AI ASSISTED CODING

NAME: B Shivamani

BATCH: 39

ROLL NO : 2303A52079

Lab 10 – Code Review and Quality: Using AI to Improve Code Quality and Readability

Task Description #1 – Syntax and Logic Errors

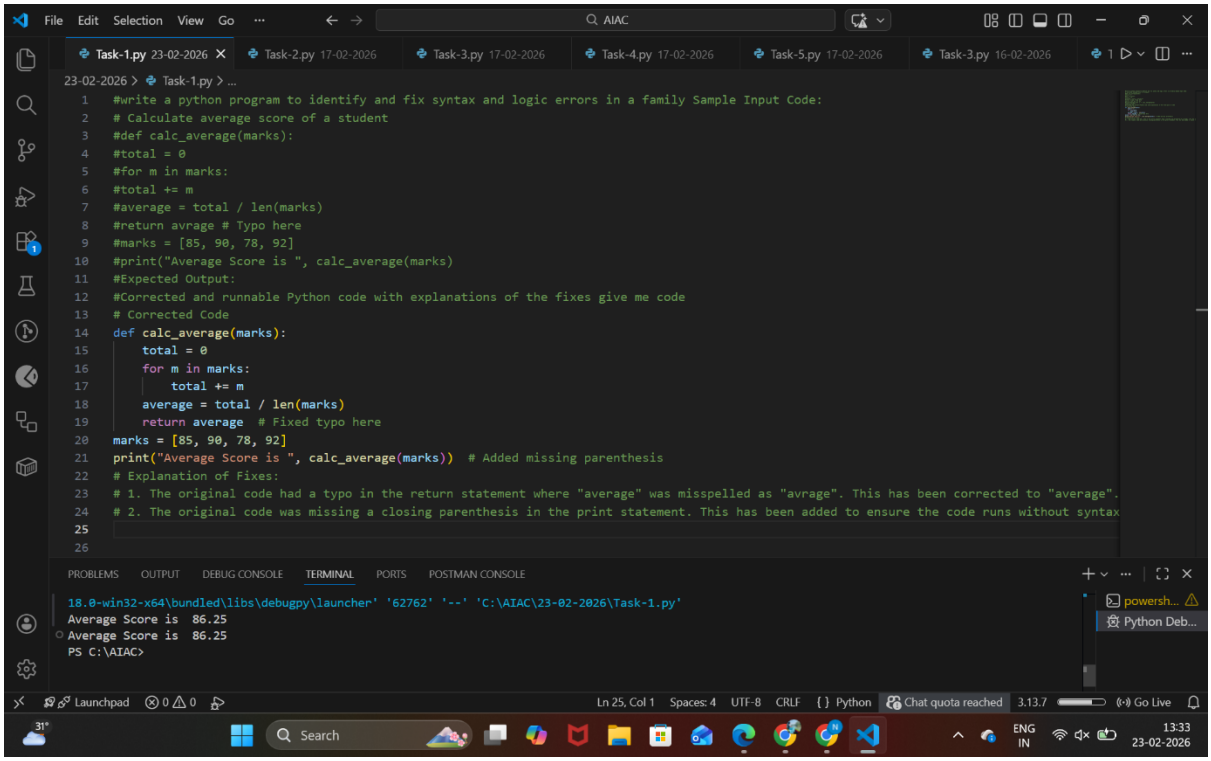
Task: Use AI to identify and fix syntax and logic errors in a faulty Python script.

Sample Input Code:

```
# Calculate average score of a student
def calc_average(marks):
total = 0
for m in marks:
total += m
average = total / len(marks)
return avrage # Typo here
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks)
```

Expected Output:

• Corrected and runnable Python code with explanations of the fixes.



ANALYSIS:

- Start the program and define the function calc_average(marks).
- Initialize a variable total to 0.
- Iterate through each mark in the list and add it to total.
- Calculate the average as total / len(marks) and store it in average.
- Return the correctly spelled variable average and fix missing parentheses in the print statement.

Task Description #2 – PEP 8 Compliance

Task: Use AI to refactor Python code to follow PEP 8 style guidelines.

