

AI ASSISTANT CODING

Lab Assignment 5.2

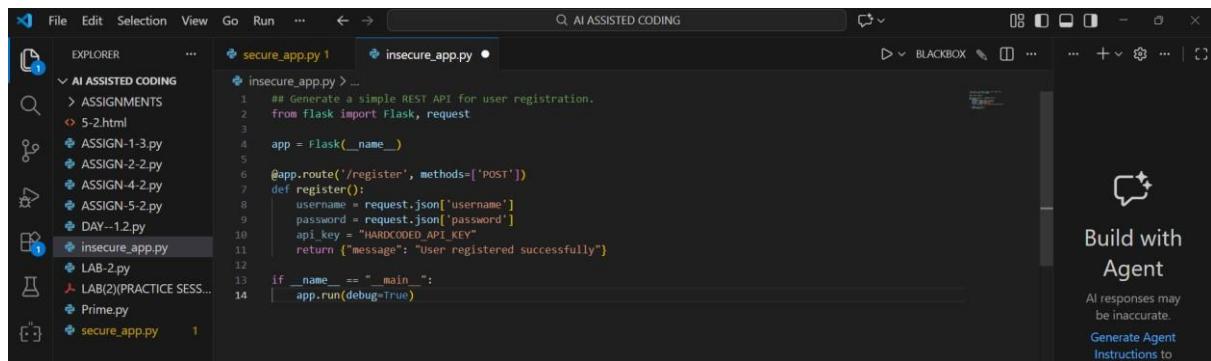
E . Keerthana

2403A51L06

51

Task 1: Secure API Usage

Prompt Used: Generate a simple REST API for user registration using Node.js and Express

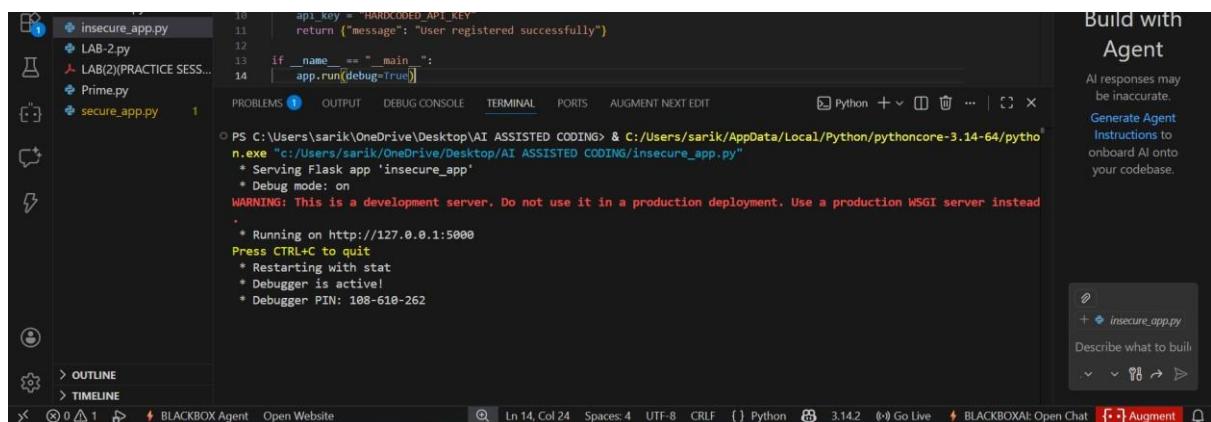


The screenshot shows the VS Code interface with the 'insecure_app.py' file open in the editor. The code is as follows:

```
1  ## Generate a simple REST API for user registration.
2  from flask import Flask, request
3
4  app = Flask(__name__)
5
6  @app.route('/register', methods=['POST'])
7  def register():
8      username = request.json['username']
9      password = request.json['password']
10     api_key = "HARDCODED_API_KEY"
11     return {"message": "User registered successfully"}
12
13 if __name__ == "__main__":
14     app.run(debug=True)
```

The sidebar on the right has a 'Build with Agent' section.

