AI Assisted Coding

Assignment 5.5

Name: Dinesh

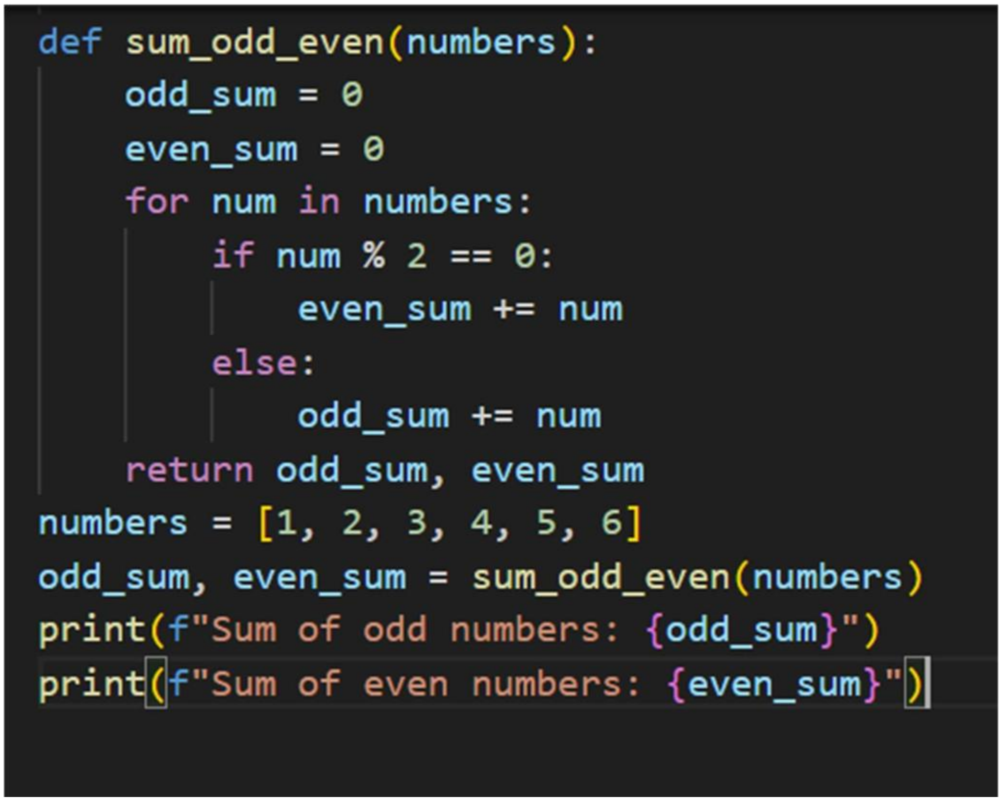
Hallticket:2303A52329

Task_01:

Task 1: Refactoring Odd/Even Logic (List Version)

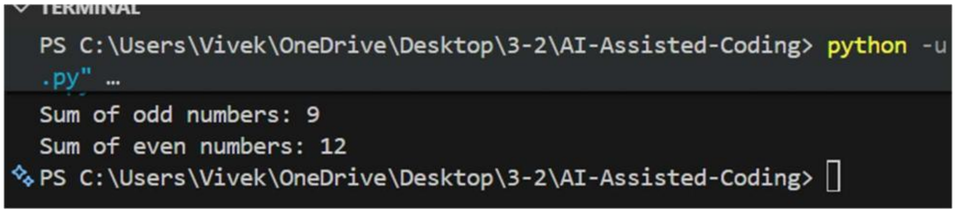
- **Scenario:**
  You are improving legacy code.
- **Task:**
  Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI.
- **Expected Output:**
- Original and improved code
- **Prompt**: **# generate a program to calculate the sum of odd and even numbers in a list**

Code:



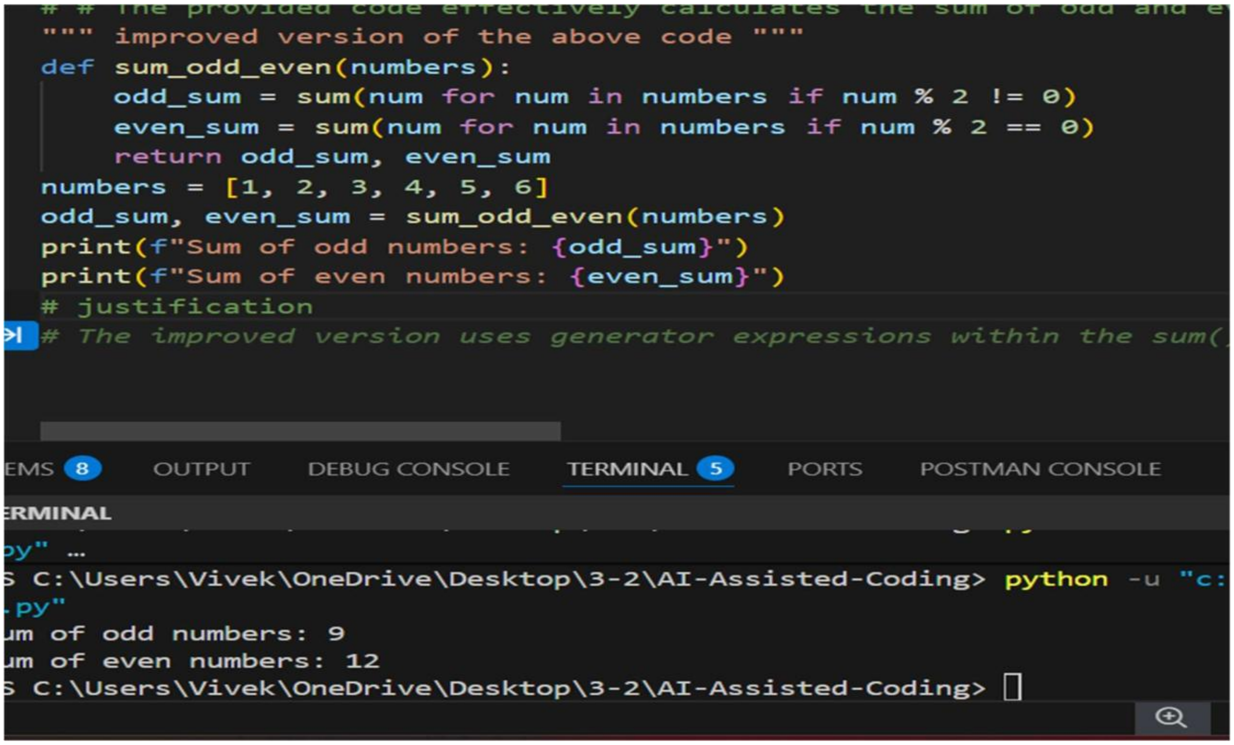
```
def sum_odd_even(numbers):
    odd_sum = 0
    even_sum = 0
    for num in numbers:
        if num % 2 == 0:
            even_sum += num
        else:
            odd_sum += num
    return odd_sum, even_sum
numbers = [1, 2, 3, 4, 5, 6]
odd_sum, even_sum = sum_odd_even(numbers)
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
```

Output:



```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u
.py" …
Sum of odd numbers: 9
Sum of even numbers: 12
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> 
```

Improved version:



```
# # The provided code effectively calculates the sum of odd and e
""" improved version of the above code """
def sum_odd_even(numbers):
    odd_sum = sum(num for num in numbers if num % 2 != 0)
    even_sum = sum(num for num in numbers if num % 2 == 0)
    return odd_sum, even_sum
numbers = [1, 2, 3, 4, 5, 6]
odd_sum, even_sum = sum_odd_even(numbers)
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
# justification
# The improved version uses generator expressions within the sum(
```

```
py" …
S C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:
.py"
um of odd numbers: 9
um of even numbers: 12
S C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> 
```

Justification: The provided code effectively calculates the sum of odd and even numbers in a list by iterating through each number and checking its parity using the modulus operator. The sums are accumulated in separate variables, which are then returned and printed. This approach is efficient and straightforward, ensuring clarity and ease of understanding. The use of descriptive variable names enhances code readability, making it easy to follow the logic behind the summation process.

