

LAB NAME : AI ASSISTED CODING

LAB NUMBER :02

ROLL NO :2503A51L16

BRANCH : CSE

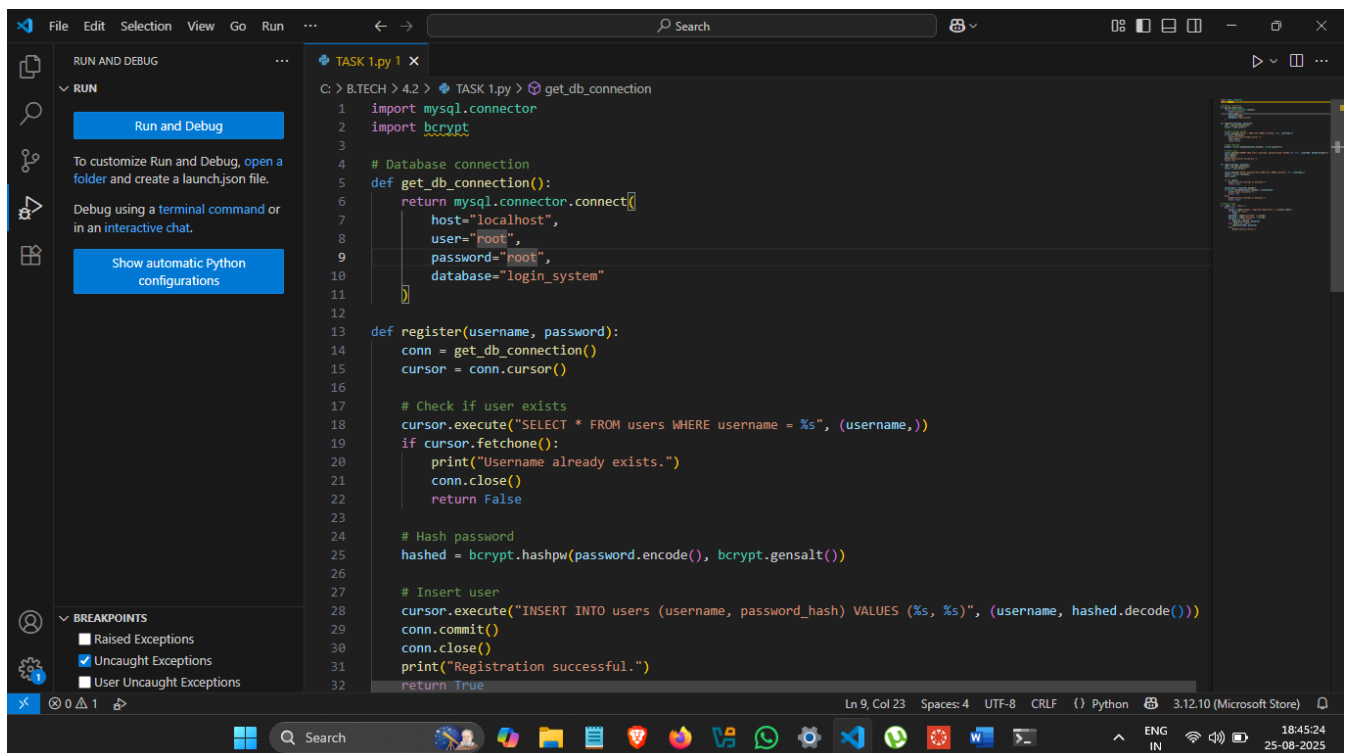
NAME : K.JASHUVA

TASK 1

TASK DESCRIPTION: 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: Generate a secure login system in Python with user registration, hashed password storage, and login verification, then review the code for hardcoded passwords, plain-text storage, or missing encryption.

