AI ASSISTED CODING

ASSIGNMENT-10.1

N. Madhuvani

2303A52117

Batch-40

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:

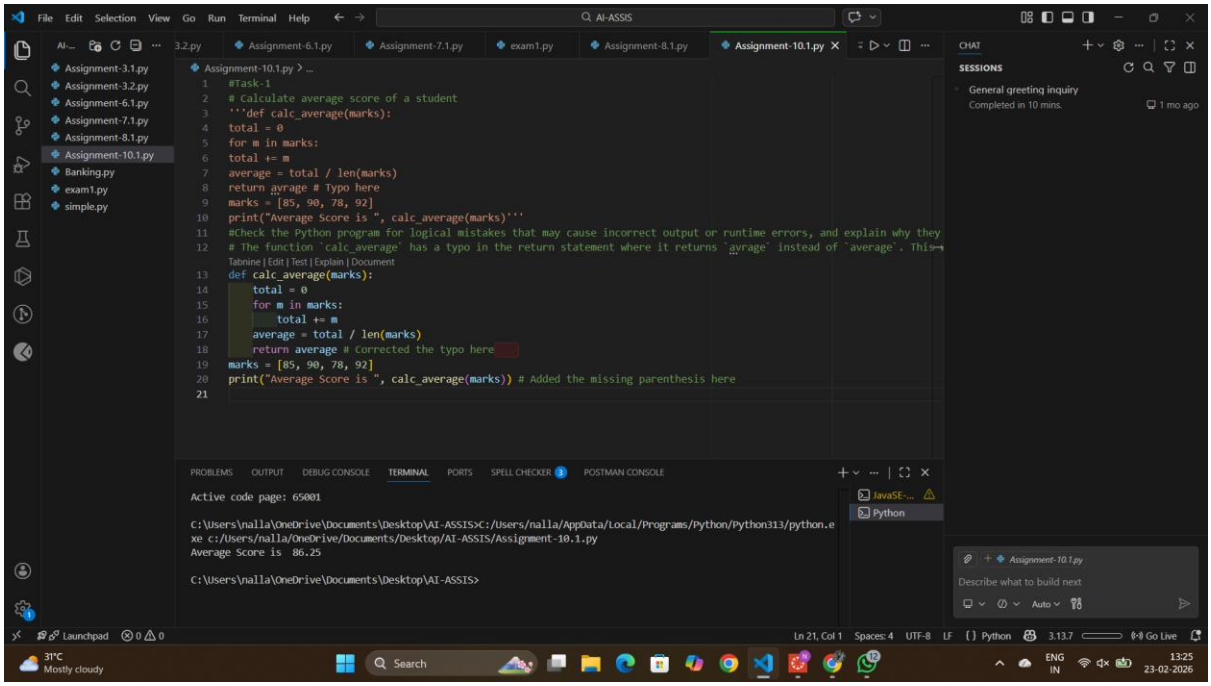
```
# Calculate average score of a student
def calc_average(marks):
total = 0
for m in marks:
total += m
average = total / len(marks)
return avrage # Typo here
marks = [85, 90, 78, 92]
print("Average Score is ", calc_average(marks)
```

Expected Output:

• Corrected and runnable Python code with explanations of the

fixes.

Code and Output:



Analysis:

- The AI detects indentation mistakes, missing brackets, and spelling errors in variables
- It identifies logic flaws such as incorrect variable usage or faulty return values
- The AI corrects the code and produces a runnable program with explanations

Task Description #2 – PEP 8 Compliance

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

guidelines.

Sample Input Code:

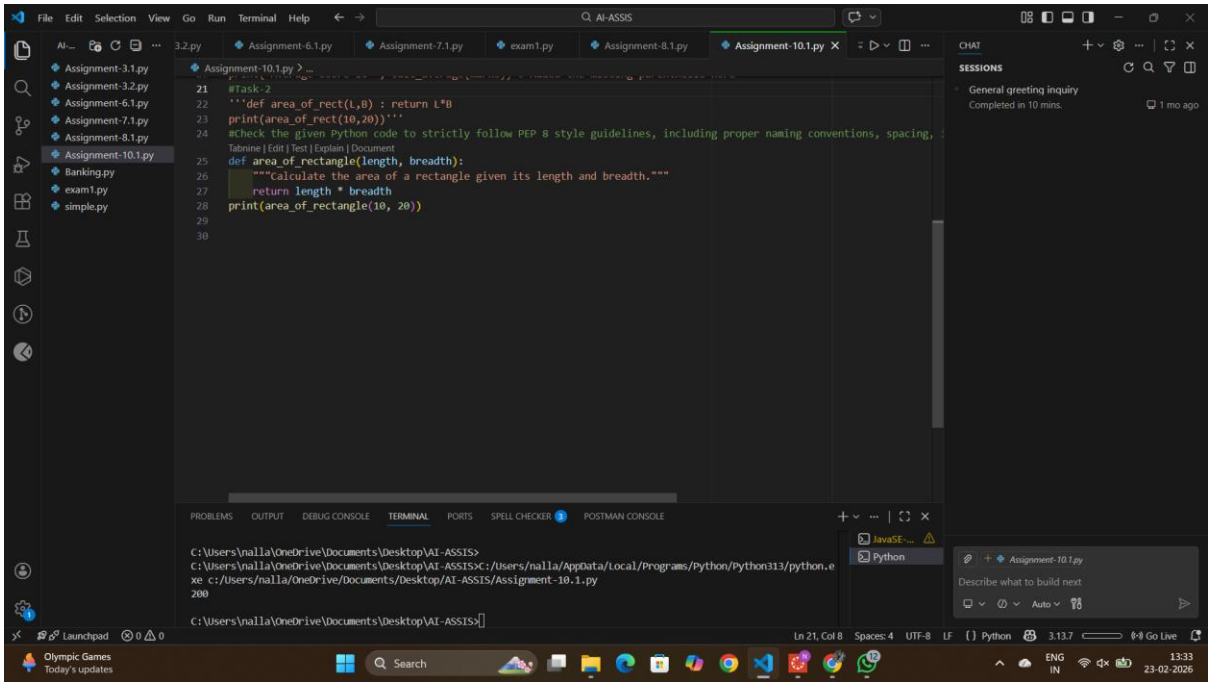
def area_of_rect(L,B) : return L*B

print(area_of_rect(10,20))

Expected Output:

- Well-formatted PEP 8-compliant Python code.

Code and Output:



Analysis:

- The AI enforces proper naming conventions and spacing rules
- It restructures code layout for better readability and consistency
- The program remains functionally unchanged but becomes standard-compliant

Task Description #3 – Readability Enhancement

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

logic.

Sample Input Code:

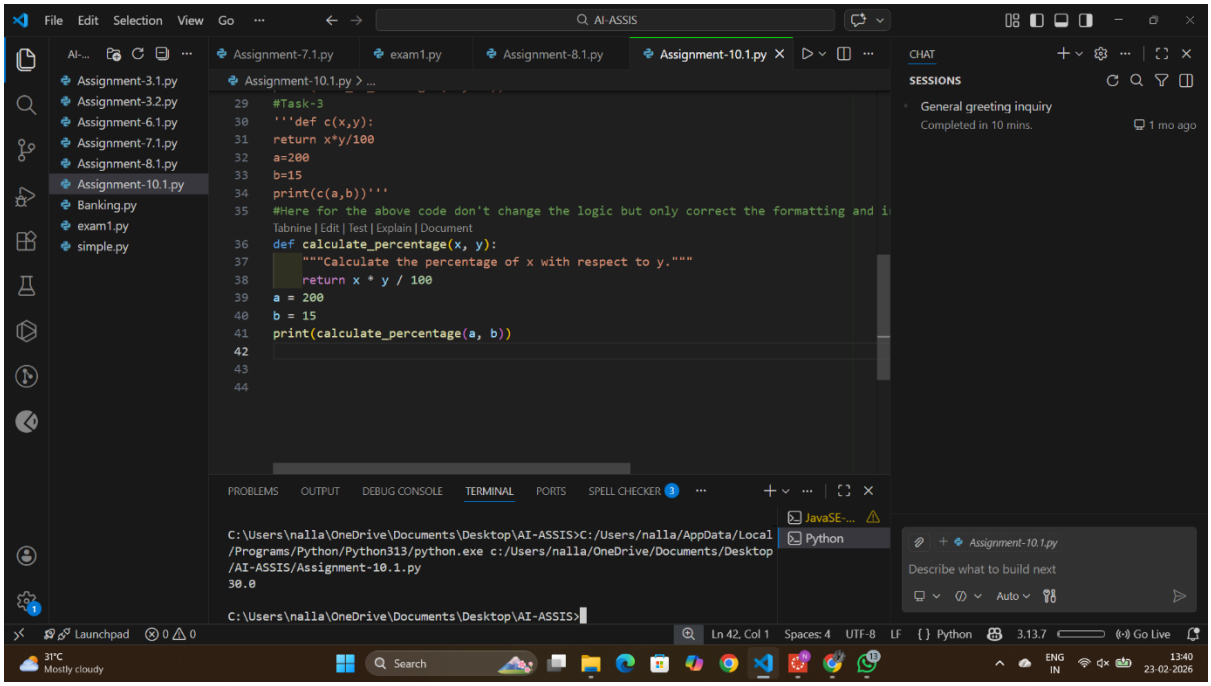
```
def c(x,y):
return x*y/100
a=200
b=15
print(c(a,b))
```