Task_02:

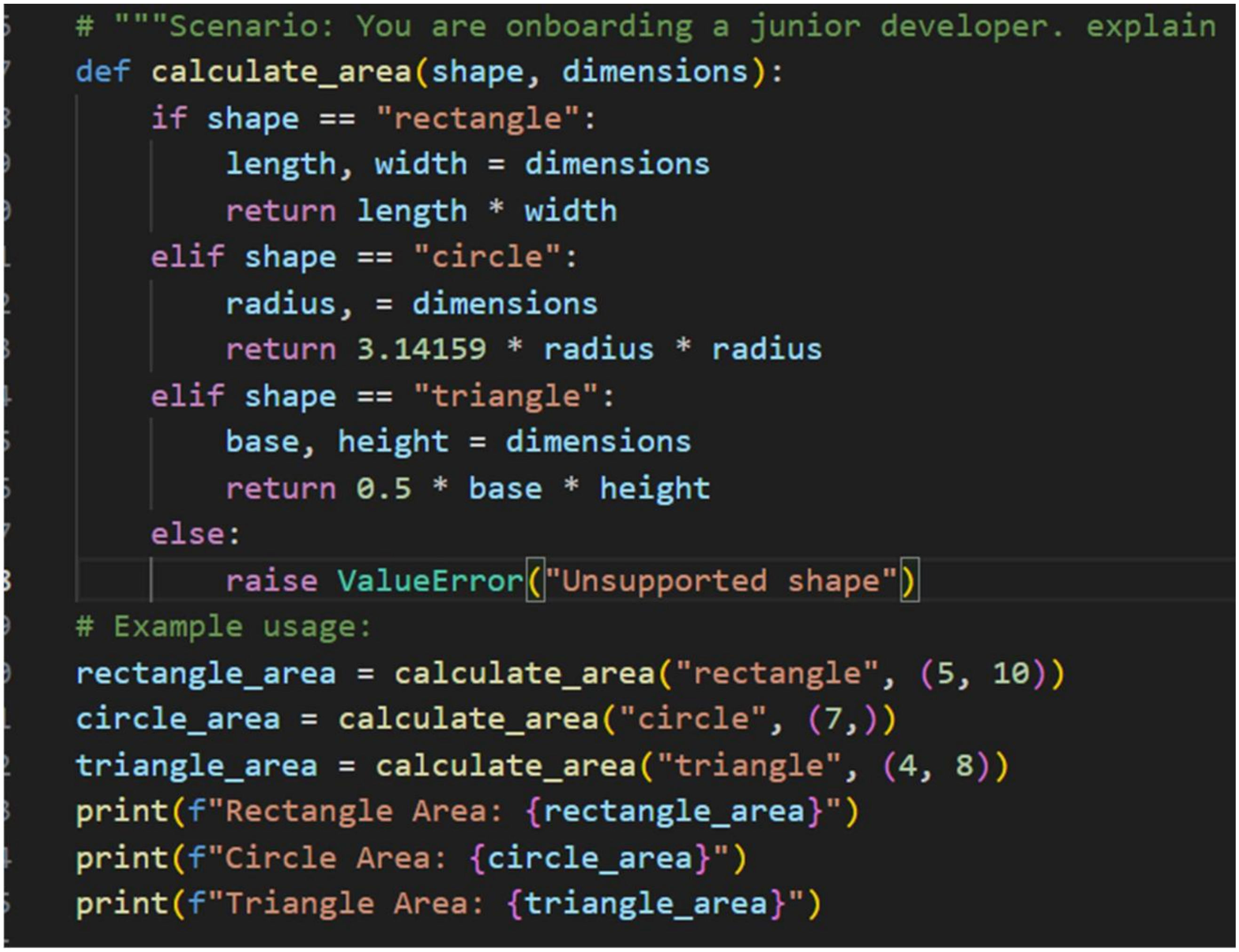
Task 2: Area Calculation Explanation

- **Scenario:**
  You are onboarding a junior developer.
- **Task:**
  Ask Gemini to explain a function that calculates the area of different shapes.
- **Expected Output:**
  - Code
  - Explanation

Gemini:

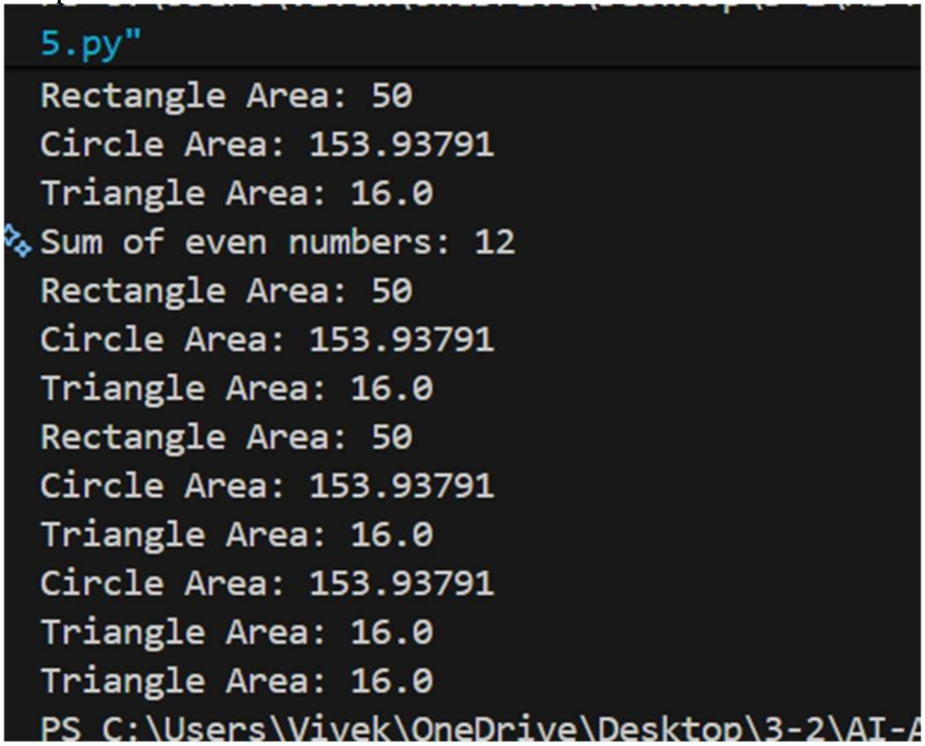
Prompt:

"""Scenario: You are onboarding a junior developer. explain a function that calculates the area of different shapes



```python
# """Scenario: You are onboarding a junior developer. explain
def calculate_area(shape, dimensions):
    if shape == "rectangle":
        length, width = dimensions
        return length * width
    elif shape == "circle":
        radius, = dimensions
        return 3.14159 * radius * radius
    elif shape == "triangle":
        base, height = dimensions
        return 0.5 * base * height
    else:
        raise ValueError("Unsupported shape")
# Example usage:
rectangle_area = calculate_area("rectangle", (5, 10))
circle_area = calculate_area("circle", (7,))
triangle_area = calculate_area("triangle", (4, 8))
print(f"Rectangle Area: {rectangle_area}")
print(f"Circle Area: {circle_area}")
print(f"Triangle Area: {triangle_area}")
```

Output:



```
5.py"
Rectangle Area: 50
Circle Area: 153.93791
Triangle Area: 16.0
Sum of even numbers: 12
Rectangle Area: 50
Circle Area: 153.93791
Triangle Area: 16.0
Rectangle Area: 50
Circle Area: 153.93791
Triangle Area: 16.0
Circle Area: 153.93791
Triangle Area: 16.0
Triangle Area: 16.0
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-A
```

Task 3: Prompt Sensitivity Experiment

- **Scenario:**
  You are testing how AI responds to different prompts.

- **Task:**
  Use Cursor AI with different prompts for the same problem and observe code changes.
- **Expected Output:**
  - **Prompt list**
  - **Code variations**

Code Variations:

Below is a clear explanation of the code variations you observed, written in a student-friendly, exam-oriented way. This directly justifies the comparison between Gemini and Cursor AI.