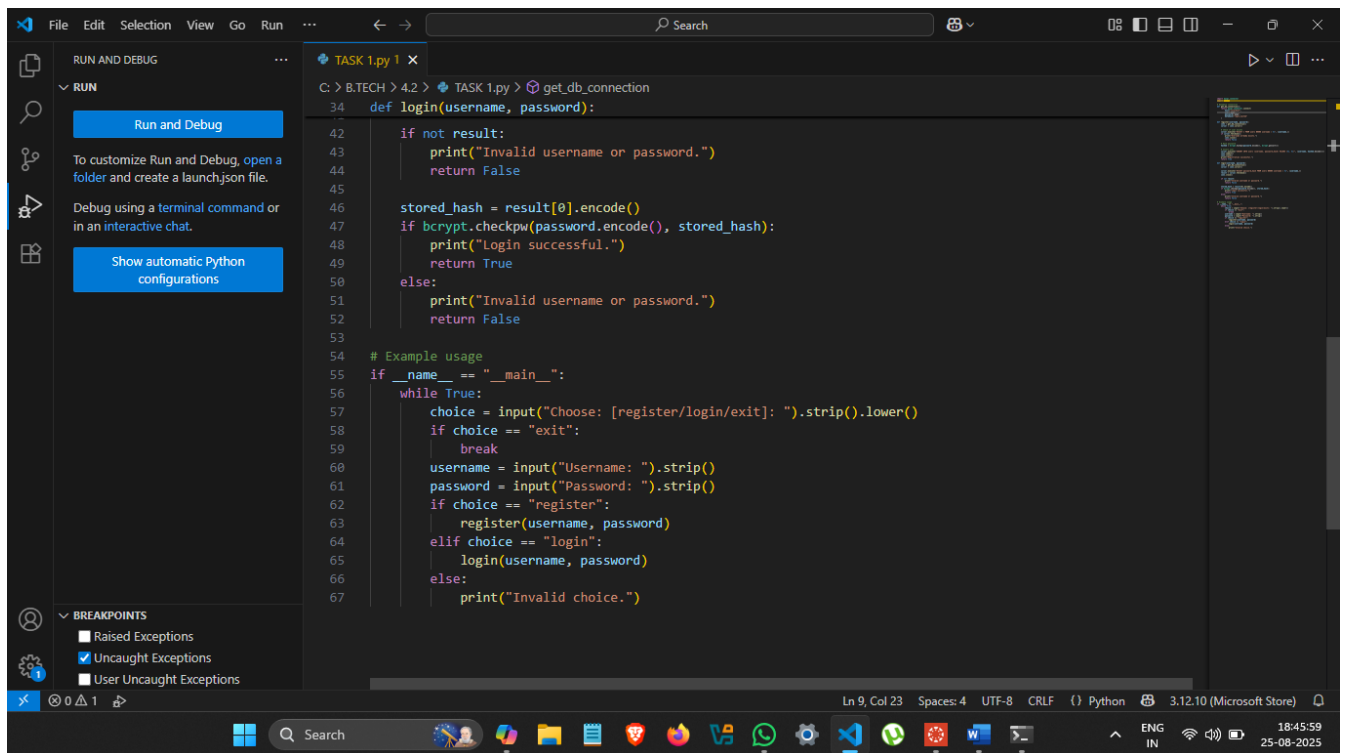
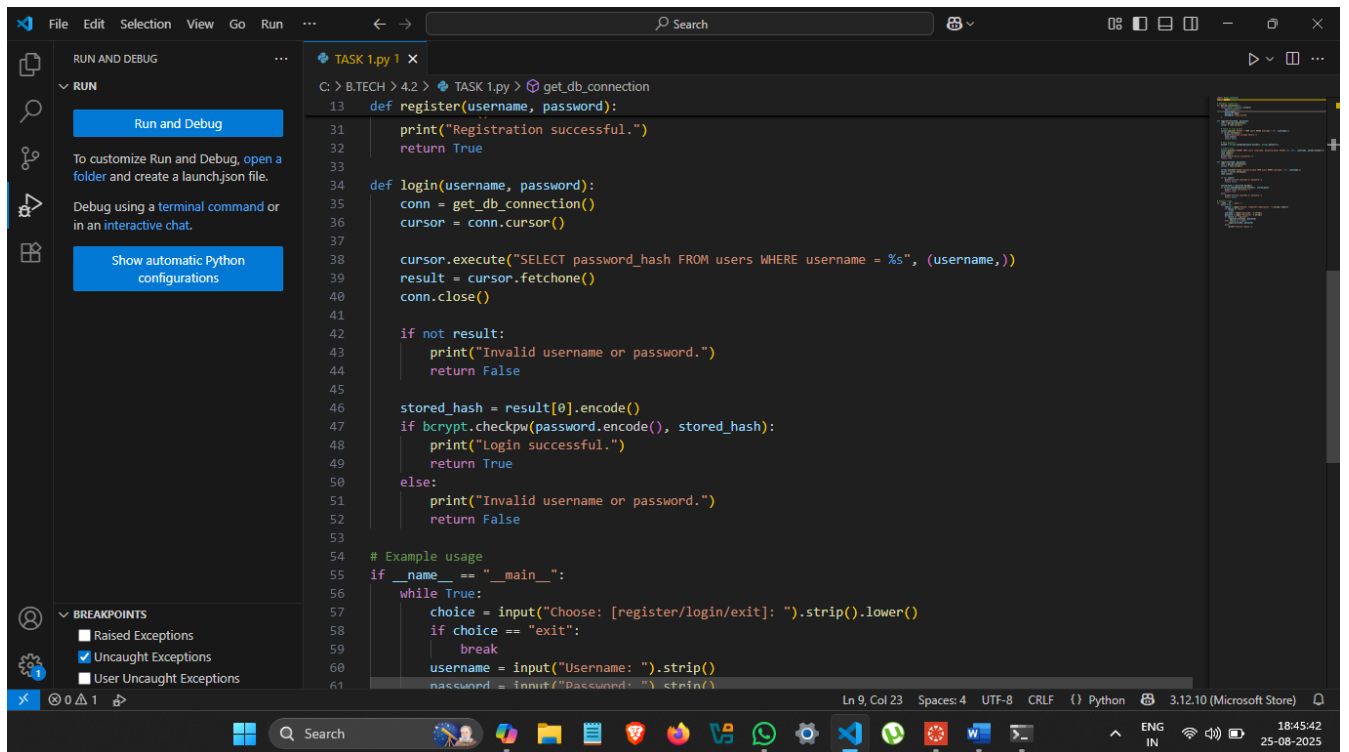
CODE:-