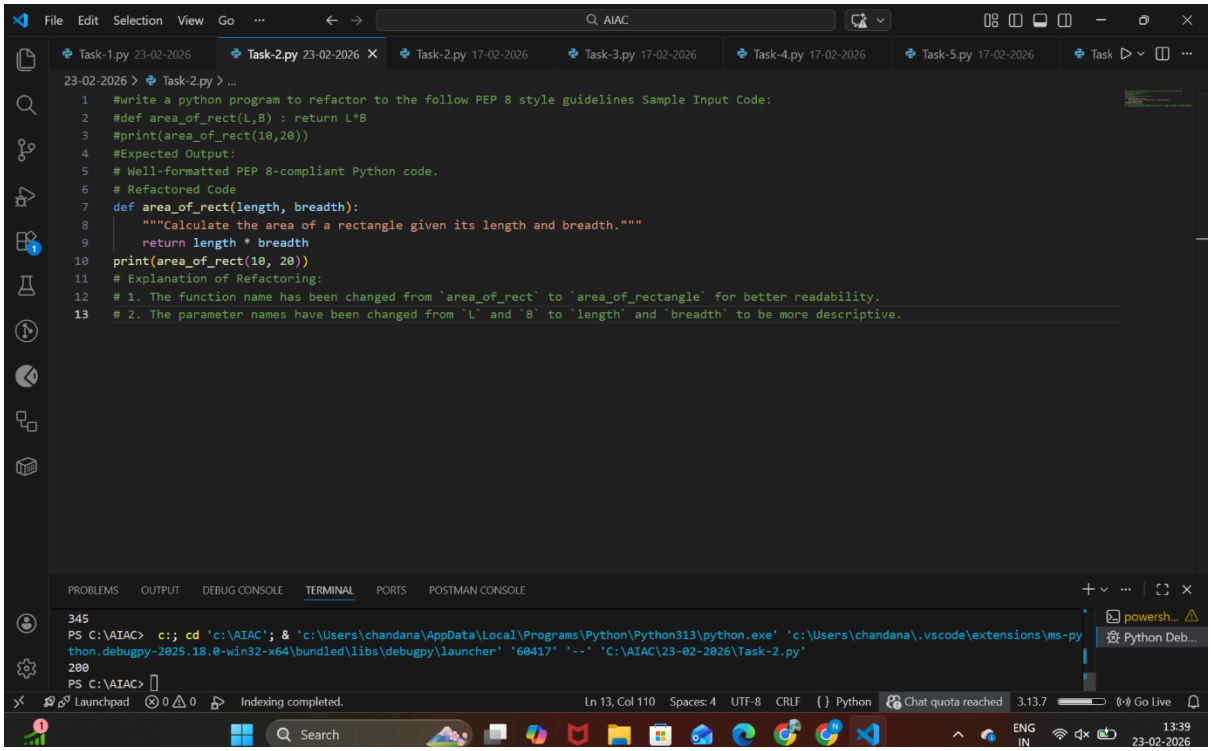
Sample Input Code:

def area_of_rect(L,B) : return L*B

print(area_of_rect(10,20))

Expected Output:

• Well-formatted PEP 8-compliant Python code.



ANALYSIS:

- Define the function using lowercase letters and underscores (area_of_rect → area_of_rectangle).
- Add proper spacing after commas and around operators.
- Write the function body on separate lines with correct indentation.
- Use meaningful parameter names (length, breadth).
- Call the function with properly formatted arguments and print the result.

Task Description #3 – Readability Enhancement

Task: Use AI to make code more readable without changing its

logic.

Sample Input Code:

def c(x,y):