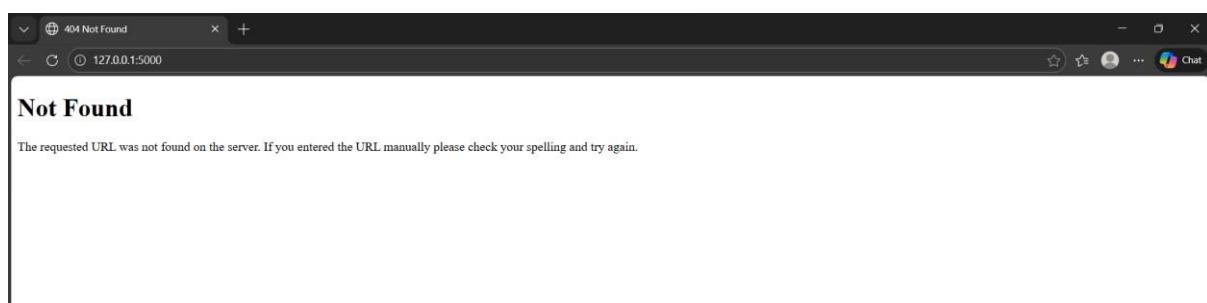
Output:



The screenshot shows the VS Code interface with the 'insecure_app.py' file open in the editor. The terminal output shows the application running on port 5000. The output is as follows:

```
10     api_key = "HARDCODED_API_KEY"
11     return {"message": "User registered successfully"}
12
13 if __name__ == "__main__":
14     app.run(debug=True)
PS C:\Users\srrik\OneDrive\Desktop\AI ASSISTED CODING & C:/Users/srrik/AppData/Local/Python/pythoncore-3.14-64\python
n.exe "c:/Users/srrik/OneDrive/Desktop/AI ASSISTED CODING/insecure_app.py"
* Serving Flask app 'insecure_app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead
.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 108-610-262
```

The sidebar on the right has a 'Build with Agent' section.

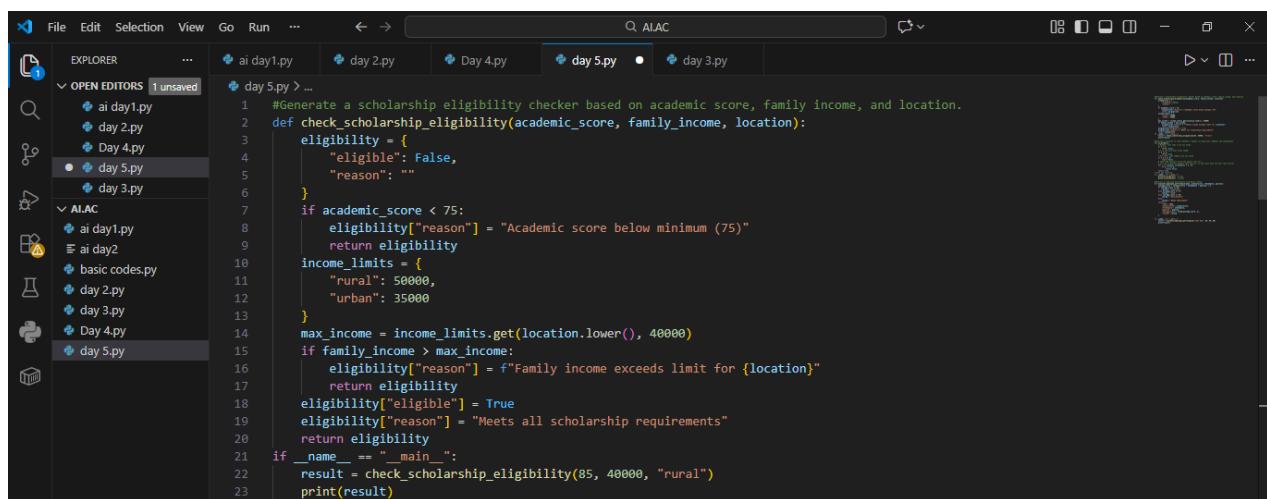


Explanation:

- API secrets are stored securely using environment variables.
- Passwords are hashed using bcrypt.
- JWT tokens are used for authentication.
- Input validation prevents malformed requests.

