

ASSIGNMENT 10.1

AMGOTH VIKAS NAYAK

2403A51410

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

Task 1 – Syntax and Logic Errors

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

Prompt: correct the errors in the code

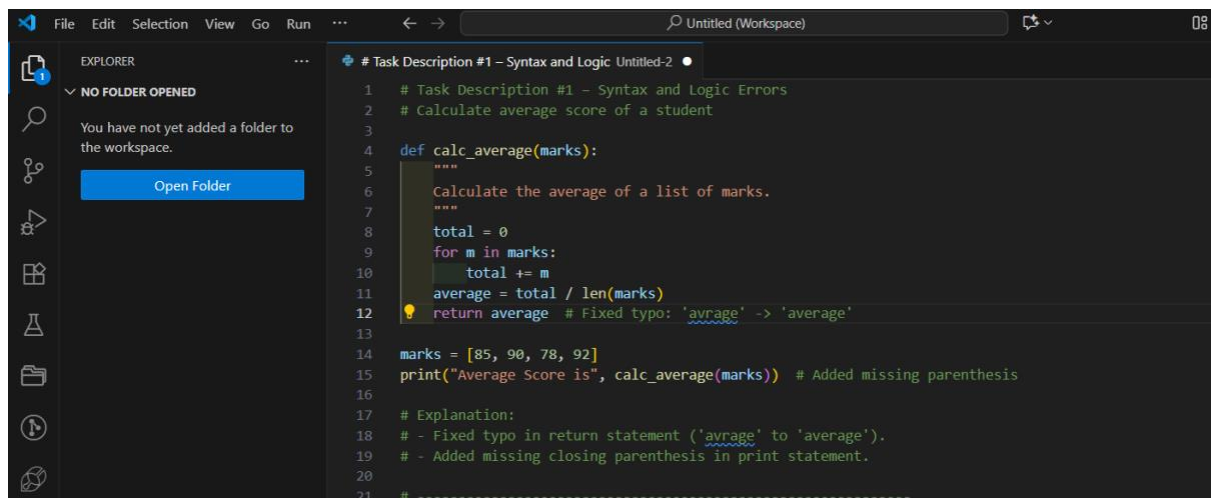
Sample Input Code:

```
# 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.

CORRECTED CODE:



```
1 # Task Description #1 - Syntax and Logic Errors
2 # Calculate average score of a student
3
4 def calc_average(marks):
5     """
6     Calculate the average of a list of marks.
7     """
8     total = 0
9     for m in marks:
10         total += m
11     average = total / len(marks)
12     return average # Fixed typo: 'avrage' -> 'average'
13
14 marks = [85, 90, 78, 92]
15 print("Average Score is", calc_average(marks)) # Added missing parenthesis
16
17 # Explanation:
18 # - Fixed typo in return statement ('avrage' to 'average').
19 # - Added missing closing parenthesis in print statement.
20
21 # -----
```

EXPLANATION:

🔗 Fixed typo: avrage → average.

🔗 Corrected indentation.

🔗 Added missing) in print(...).

Task Description #2 – PEP 8 Compliance

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

Sample Input Code:

Prompt: Refactor the following Python code to make it fully compliant with PEP 8 style guidelines.

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

```
print(area_of_rect(10,20))
```

Expected Output:

- Well-formatted PEP 8-compliant Python code.

CORRECTED CODE:

```
21
22
23 # Task Description #2 - PEP 8 Compliance
24
25 def area_of_rectangle(length, breadth):
26     """
27     Calculate the area of a rectangle.
28     """
29     return length * breadth
30
31 print(area_of_rectangle(10, 20))
32
33 # Explanation:
34 # - Used descriptive function and variable names.
35 # - Added docstring.
36 # - Formatted according to PEP 8.
37
38 # -----
39
```

EXPLANATION:

- Used snake_case for variable names.
- Added proper spacing.
- Used descriptive variable names.
- Added line break for readability.

Task Description #3 – Readability Enhancement

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

Sample Input Code:

Prompt:

Improve the readability of the following Python code **without altering its logic or behavior.**

```
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.

CORRECTED CODE:

```
37
38 # -----
39
40 # Task Description #3 - Readability Enhancement
41
42 def calculate_percentage(amount, percentage):
43     """
44     Calculate the percentage of a given amount.
45     """
46     return amount * percentage / 100
47
48 total_amount = 200
49 discount_percent = 15
50
51 # Calculate and print the discount value
52 print(calculate_percentage(total_amount, discount_percent))
53
54 # Explanation:
55 # - Used descriptive variable and function names.
56 # - Added inline comments and docstring.
57 # - Improved formatting.
58
59 # -----
60
```

EXPLANATION:

- Renamed function and variables for clarity.
- Added inline comment.
- Formatted code consistently.