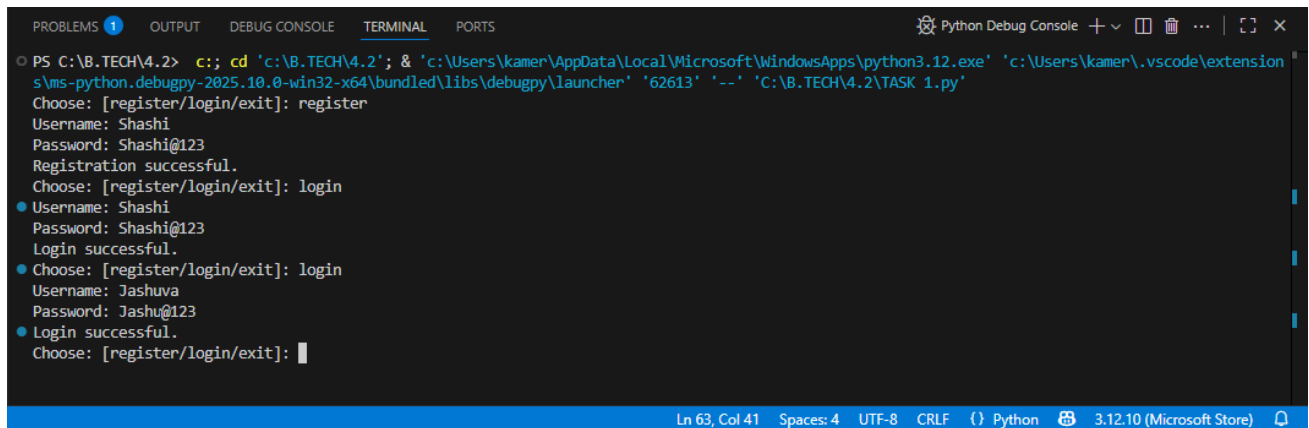
The screenshot shows a code editor with a dark theme. The left sidebar contains a 'RUN AND DEBUG' panel with a 'Run and Debug' button and instructions. Below it is a 'BREAKPOINTS' panel with checkboxes for 'Raised Exceptions', 'Uncaught Exceptions' (checked), and 'User Uncaught Exceptions'. The main editor area displays a Python file named 'TASK 1.py' with the following code:

```
C:\> B.TECH > 4.2 > TASK 1.py > get_db_connection
1 import mysql.connector
2 import bcrypt
3
4 # Database connection
5 def get_db_connection():
6     return mysql.connector.connect(
7         host="localhost",
8         user="root",
9         password="root",
10        database="login_system"
11    )
12
13 def register(username, password):
14     conn = get_db_connection()
15     cursor = conn.cursor()
16
17     # Check if user exists
18     cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
19     if cursor.fetchone():
20         print("Username already exists.")
21         conn.close()
22         return False
23
24     # Hash password
25     hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
26
27     # Insert user
28     cursor.execute("INSERT INTO users (username, password_hash) VALUES (%s, %s)", (username, hashed.decode()))
29     conn.commit()
30     conn.close()
31     print("Registration successful.")
32     return True
```

The status bar at the bottom indicates 'Ln 9, Col 23', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', '3.12.10 (Microsoft Store)', and the system clock shows '18:45:24 25-08-2025'.