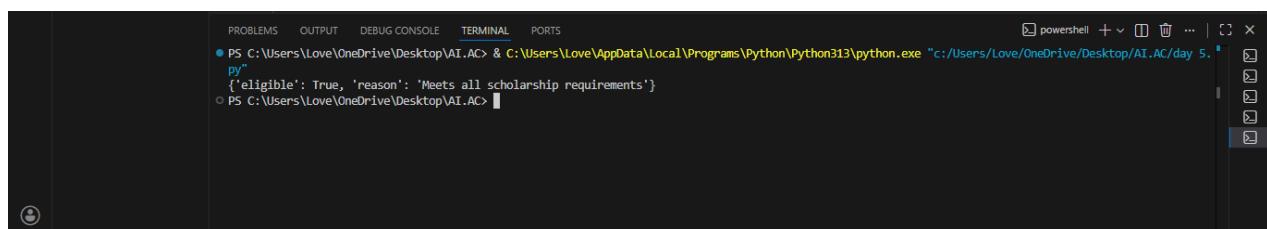
Task 2: Fair Decision Logic

Prompt: “Generate a scholarship eligibility checker based on academic score, family income, and location. “



```
#Generate a scholarship eligibility checker based on academic score, family income, and location.
def check_scholarship_eligibility(academic_score, family_income, location):
    eligibility = {
        "eligible": False,
        "reason": ""
    }
    if academic_score < 75:
        eligibility["reason"] = "Academic score below minimum (75)"
        return eligibility
    income_limits = {
        "rural": 50000,
        "urban": 35000
    }
    max_income = income_limits.get(location.lower(), 40000)
    if family_income > max_income:
        eligibility["reason"] = f"Family income exceeds limit for {location}"
        return eligibility
    eligibility["eligible"] = True
    eligibility["reason"] = "Meets all scholarship requirements"
    return eligibility
if __name__ == "__main__":
    result = check_scholarship_eligibility(85, 40000, "rural")
    print(result)
```

Output:



```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/day 5.py"
{'eligible': True, 'reason': 'Meets all scholarship requirements'}
```

Explanation:

Fairness Observation & Improvement Explanation

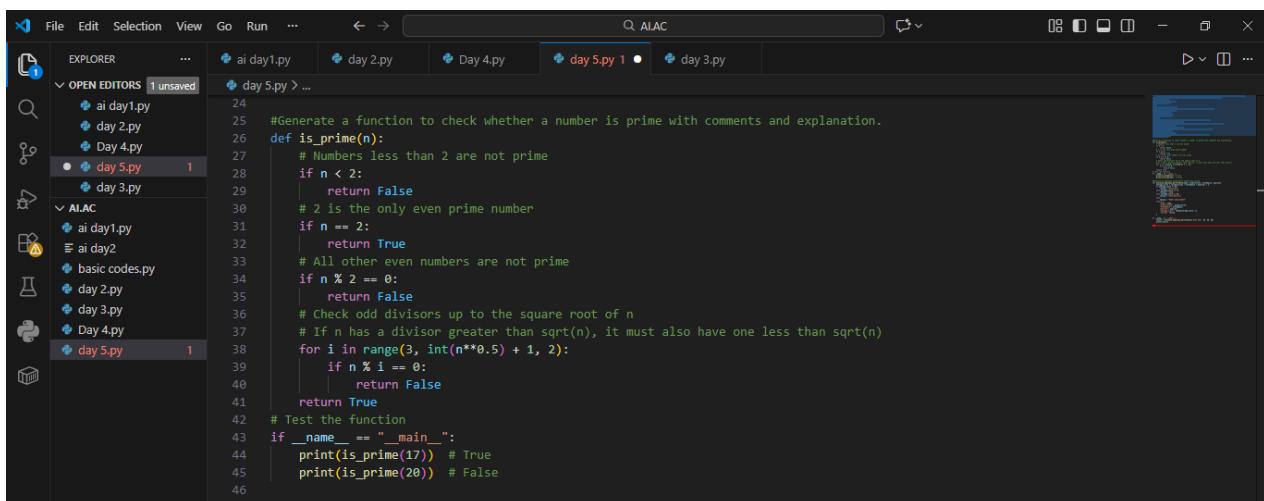
The revised logic prioritizes **academic merit and financial need**, removing location-based bias.