Expected Output:

- Python code with descriptive variable names, inline

comments, and clear formatting.

Code and Output:



Analysis:

- The AI replaces unclear variable names with descriptive ones
- It adds meaningful comments explaining each step
- The code is neatly formatted for easy understanding

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])

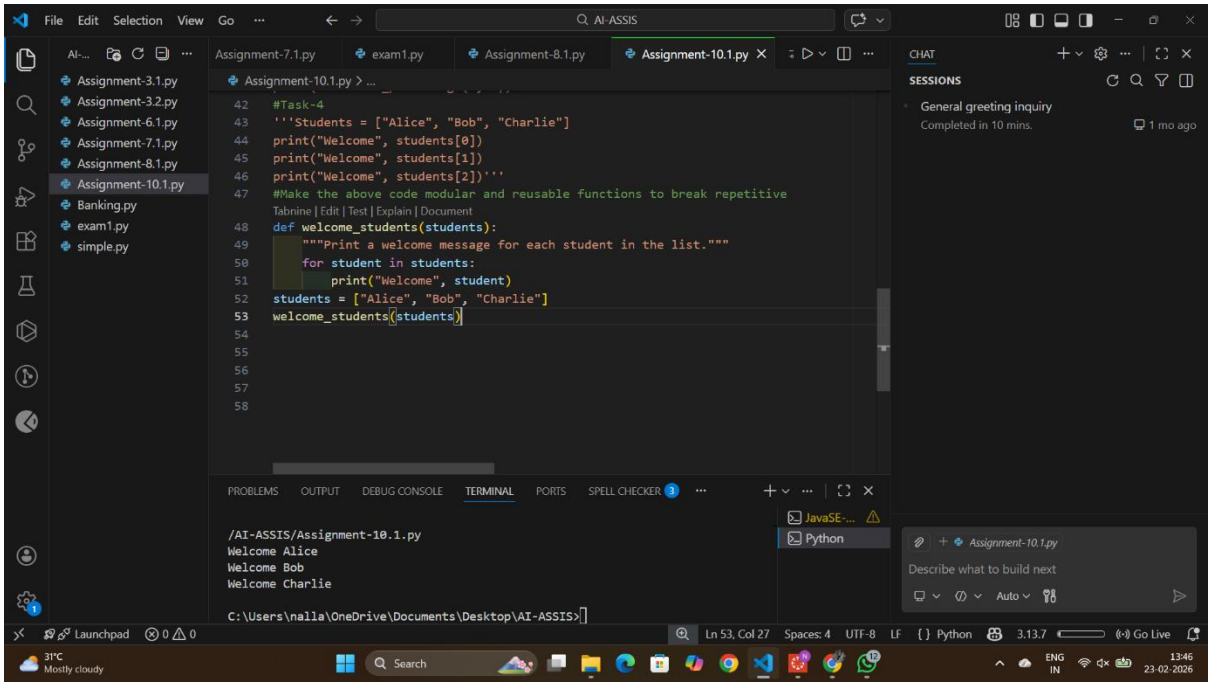
print("Welcome", students[1])

print("Welcome", students[2])

Expected Output:

- Modular code with reusable functions.