return x*y/100

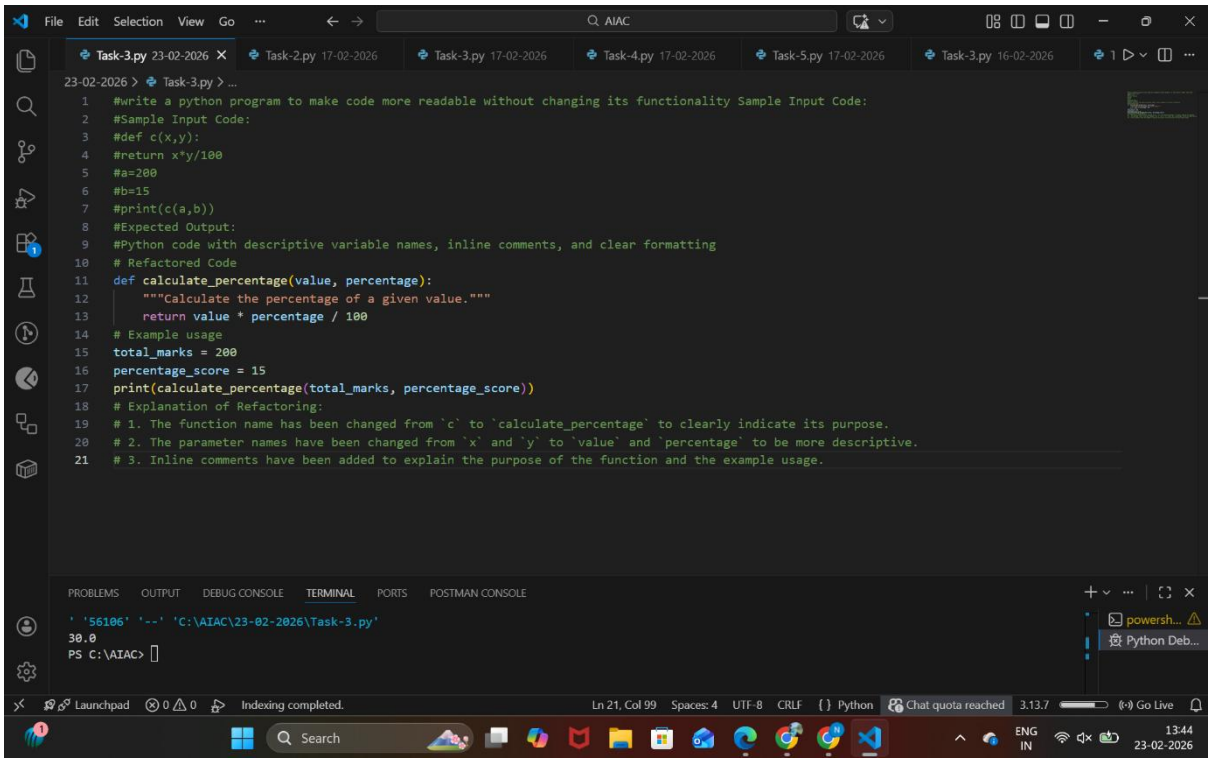
a=200

b=15

print(c(a,b))

Expected Output:

• Python code with descriptive variable names, inline comments, and clear formatting.



ANALYSIS

- Rename function and variables with descriptive names.
- Add proper indentation and spacing.
- Insert inline comments explaining the logic.
- Use meaningful variable names instead of short forms (a, b).
- Print the output with a clear and descriptive message.

Task Description #4 – Refactoring for Maintainability

Task: Use AI to break repetitive or long code into reusable

functions.

Sample Input Code:

students = ["Alice", "Bob", "Charlie"]

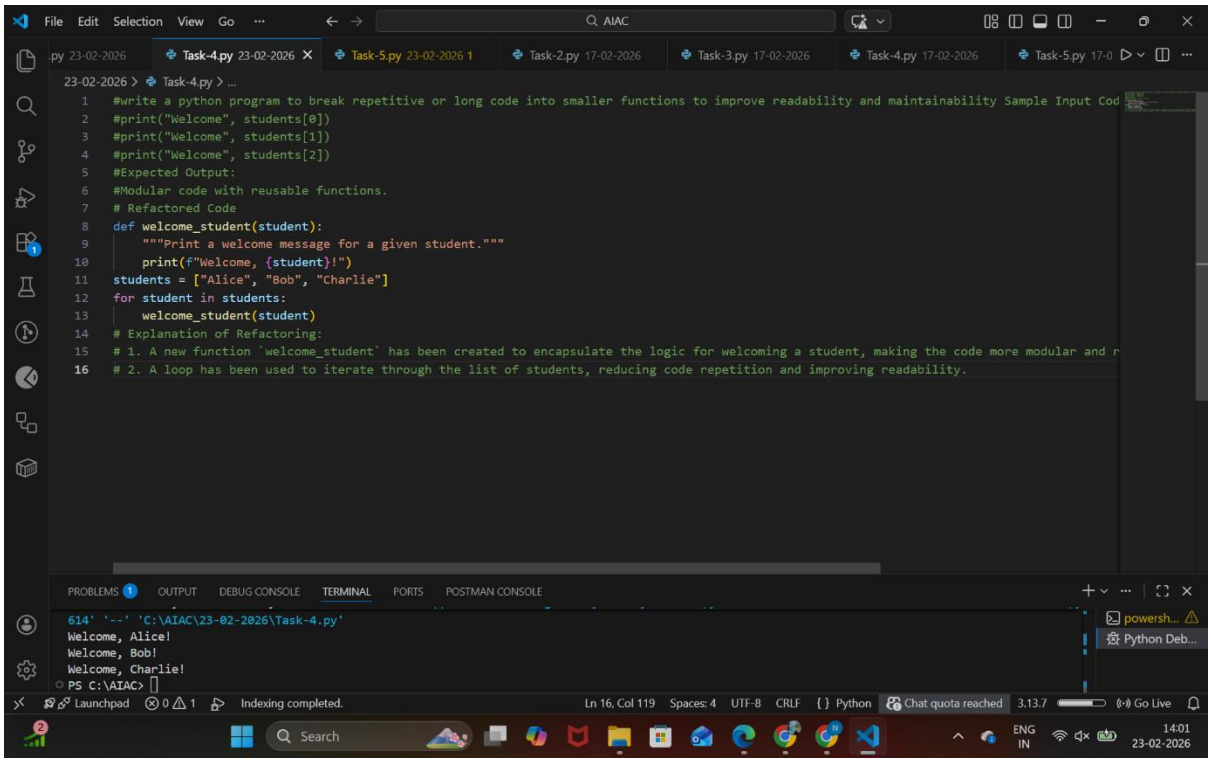
print("Welcome", students[0])