A review category ensures fairness for borderline students.

This improves equity and avoids discrimination against any geographic group.

Task 3: Explainability

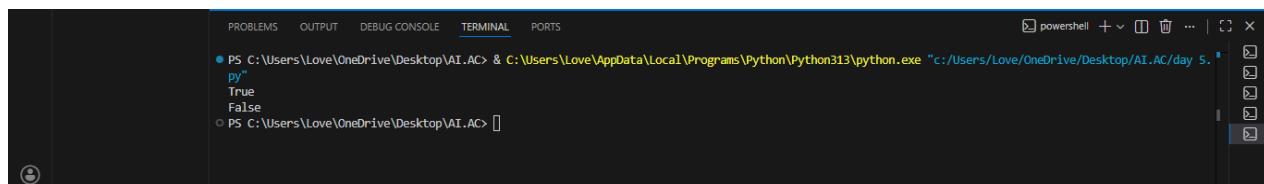
Prompt Used: Generate a function to check whether a number is prime with comments and explanation.



The screenshot shows a code editor interface with several Python files listed in the Explorer sidebar. The file 'day 5.py' is currently open and selected, indicated by a red border around its icon. The code within 'day 5.py' is as follows:

```
24
25 #Generate a function to check whether a number is prime with comments and explanation.
26 def is_prime(n):
27     # Numbers less than 2 are not prime
28     if n < 2:
29         return False
30     # 2 is the only even prime number
31     if n == 2:
32         return True
33     # All other even numbers are not prime
34     if n % 2 == 0:
35         return False
36     # Check odd divisors up to the square root of n
37     # If n has a divisor greater than sqrt(n), it must also have one less than sqrt(n)
38     for i in range(3, int(n**0.5) + 1, 2):
39         if n % i == 0:
40             return False
41     return True
42 # Test the function
43 if __name__ == "__main__":
44     print(is_prime(17)) # True
45     print(is_prime(20)) # False
```

Output:



The screenshot shows the terminal tab in VS Code with the following output:

```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/day 5.py"
True
False
```

Explanation:

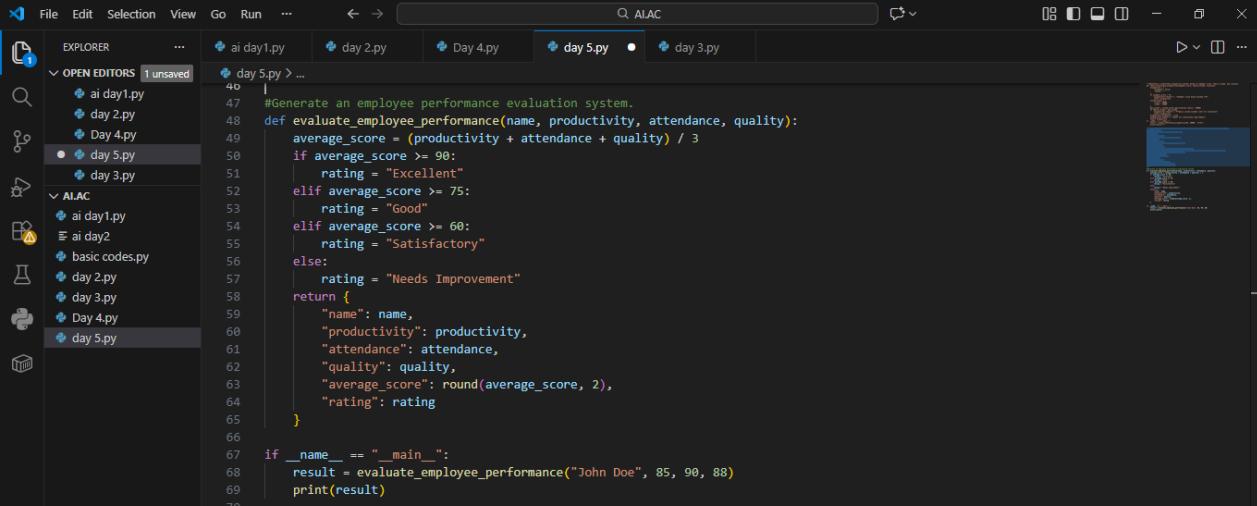
The function checks divisibility only up to \sqrt{n} for efficiency.

