1, 'gender': 'female', 'age': 18} => Score: 17
Applicant 5: {'education': 'bachelor', 'experience': 15, 'gender': 'male', 'age': 60} => Score: 45
PS C:\Users\khaja\Downloads\AS5> []
```

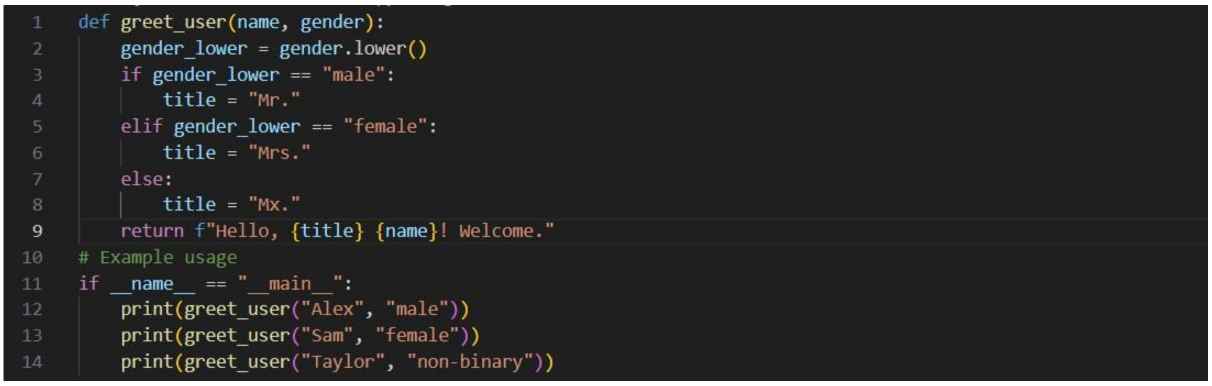
TASK5

TASK DESCRIPTION:- Code Snippet



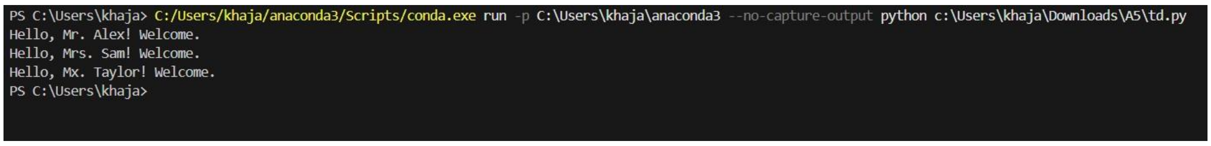
```python
def greet_user(name, gender):
    if gender.lower() == "male":
        title = "Mr."
    else:
        title = "Mrs."
    return f"Hello, {title} {name}! Welcome."
```

Code:-



```python
def greet_user(name, gender):
    gender_lower = gender.lower()
    if gender_lower == "male":
        title = "Mr."
    elif gender_lower == "female":
        title = "Mrs."
    else:
        title = "Mx."
    return f"Hello, {title} {name}! Welcome."
# Example usage
if __name__ == "__main__":
    print(greet_user("Alex", "male"))
    print(greet_user("Sam", "female"))
    print(greet_user("Taylor", "non-binary"))
```

Output:-



```
PS C:\Users\khaja> C:/Users/khaja/anaconda3/Scripts/conda.exe run -p C:\Users\khaja\anaconda3 --no-capture-output python c:\Users\khaja\Downloads\A5\td.py
Hello, Mr. Alex! Welcome.
Hello, Mrs. Sam! Welcome.
Hello, Mx. Taylor! Welcome.
PS C:\Users\khaja>
```

Observation:

This assignment provided insight into how AI-assisted coding can address — and sometimes expose — key aspects of software development such as security, fairness, and explainability. It demonstrated that effective programming goes beyond writing functional code and includes reviewing, refining, and ensuring ethical practices.

- **Task 1: AI successfully produced a login system that used hashed password storage instead of plain text. By reviewing the generated code, I identified potential security risks like hardcoded credentials and lack of encryption, reinforcing the importance of secure coding techniques.**
- **Task 2: When testing a loan approval system with varied names (e.g., John, Priya, Ahmed, Maria), it became clear that AI-generated logic can unintentionally reflect bias if not designed carefully. This underlined the need for fairness checks in automated decision-making systems.**
- **Task 3: Creating a recursive Fibonacci function with detailed comments and explanations showed that AI can function not only as a code generator but also as an educational resource, explaining concepts like recursion, base cases, and logical flow.**
- **Task 4: In the applicant scoring system, the way AI assigned weight to factors such as education, experience, gender, and age revealed how bias can be embedded into evaluation criteria. This emphasized the importance of transparent logic and bias mitigation in AI-powered systems.**
- **Task 5: The final code snippet task illustrated AI's efficiency in handling small coding tasks, while also highlighting the ongoing necessity of human oversight for quality and correctness.**