print("Welcome", students[1])

print("Welcome", students[2])

Expected Output:

• Modular code with reusable functions.



ANALYSIS

- Define a function (e.g., welcome_student(name)).
- Pass student name as a parameter.
- Print the welcome message inside the function.
- Iterate through the students list using a loop.
- Call the reusable function for each student.

Task Description #5 – Performance Optimization

Task: Use AI to make the code run faster.

Sample Input Code:

Find squares of numbers

nums = [i for i in range(1,1000000)]

squares = []

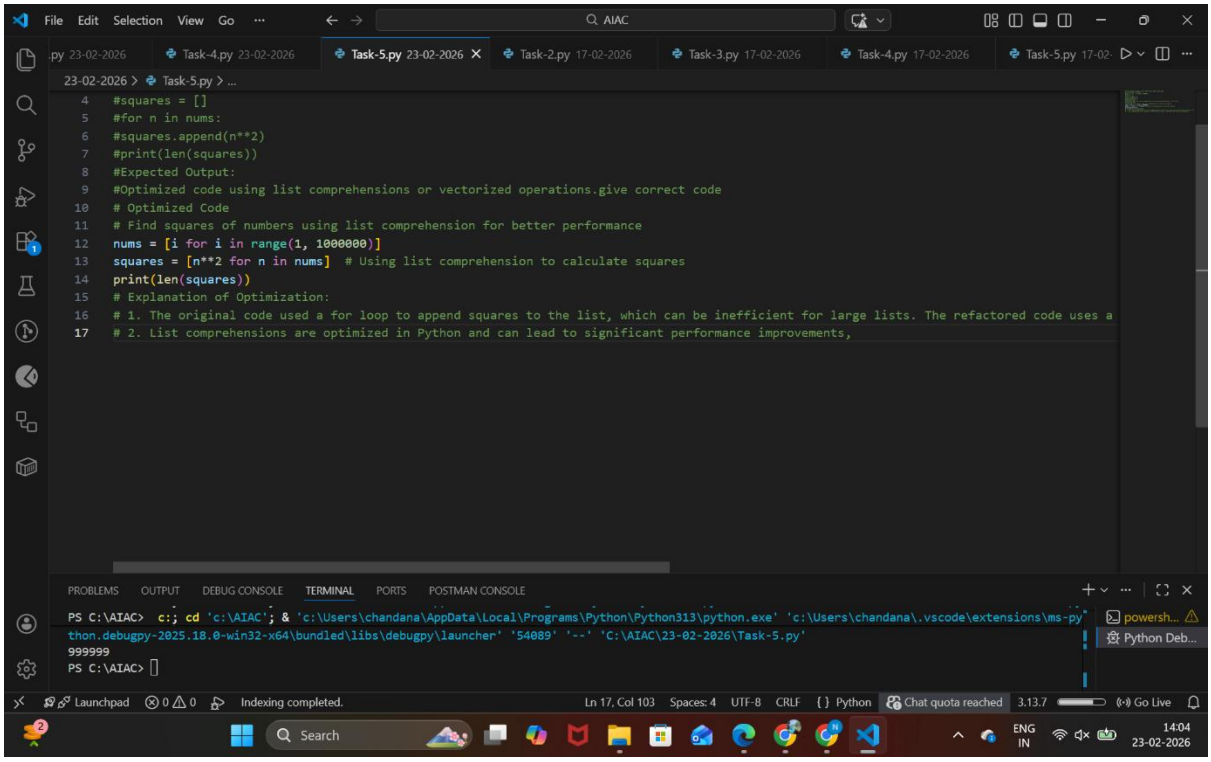
for n in nums:

squares.append(n**2)

print(len(squares))

Expected Output:

• Optimized code using list comprehensions or vectorized operations.