OUTPUT:



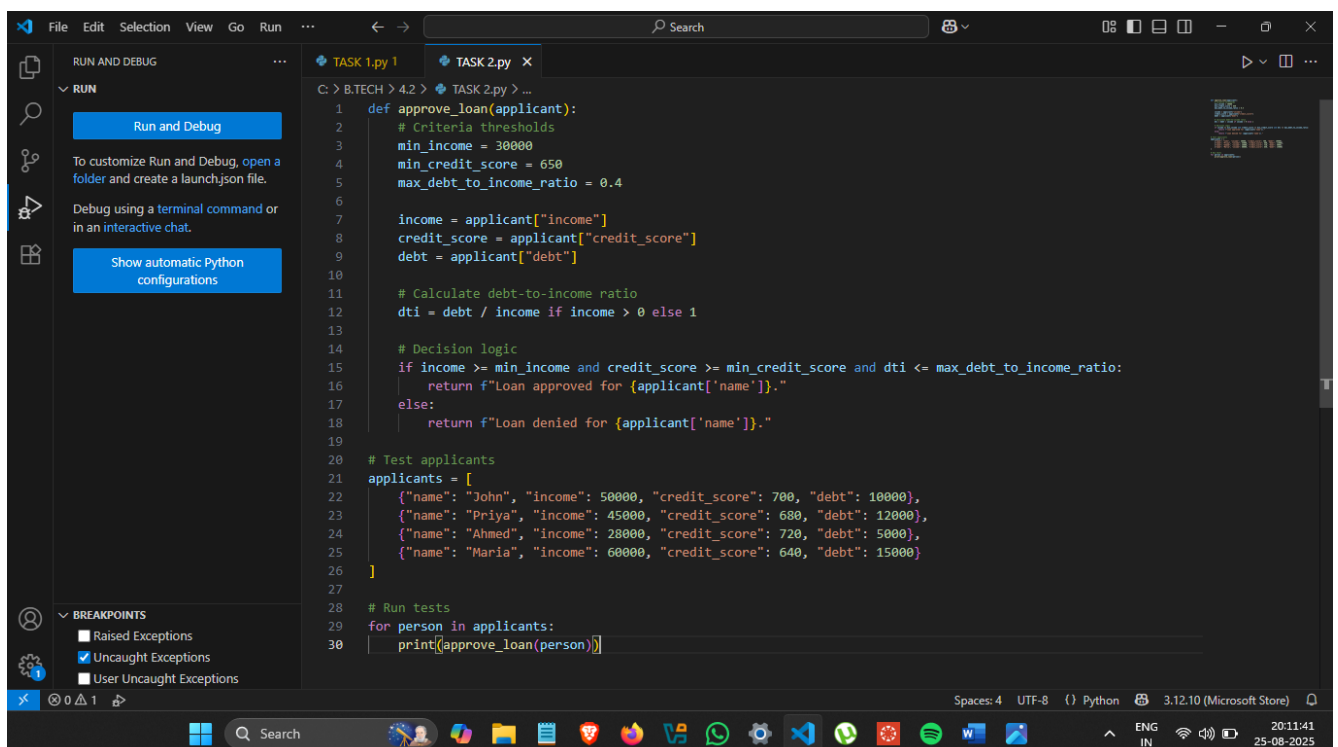
```
PS C:\B.TECH\4.2> c:: cd 'c:\B.TECH\4.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '62613' '--' 'C:\B.TECH\4.2\TASK 1.py'
Choose: [register/login/exit]: register
Username: Shashi
Password: Shashi@123
Registration successful.
Choose: [register/login/exit]: login
Username: Shashi
Password: Shashi@123
Login successful.
Choose: [register/login/exit]: login
Username: Jashuva
Password: Jashu@123
Login successful.
Choose: [register/login/exit]:
```

TASK 2

TASK DESCRIPTION: 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.

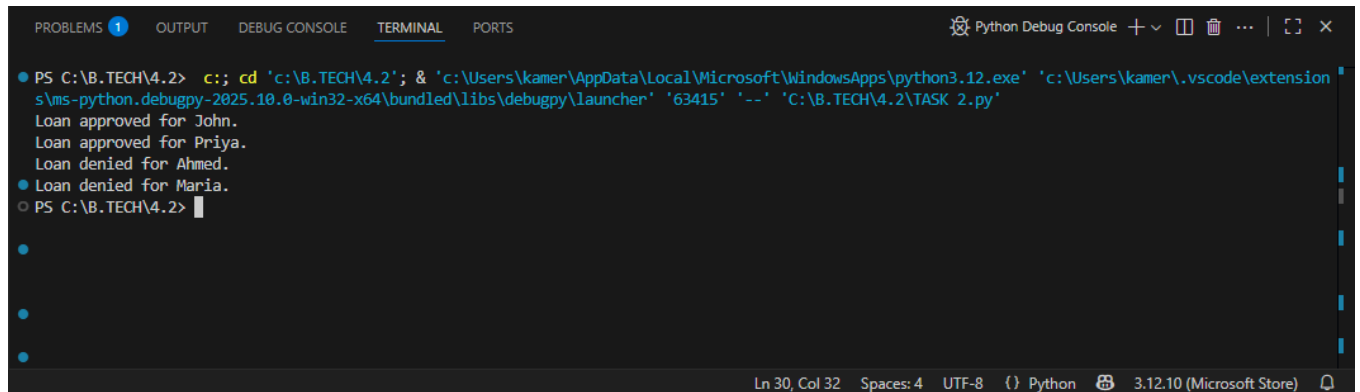
PROMPT: Generate a loan approval system, test it with names like John, Priya, Ahmed, and Maria, check if the logic is biased by name or gender

