ASSIGNMENT 10.1

G.ASHOK

2503A52L21

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

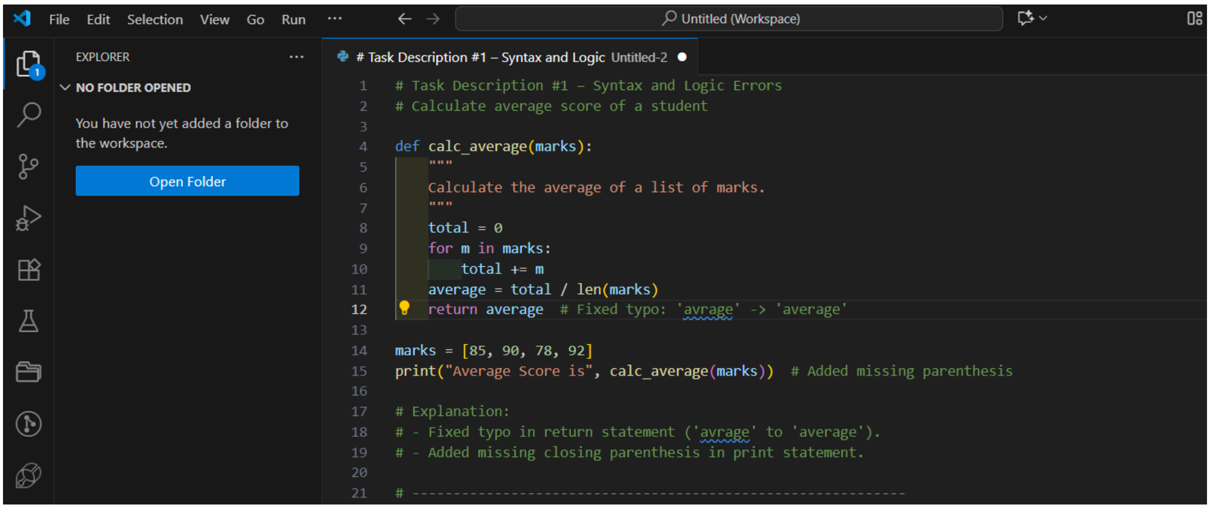
Task 1 – Syntax and Logic Errors

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:



File
Edit
Selection
View
Go
Run
Untitled (Workspace)
EXPLORER
NO FOLDER OPENED
You have not yet added a folder to the workspace.
Open Folder
# Task Description #1 – Syntax and Logic Untitled-2

```python
# Task Description #1 – Syntax and Logic Errors
# Calculate average score of a student

def calc_average(marks):
    """
    Calculate the average of a list of marks.
    """
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return average  # Fixed typo: 'avrage' -> 'average'

marks = [85, 90, 78, 92]
print("Average Score is", calc_average(marks))  # Added missing parenthesis

# Explanation:
# - Fixed typo in return statement ('avrage' to 'average').
# - Added missing closing parenthesis in print statement.

# ------------------------------------------------------------
```

EXPLANATION:

⍰ Fixed typo: avrage → average.

⍰ Corrected indentation.

⍰ Added missing) in print(...).

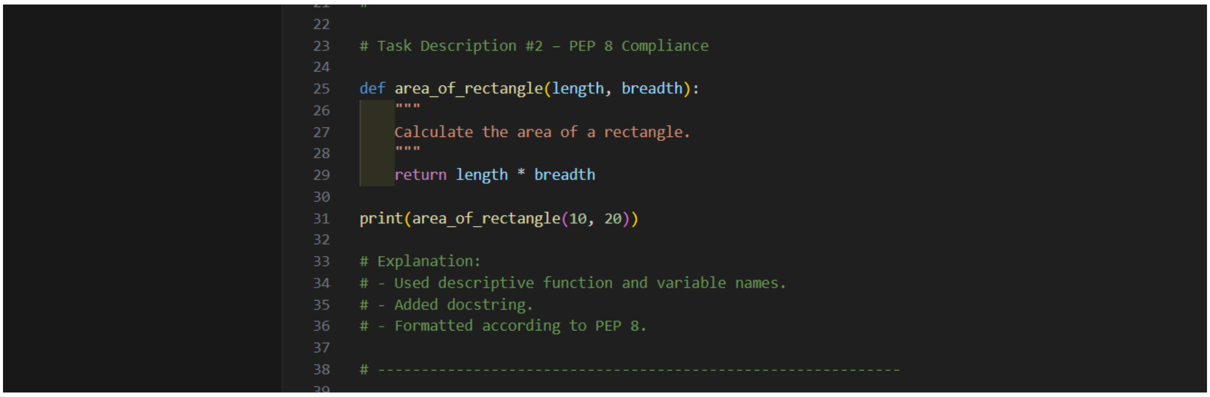
Task Description #2 – PEP 8 Compliance

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:



```
# Task Description #2 – PEP 8 Compliance

def area_of_rectangle(length, breadth):
    """
    Calculate the area of a rectangle.
    """
    return length * breadth

print(area_of_rectangle(10, 20))

# Explanation:
# - Used descriptive function and variable names.
# - Added docstring.
# - Formatted according to PEP 8.

# ------------------------------------------------------------
```