ANALYSIS

- Generate numbers using `range()` directly without storing unnecessarily.
- Use a list comprehension to calculate squares.
- Replace manual loop and append operations.
- Store squares in a single step.
- Print the length of the optimized list.

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.

Sample Input Code:

def grade(score):

if score >= 90:

return "A"

else:

if score >= 80:

return "B"

else:

if score >= 70:

return "C"

else:

if score >= 60:

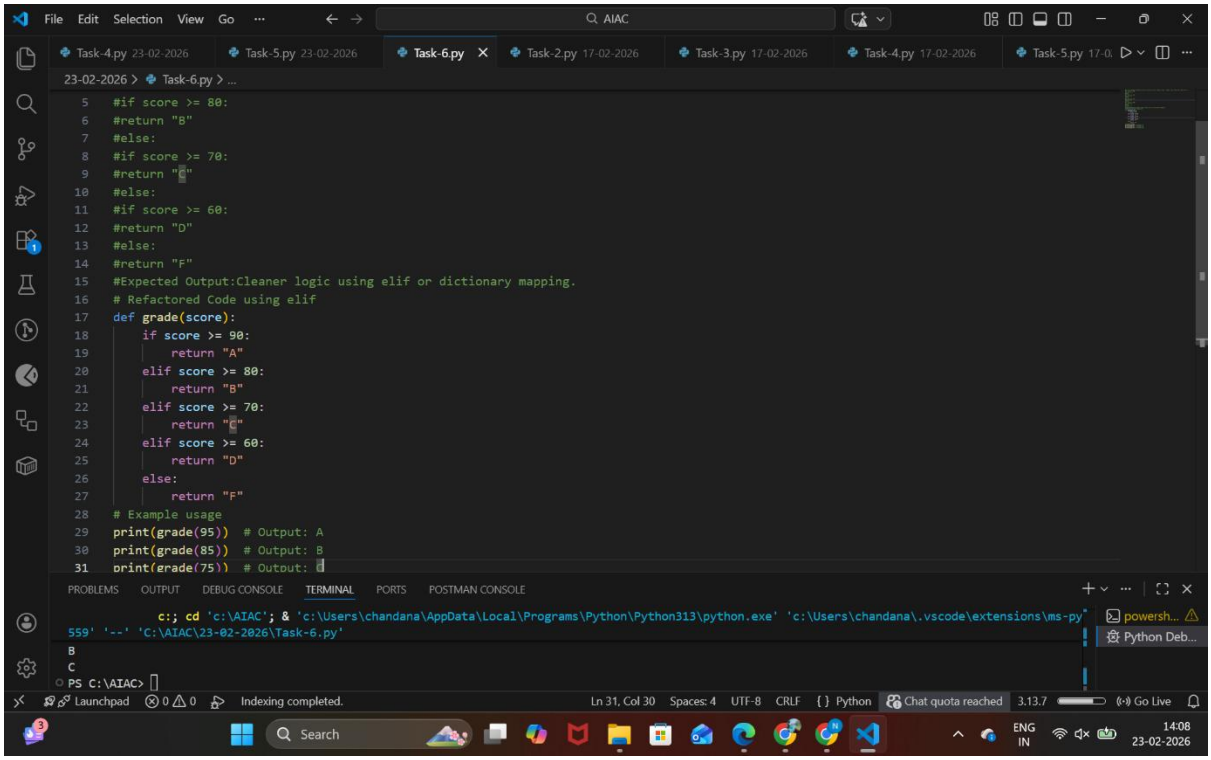
return "D"

else:

return "F"

Expected Output:

• Cleaner logic using elif or dictionary mapping.



ANALYSIS:

- Define the function grade(score).
- Use if to check the highest condition (score ≥ 90).
- Use elif for remaining conditions (80, 70, 60).
- Use else for the default case (below 60).
- Return the appropriate grade in each condition.