CODE:



```
1 def approve_loan(applicant):
2     # Criteria thresholds
3     min_income = 30000
4     min_credit_score = 650
5     max_debt_to_income_ratio = 0.4
6
7     income = applicant["income"]
8     credit_score = applicant["credit_score"]
9     debt = applicant["debt"]
10
11     # Calculate debt-to-income ratio
12     dti = debt / income if income > 0 else 1
13
14     # Decision logic
15     if income >= min_income and credit_score >= min_credit_score and dti <= max_debt_to_income_ratio:
16         return f"Loan approved for {applicant['name']}."
17     else:
18         return f"Loan denied for {applicant['name']}."
19
20 # Test applicants
21 applicants = [
22     {"name": "John", "income": 50000, "credit_score": 700, "debt": 10000},
23     {"name": "Priya", "income": 45000, "credit_score": 680, "debt": 12000},
24     {"name": "Ahmed", "income": 28000, "credit_score": 720, "debt": 5000},
25     {"name": "Maria", "income": 60000, "credit_score": 640, "debt": 15000}
26 ]
27
28 # Run tests
29 for person in applicants:
30     print(approve_loan(person))
```

OUTPUT:



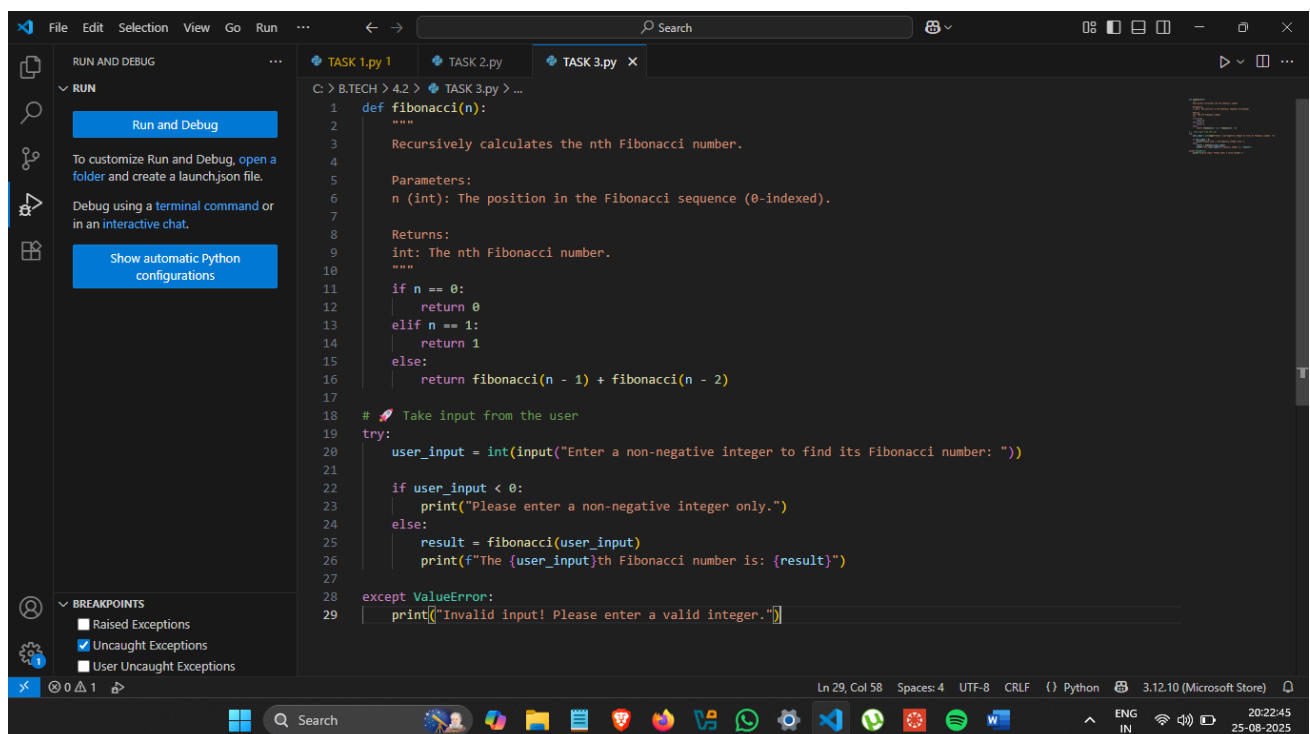
```
PS C:\B.TECH\4.2> c:: cd 'c:\B.TECH\4.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\.vscode\extension
s\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '63415' '--' 'C:\B.TECH\4.2\TASK 2.py'
Loan approved for John.
Loan approved for Priya.
Loan denied for Ahmed.
Loan denied for Maria.
PS C:\B.TECH\4.2>
```