EXPLANATION:

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

Task Description #3 – Readability Enhancement

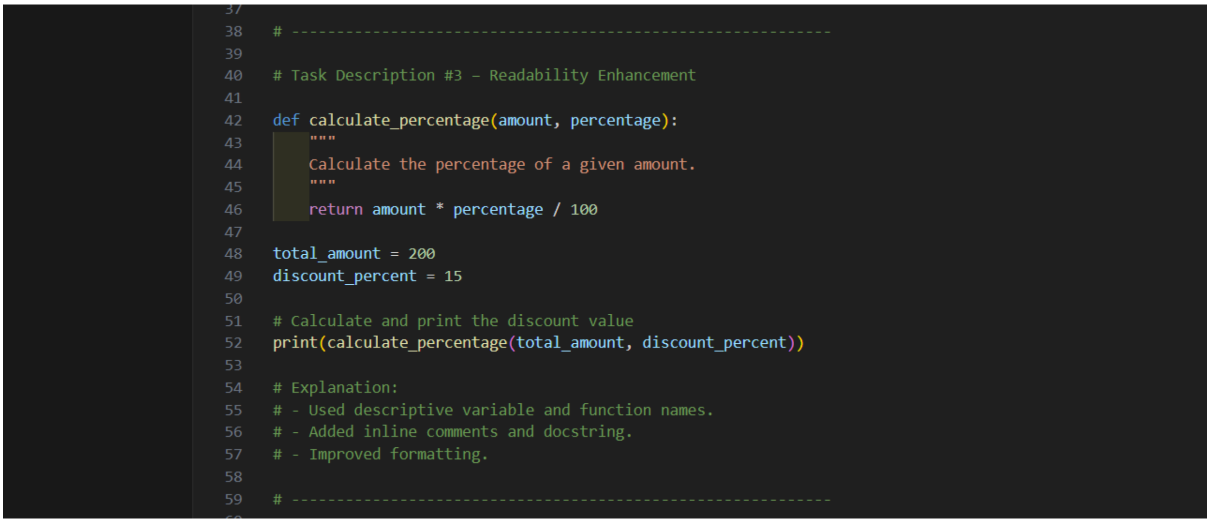
Task: Use AI to make code more readable without changing its logic.
Sample Input Code:
Prompt:
Improve the readability of the following Python code **without altering its logic or behavior**.

```
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:



```python
# ------------------------------------------------------------

# Task Description #3 – Readability Enhancement

def calculate_percentage(amount, percentage):
    """
    Calculate the percentage of a given amount.
    """
    return amount * percentage / 100

total_amount = 200
discount_percent = 15

# Calculate and print the discount value
print(calculate_percentage(total_amount, discount_percent))

# Explanation:
# - Used descriptive variable and function names.
# - Added inline comments and docstring.
# - Improved formatting.

# ------------------------------------------------------------
```

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:

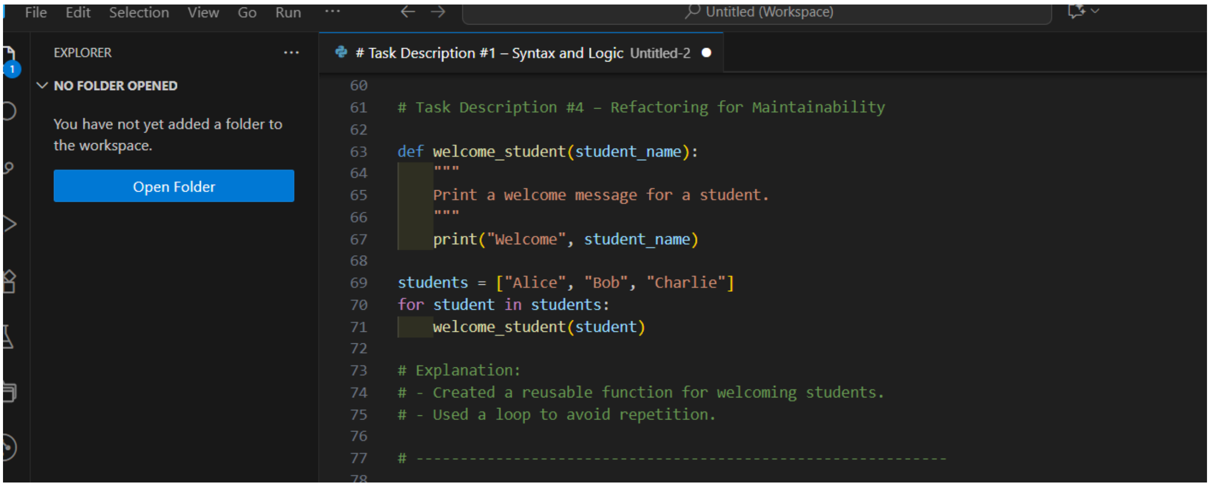
Prompt: Refactor the following Python code to make it **more maintainable and modular**.