Task Description #4 – Refactoring for Maintainability

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

Sample Input Code:

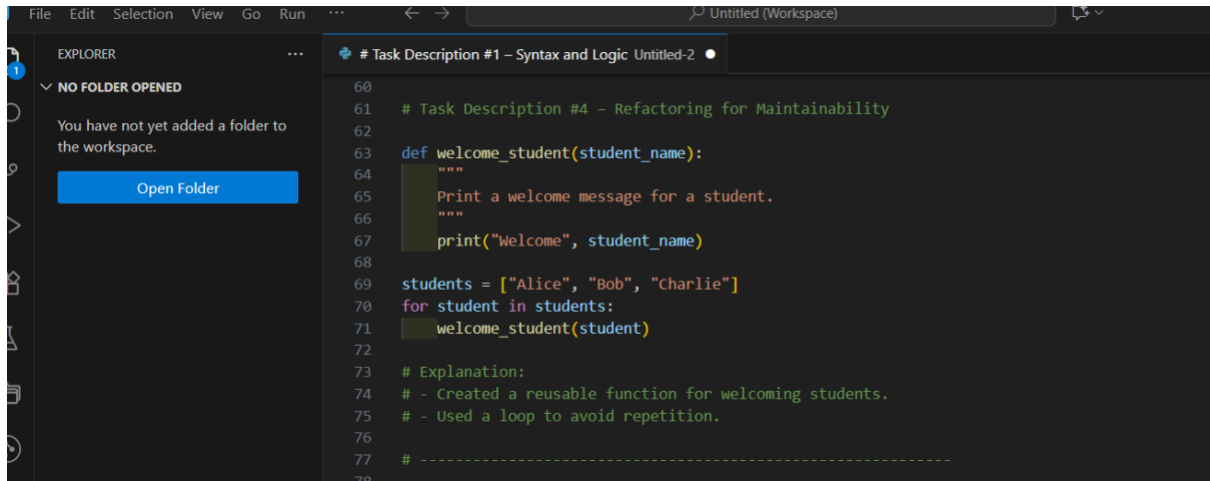
Prompt: Refactor the following Python code to make it **more maintainable and modular**.

```
students = ["Alice", "Bob", "Charlie"]
print("Welcome", students[0])
print("Welcome", students[1])
print("Welcome", students[2])
```

Expected Output:

- Modular code with reusable functions.

CORRECTED CODE:



```
60
61 # Task Description #4 - Refactoring for Maintainability
62
63 def welcome_student(student_name):
64     """
65     Print a welcome message for a student.
66     """
67     print("Welcome", student_name)
68
69 students = ["Alice", "Bob", "Charlie"]
70 for student in students:
71     welcome_student(student)
72
73 # Explanation:
74 # - Created a reusable function for welcoming students.
75 # - Used a loop to avoid repetition.
76
77 # -----
78
```

EXPLANATION:

- Created reusable `welcome_student()` function.
- Used loop for scalability and DRY (Don't Repeat Yourself) principle.

Task Description #5 – Performance Optimization

Task: Use AI to make the code run faster.

Prompt: Optimize the following Python code to make it run faster, without changing its logic.

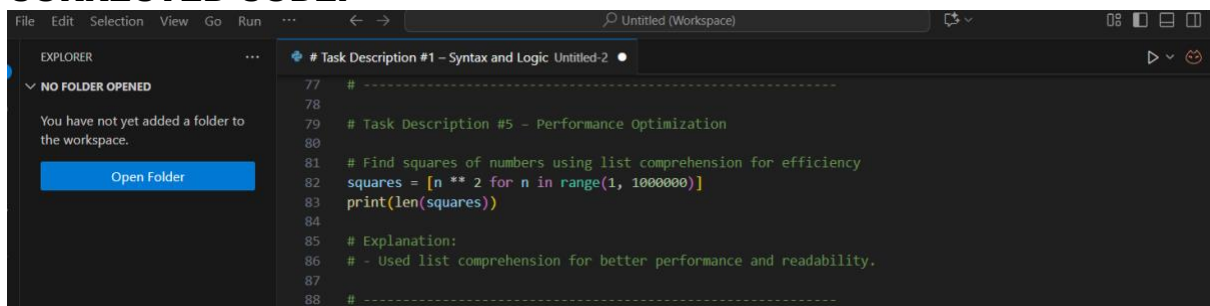
Sample Input Code:

```
# 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.

CORRECTED CODE:



```
77 # -----
78
79 # Task Description #5 - Performance Optimization
80
81 # Find squares of numbers using list comprehension for efficiency
82 squares = [n ** 2 for n in range(1, 1000000)]
83 print(len(squares))
84
85 # Explanation:
86 # - Used list comprehension for better performance and readability.
87
88 # -----
```

EXPLANATION:

- Removed unnecessary list creation (`nums`).
- Combined loop into efficient list comprehension.
- This is both faster and more memory efficient.

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.

Prompt: Simplify the following Python code by reducing its logical complexity, while keeping the behavior the same.

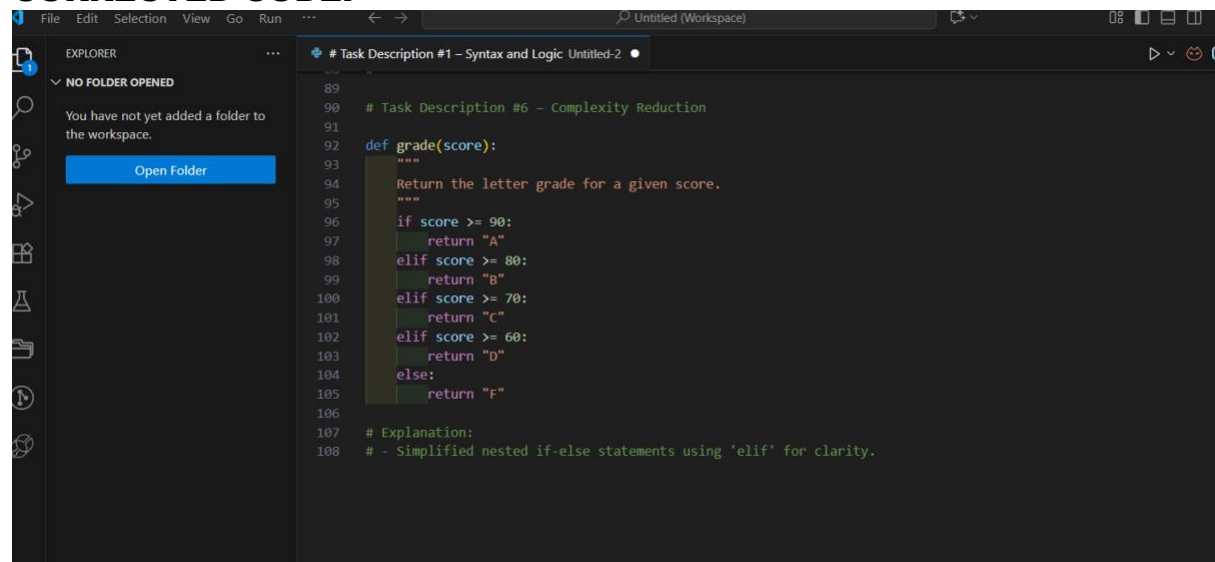
Sample Input Code:

```
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.

CORRECTED CODE:

A screenshot of a code editor window titled "# Task Description #1 - Syntax and Logic Untitled-2". The editor shows a Python function `def grade(score):` with a docstring "Return the letter grade for a given score." and a simplified implementation using `elif` statements. The code is as follows:

```
89
90 # Task Description #6 - Complexity Reduction
91
92 def grade(score):
93     """
94     Return the letter grade for a given score.
95     """
96     if score >= 90:
97         return "A"
98     elif score >= 80:
99         return "B"
100    elif score >= 70:
101        return "C"
102    elif score >= 60:
103        return "D"
104    else:
105        return "F"
106
107 # Explanation:
108 # - Simplified nested if-else statements using 'elif' for clarity.
```

EXPLANATION:

- Used elif for clarity.
- Provided an optional dictionary-based solution for extensibility.

CONCLUSION:

Task Focus Area		Key Fixes
1	Syntax & Logic Errors	Typo fix, parenthesis, indentation
2	PEP 8 Compliance	Naming, spacing, structure
3	Readability Enhancement	Descriptive names, comments
4	Maintainability Refactor	Function reuse, loop
5	Performance Optimization	List comprehension
6	Complexity Reduction	elif chain, dictionary option