TASK 3

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

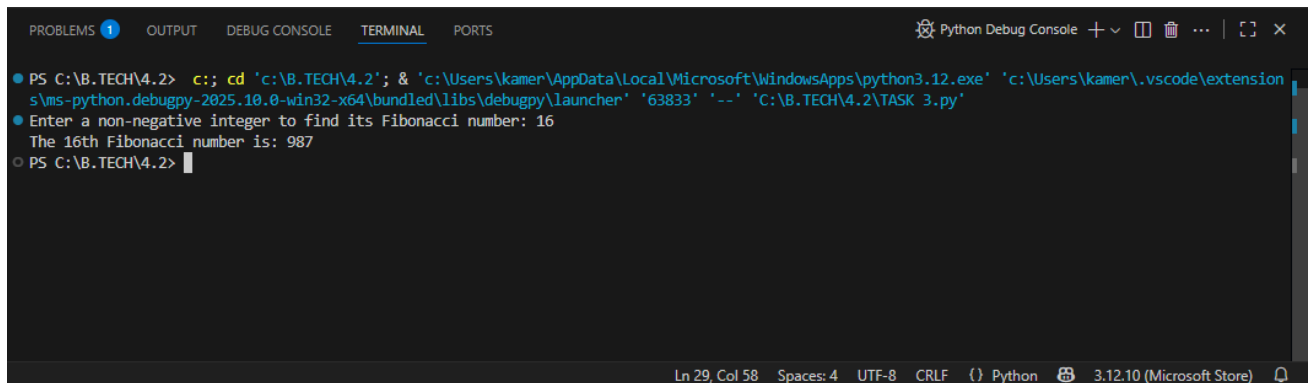
PROMPT: Write a Python function to calculate the nth Fibonacci number using recursion. Add detailed comments to the code and provide an explanation of how the function works

CODE:-



```
1 def fibonacci(n):
2     """
3     Recursively calculates the nth Fibonacci number.
4
5     Parameters:
6     n (int): The position in the Fibonacci sequence (0-indexed).
7
8     Returns:
9     int: The nth Fibonacci number.
10    """
11    if n == 0:
12        return 0
13    elif n == 1:
14        return 1
15    else:
16        return fibonacci(n - 1) + fibonacci(n - 2)
17
18    # Take input from the user
19    try:
20        user_input = int(input("Enter a non-negative integer to find its Fibonacci number: "))
21
22        if user_input < 0:
23            print("Please enter a non-negative integer only.")
24        else:
25            result = fibonacci(user_input)
26            print(f"The {user_input}th Fibonacci number is: {result}")
27
28    except ValueError:
29        print("Invalid input! Please enter a valid integer.")
```

OUTPUT:



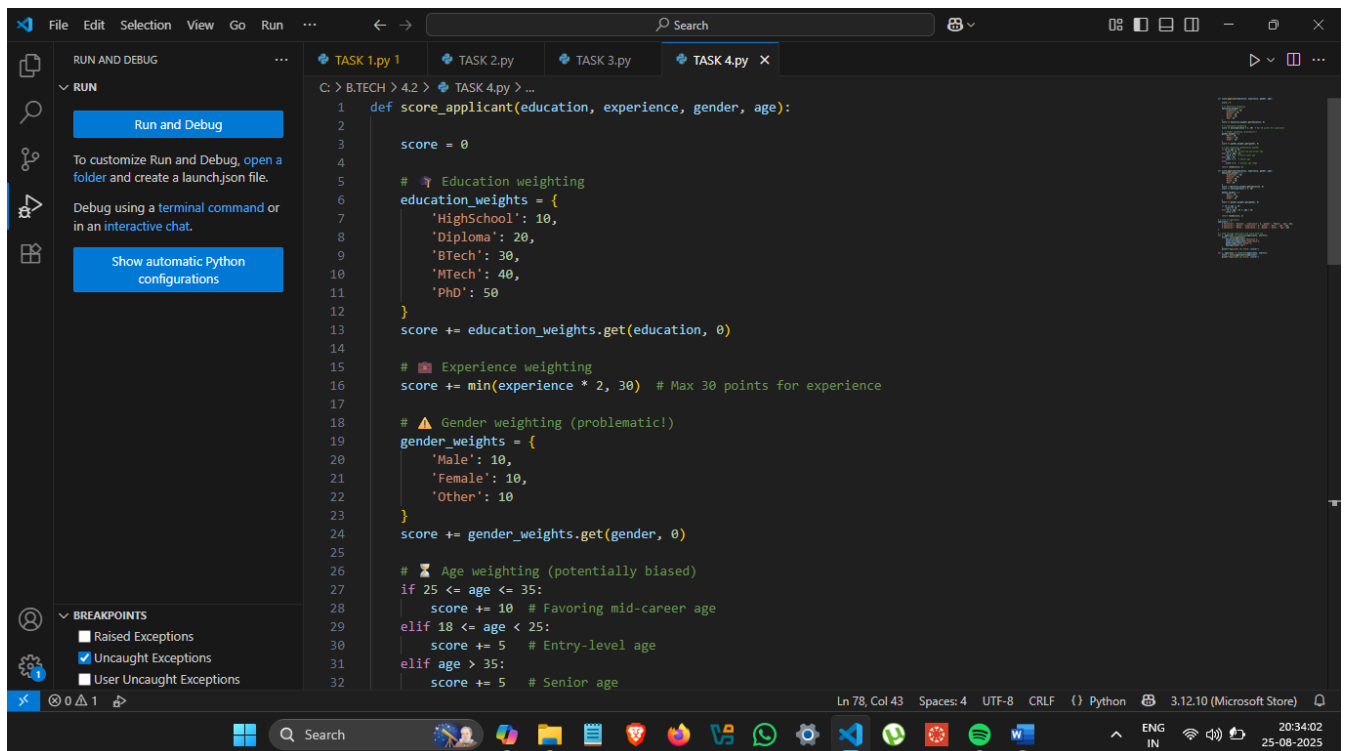
```
PS C:\B.TECH\4.2> c:: cd 'c:\B.TECH\4.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\.vscode\extension
s\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '63833' '--' 'C:\B.TECH\4.2\TASK 3.py'
Enter a non-negative integer to find its Fibonacci number: 16
The 16th Fibonacci number is: 987
PS C:\B.TECH\4.2> |
```

TASK 4

TASK DESCRIPTION: Ask to generate a job applicant scoring system based on input features (e.g., education, experience, gender, age). Analyse 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). Analyse the scoring logic for bias or unfair weighting, Analyse is there any bias with respect to gender or any.

CODE:



```
C:\B.TECH\4.2> TASK 4.py ...
1 def score_applicant(education, experience, gender, age):
2
3     score = 0
4
5     # Education weighting
6     education_weights = {
7         'HighSchool': 10,
8         'Diploma': 20,
9         'BTech': 30,
10        'MTech': 40,
11        'PHD': 50
12    }
13     score += education_weights.get(education, 0)
14
15     # Experience weighting
16     score += min(experience * 2, 30) # Max 30 points for experience
17
18     # Gender weighting (problematic!)
19     gender_weights = {
20         'Male': 10,
21         'Female': 10,
22         'Other': 10
23     }
24     score += gender_weights.get(gender, 0)
25
26     # Age weighting (potentially biased)
27     if 25 <= age <= 35:
28         score += 10 # Favoring mid-career age
29     elif 18 <= age < 25:
30         score += 5 # Entry-level age
31     elif age > 35:
32         score += 5 # Senior age
```

VS Code editor showing a Python script for calculating applicant scores. The script includes a function `score_applicant` and a list of applicants.

```
C:\B.TECH > 4.2 > TASK 4.py > ...
32     score += 5 # Senior age
33     else:
34         score += 0 # Unusual age range
35
36     return round(score, 2)
37
38 def score_applicant(education, experience, gender, age):
39     education_weights = {
40         'HighSchool': 10,
41         'Diploma': 20,
42         'BTech': 30,
43         'MTech': 40,
44         'PhD': 50
45     }
46     score = education_weights.get(education, 0)
47     score += min(experience * 2, 30)
48
49     gender_weights = {
50         'Male': 10,
51         'Female': 10,
52         'Other': 10
53     }
54     score += gender_weights.get(gender, 0)
55
56     if 25 <= age <= 35:
57         score += 10
58     elif 18 <= age < 25 or age > 35:
59         score += 5
60
61     return round(score, 2)
62
63 # List of applicants
```