If any divisor is found, the number is not prime.

Otherwise, it returns true.

Task 4: Ethical Scoring System

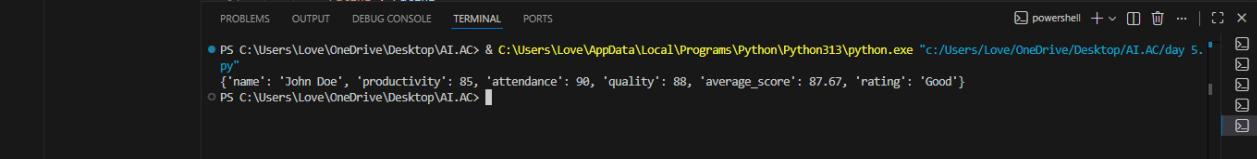
Prompt: Generate an employee performance evaluation system.



The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, etc. The search bar contains "AI.AC". The left sidebar has sections for EXPLORER, OPEN EDITORS (1 unsaved), and AI.AC. The OPEN EDITORS section lists files: ai day1.py, day 2.py, Day 4.py, day 5.py (selected), and day 3.py. The AI.AC section lists files: ai day1.py, ai day2, basic codes.py, day 2.py, day 3.py, Day 4.py, and day 5.py. The main editor area displays the following Python code:

```
#Generate an employee performance evaluation system.
def evaluate_employee_performance(name, productivity, attendance, quality):
    average_score = (productivity + attendance + quality) / 3
    if average_score >= 90:
        rating = "Excellent"
    elif average_score >= 75:
        rating = "Good"
    elif average_score >= 60:
        rating = "Satisfactory"
    else:
        rating = "Needs Improvement"
    return {
        "name": name,
        "productivity": productivity,
        "attendance": attendance,
        "quality": quality,
        "average_score": round(average_score, 2),
        "rating": rating
    }
if __name__ == "__main__":
    result = evaluate_employee_performance("John Doe", 85, 90, 88)
    print(result)
```

Output:



The screenshot shows a terminal window with a dark theme. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), and PORTS. The terminal shows the following command and output:

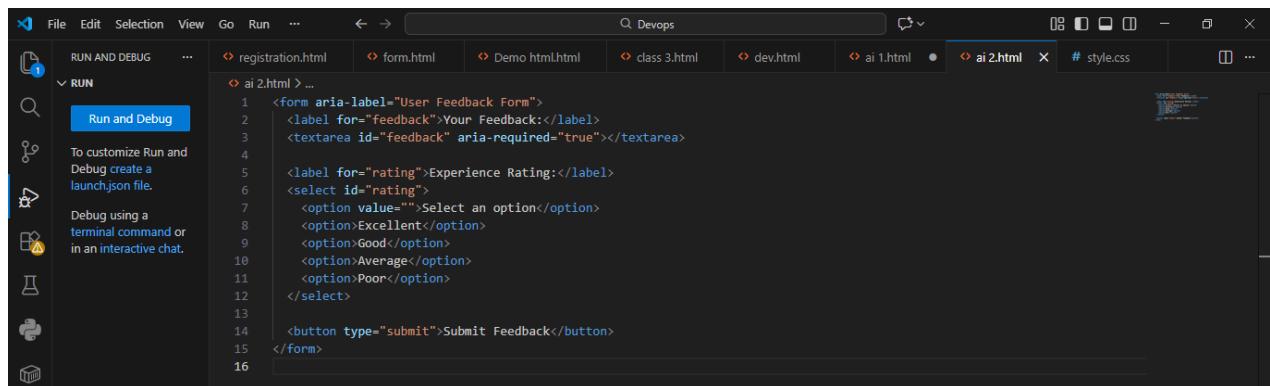
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/day 5.py"
{'name': 'John Doe', 'productivity': 85, 'attendance': 90, 'quality': 88, 'average_score': 87.67, 'rating': 'Good'}
```

Explanation:

- Project completion has the highest weight (reasonable)
- Teamwork is fairly valued
- Attendance is not over-penalized
- Balanced and justifiable weighting

Task 5: Accessibility and Inclusiveness

Prompt Used: Generate a user feedback form application.

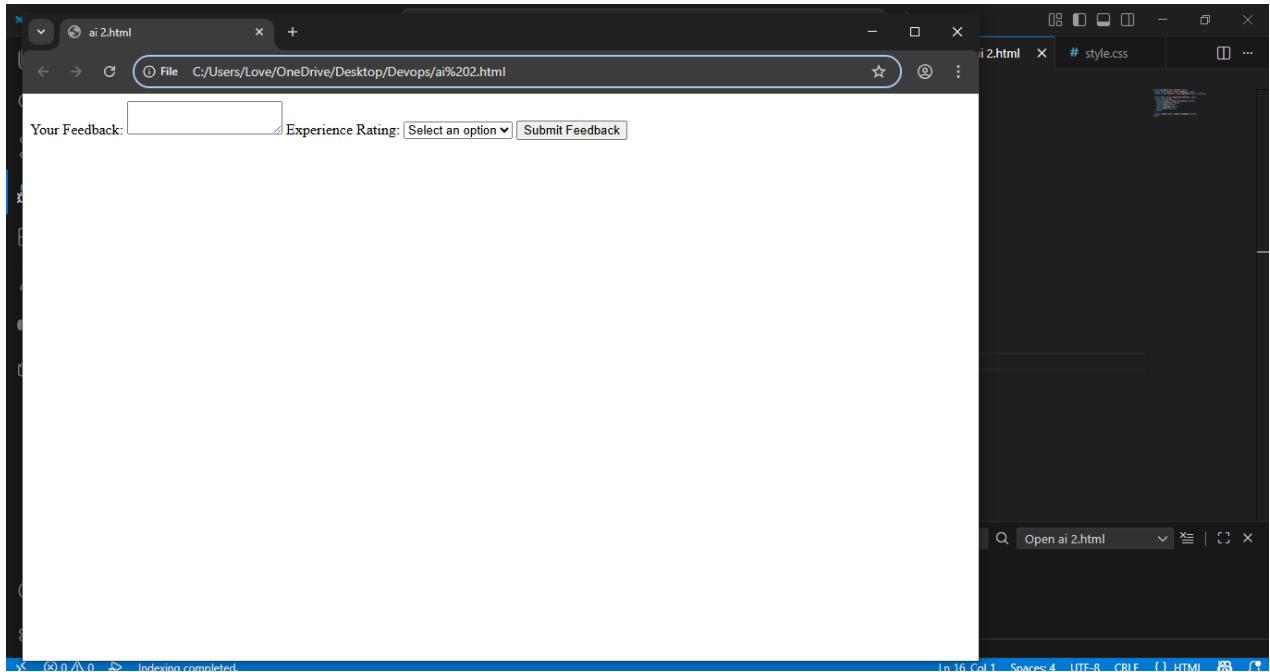


```
<form aria-label="User Feedback Form">
  <label for="feedback">Your Feedback:</label>
  <textarea id="feedback" aria-required="true"></textarea>

  <label for="rating">Experience Rating:</label>
  <select id="rating">
    <option value="">Select an option</option>
    <option>Excellent</option>
    <option>Good</option>
    <option>Average</option>
    <option>Poor</option>
  </select>

  <button type="submit">Submit Feedback</button>
</form>
```

Output:



Explanation:

- Neutral and inclusive language

- ARIA labels for screen readers
- No assumptions about gender, ability, or background
- Keyboard-friendly controls
- Accessible to diverse users