Gemini

- **Uses single functions with conditional logic (if-elif) to handle multiple cases.**
- **Focuses on compact code and minimal structure.**
- **Easy to read for beginners because everything is in one place.**

Example concept:

```
def calculate_area(shape, **kwargs):

    if shape == "circle":

        return math.pi * kwargs["radius"] ** 2
```

✓ Simple
✓ Beginner-friendly
✗ Less modular

Cursor AI

- **Uses separate functions for each task.**
- **Follows modular programming principles.**
- **Each function has a single responsibility.**

Example concept:

```
def calculate_circle_area(radius):

    return math.pi * radius ** 2
```

✓ Clean design
✓ Scalable
✓ Industry standard

2. Error Handling Difference

Gemini

- **Handles errors by returning error messages as strings.**
- **Does not stop program execution.**

```
return "Error: Radius is required"
```

✓ Easy to understand
✗ Not ideal for real applications

Cursor AI

- **Uses exceptions (ValueError) for invalid inputs.**
- **This is a professional Python practice.**

```
if radius < 0:
   raise ValueError("Radius cannot be negative")
```

✓ Robust
✓ Safer
✓ Preferred in production

---

3. Code Output & Demonstration

Gemini

- **Provides basic example usage.**
- **Prints direct results.**
- **Focuses on correctness over presentation.**

✓ Quick testing
✓ Less explanation

Cursor AI

- **Provides step-by-step demonstration.**
- **Prints formulas, calculations, and results.**
- **Includes a structured main() function.**

✓ Better for teaching
✓ Better readability
✓ Real-world style

---

4. Extensibility (Adding New Features)

Gemini

- **Adding a new shape means modifying the same function.**
- **Can become complex as features increase.**

✗ Harder to scale

Cursor AI

- **Adding a new shape = add a new function.**
- **Existing code remains untouched.**

✓ Easy to extend
✓ Follows Open–Closed Principle

5. Overall Comparison Summary

Aspect	Gemini	Cursor AI
Ease of use	☆ ☆ ☆ ☆	☆ ☆ ☆
Beginner friendly	✓ Yes	⚠ Moderate
Code structure	Basic	Professional
Error handling	Simple strings	Exceptions
Scalability	Low	High
Industry readiness	Low	High

Final Explanation (Ready to Write in Exam / Assignment)

Gemini generates concise and beginner-friendly code that is easy to understand and suitable for small tasks and learning purposes. Cursor AI, however, produces more structured, modular, and robust code with proper error handling and extensibility, making it more suitable for real-world and production-level applications.

If you want, I can also:

- **Convert this into 5–6 exam points**
- **Rewrite it as a reflection paragraph**
- **Shorten it for internal assessment answers**