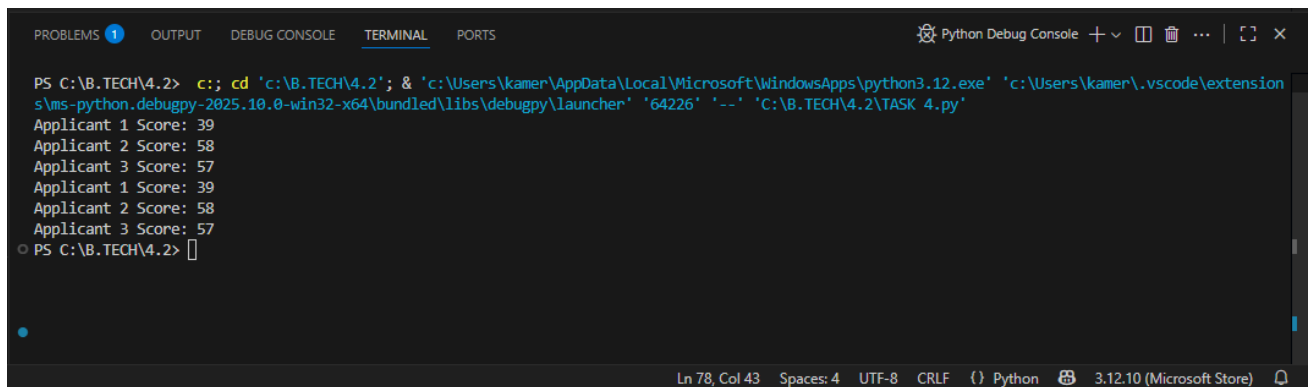
VS Code interface details: RUN AND DEBUG sidebar, BREAKPOINTS section, status bar (Ln 78, Col 43, Spaces: 4, UTF-8, CRLF, Python, 3.12.10 (Microsoft Store)).

VS Code editor showing the continuation of the Python script, including a loop to calculate scores for each applicant.

```
38 def score_applicant(education, experience, gender, age):
61     return round(score, 2)
62
63 # List of applicants
64 applicants = [
65     {'education': 'Diploma', 'experience': 2, 'gender': 'Female', 'age': 22},
66     {'education': 'BTech', 'experience': 4, 'gender': 'Male', 'age': 28},
67     {'education': 'MTech', 'experience': 1, 'gender': 'Other', 'age': 36}
68 ]
69
70 # Loop through applicants and score each one
71 for i, applicant in enumerate(applicants, start=1):
72     score = score_applicant(
73         education=applicant['education'],
74         experience=applicant['experience'],
75         gender=applicant['gender'],
76         age=applicant['age']
77     )
78     print(f"Applicant {i} Score: {score}")
79
80 for i, applicant in enumerate(applicants, start=1):
81     score = score_applicant(**applicant)
82     print(f"Applicant {i} Score: {score}")
```

VS Code interface details: RUN AND DEBUG sidebar, BREAKPOINTS section, status bar (Ln 78, Col 43, Spaces: 4, UTF-8, CRLF, Python, 3.12.10 (Microsoft Store)).

OUTPUT:



```
PS C:\B.TECH\4.2> c:: cd 'c:\B.TECH\4.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundle\libs\debugpy\launcher' '64226' '--' 'C:\B.TECH\4.2\TASK 4.py'
Applicant 1 Score: 39
Applicant 2 Score: 58
Applicant 3 Score: 57
Applicant 1 Score: 39
Applicant 2 Score: 58
Applicant 3 Score: 57
PS C:\B.TECH\4.2> 
```

TASK 5

Code Snippet:

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

CODE:

The screenshot shows the Visual Studio Code (VS Code) interface. The main editor window displays a Python file named `TASK 5.py`. The code defines a function `greet_user(name, gender)` that normalizes the gender input to lowercase and assigns a title based on the gender: "Mr." for male, "Mrs." for female, and "Mx." for gender-neutral. The function returns a formatted string: `f"Hello, {title} {name}! Welcome."`. Below the function definition, there is a call to the function with sample input: `output = greet_user("Jashuva", "Other")`, followed by `print(output)`. The left sidebar shows the "RUN AND DEBUG" panel with options to "Run and Debug" or "Show automatic Python configurations". The bottom status bar indicates the file is at line 2, column 5, using UTF-8 encoding and CRLF line endings, with the Python 3.12.10 interpreter selected.

OUTPUT:

The screenshot shows the VS Code terminal window. The terminal displays the command prompt `PS C:\B.TECH\4.2>` followed by the command `c:: cd 'c:\B.TECH\4.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\.vscode\extension\s\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '64631' '--' 'C:\B.TECH\4.2\TASK 5.py'`. The output of the script is `Hello, Mx. Jashuva! Welcome.`. The terminal also shows the prompt `PS C:\B.TECH\4.2>` followed by a cursor. The bottom status bar indicates the file is at line 2, column 5, using UTF-8 encoding and CRLF line endings, with the Python 3.12.10 interpreter selected.

OBSERVATION : I observed that GitHub copilot can quickly generate working code for tasks such as login systems, loan approvals, Fibonacci functions, and job applicant scoring. However, the generated code sometimes contains issues like hardcoded values, lack of encryption, or biased decision logic. This shows that AI tools are helpful for faster coding but require human review for security, fairness, and correctness. GitHub Copilot is a fascinating tool to observe—especially in how it transforms the developer experience. Here's a breakdown of key observations across its functionality, impact, and adoption