```
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:


File Edit Selection View Go Run ···
Untitled (Workspace)
EXPLORER
NO FOLDER OPENED
Open Folder
# Task Description #1 – Syntax and Logic Untitled-2

You have not yet added a folder to the workspace.

```
# Task Description #4 – Refactoring for Maintainability

def welcome_student(student_name):
    """
    Print a welcome message for a student.
    """
    print("Welcome", student_name)

students = ["Alice", "Bob", "Charlie"]
for student in students:
    welcome_student(student)

# Explanation:
# - Created a reusable function for welcoming students.
# - Used a loop to avoid repetition.

# ------------------------------------------------------------
```

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.

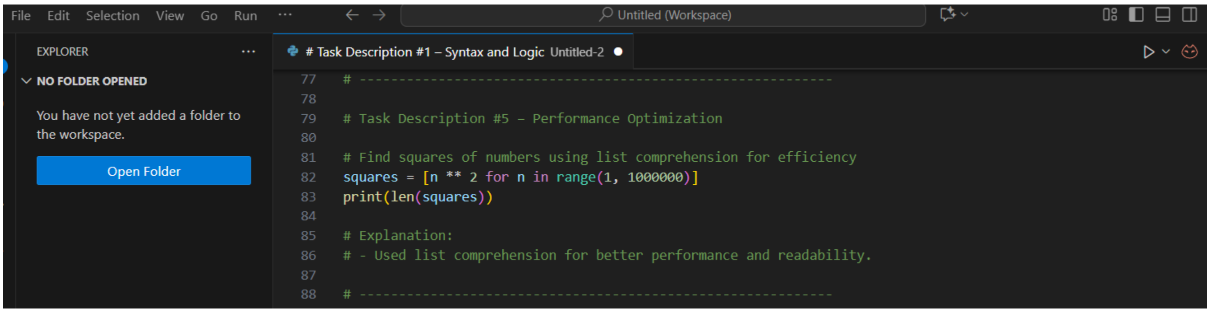
Sample Input Code:

```
# 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:


File Edit Selection View Go Run ···
Untitled (Workspace)
EXPLORER
NO FOLDER OPENED
Open Folder
# Task Description #1 – Syntax and Logic Untitled-2

You have not yet added a folder to the workspace.

```
# ------------------------------------------------------

# Task Description #5 – Performance Optimization

# Find squares of numbers using list comprehension for efficiency
squares = [n ** 2 for n in range(1, 1000000)]
print(len(squares))

# Explanation:
# - Used list comprehension for better performance and readability.

# ------------------------------------------------------
```

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.

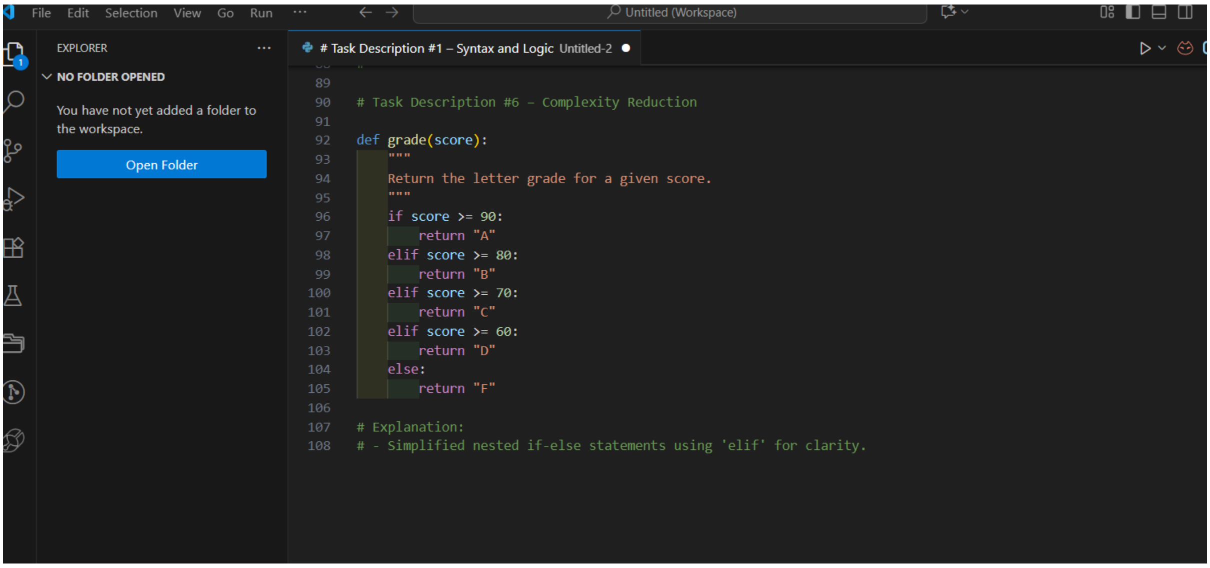
Sample Input Code:

```
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:



You have not yet added a folder to the workspace.

```python

# Task Description #6 – Complexity Reduction

def grade(score):
    """
    Return the letter grade for a given score.
    """
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

# Explanation:
# - 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