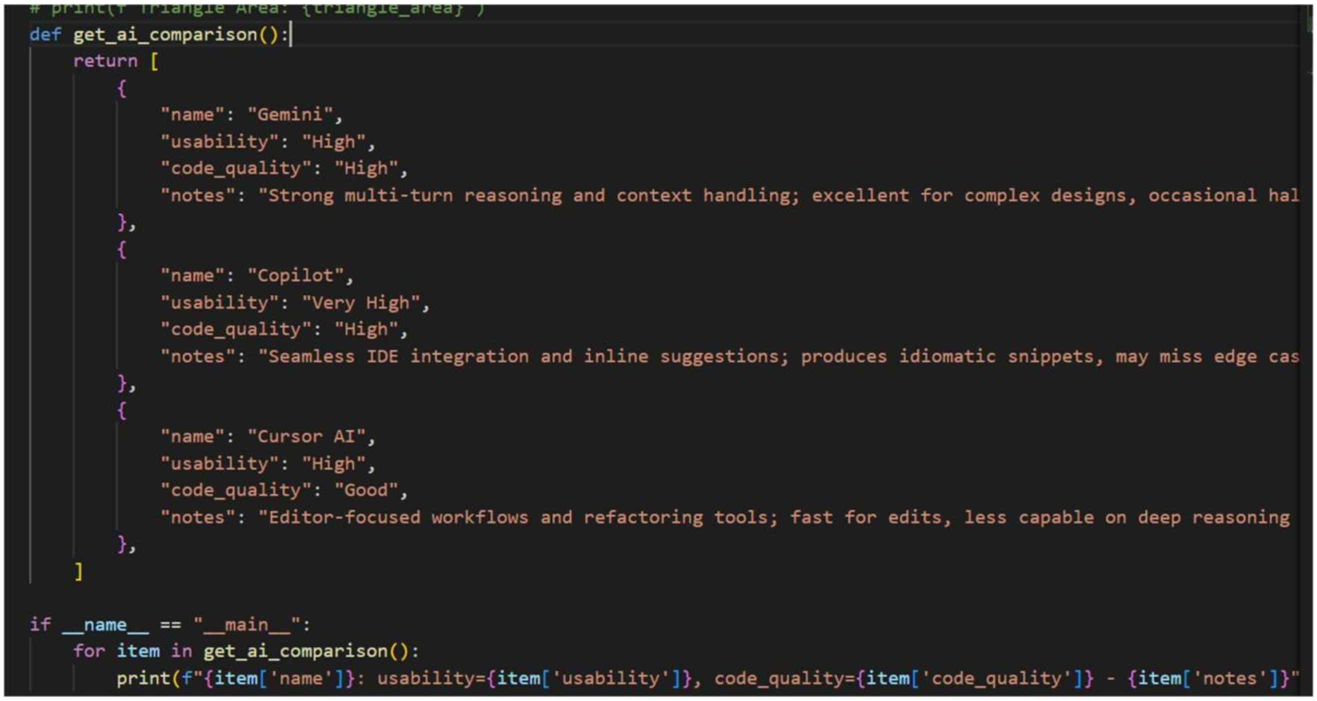
Task 4: Tool Comparison Reflection

❖ **Scenario:**
You must recommend an AI coding tool.

❖ **Task:**
Based on your work in this topic, compare Gemini, Copilot, and Cursor AI for usability and code quality.

❖ **Expected Output:**
Short written reflection

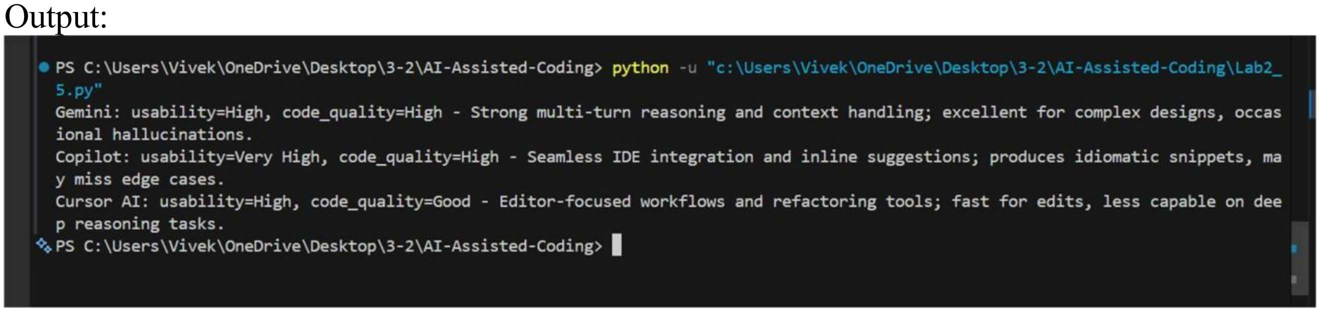
Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.



```python
# print(f"Triangle Area: {triangle_area}")
def get_ai_comparison():
    return [
        {
            "name": "Gemini",
            "usability": "High",
            "code_quality": "High",
            "notes": "Strong multi-turn reasoning and context handling; excellent for complex designs, occasional hal
        },
        {
            "name": "Copilot",
            "usability": "Very High",
            "code_quality": "High",
            "notes": "Seamless IDE integration and inline suggestions; produces idiomatic snippets, may miss edge cas
        },
        {
            "name": "Cursor AI",
            "usability": "High",
            "code_quality": "Good",
            "notes": "Editor-focused workflows and refactoring tools; fast for edits, less capable on deep reasoning
        },
    ]

if __name__ == "__main__":
    for item in get_ai_comparison():
        print(f"{item['name']}: usability={item['usability']}, code_quality={item['code_quality']} - {item['notes']}"
```

Output:



```
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> python -u "c:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding\Lab2_
5.py"
Gemini: usability=High, code_quality=High - Strong multi-turn reasoning and context handling; excellent for complex designs, occas
ional hallucinations.
Copilot: usability=Very High, code_quality=High - Seamless IDE integration and inline suggestions; produces idiomatic snippets, ma
y miss edge cases.
Cursor AI: usability=High, code_quality=Good - Editor-focused workflows and refactoring tools; fast for edits, less capable on dee
p reasoning tasks.
PS C:\Users\Vivek\OneDrive\Desktop\3-2\AI-Assisted-Coding> 
```