Code and Output:



Analysis:

- The AI identifies repeated code blocks
- It converts them into reusable functions
- This makes the program easier to update and maintain

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)

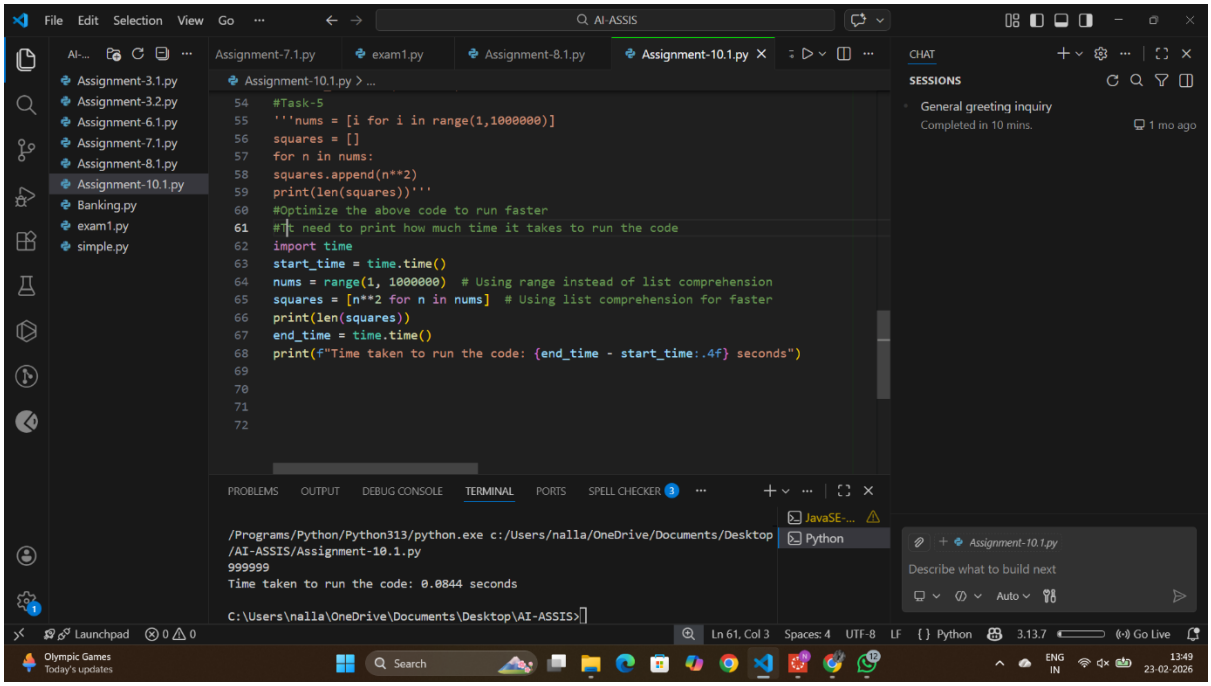
print(len(squares))

Expected Output:

- Optimized code using list comprehensions or vectorized

operations.

Code and Output:



Analysis:

- The AI removes unnecessary loops and intermediate lists
- It uses faster techniques like list comprehensions
- The code executes quicker with lower memory usage

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"

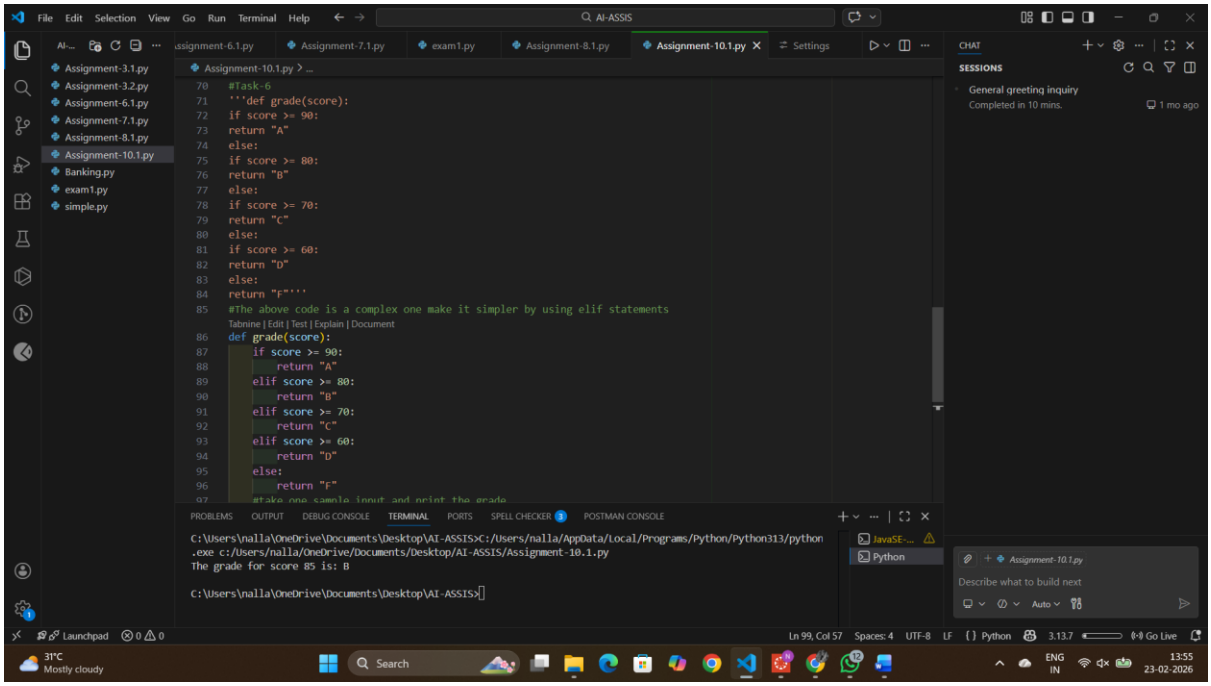
else:

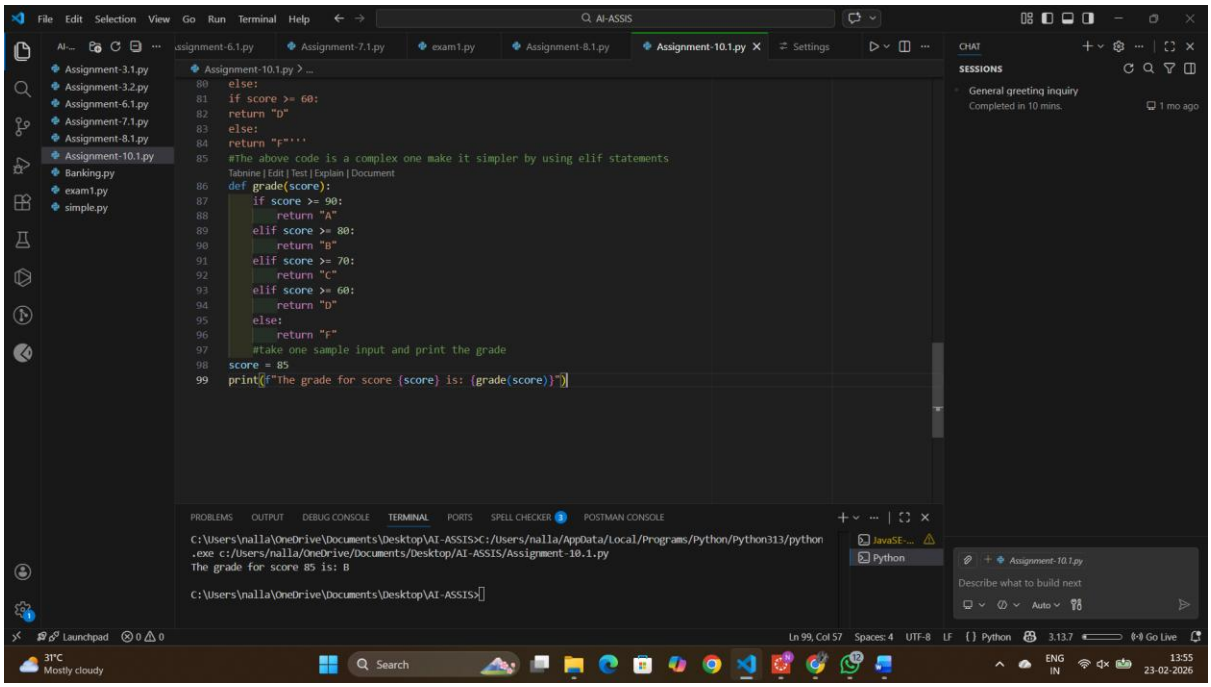
return "F"

Expected Output:

• Cleaner logic using elif or dictionary mapping.

Code and Output:





Analysis:

- The AI simplifies nested conditional statements
- It replaces complex logic with cleaner structures like elif
- The result is shorter, clearer, and easier-to-debug code