

AI-Ass-10.1

Name:T.Shivani

Ht.No:2303A51312

Batch:05

Task: Description #1 – Syntax and Logic Errors. 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: def calc_average(marks):

```
total = 0

for m in marks:

    total += m

average = total / len(marks)

return average
```

```
marks = [85, 90, 78, 92]
```

```
print("Average Score is ", calc_average(marks))
```

```
sample_text.txt
sentiment_analysis_pr...
sentiment_prompt_a...
sort_comparator.py
text_file_creator.read...
units_calculator.py
user_database.db
users.db
word_frequency_coun... PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Shivani T\OneDrive\Desktop\AI-assisted> & "c:/Users/Shivani T/AppData/Local/Python/pythoncore-3.14-64/python.exe" "c:/Users/Shivani T/.vscode/extensions/ms-python.debugpy-2025.1*"
8.0-win32-x64\bundled\libs\debug\launcher '58204' '--' 'C:/Users/Shivani T/OneDrive/Desktop/AI-assisted\Ass-10.1.py'
Average Score is 86.25
PS C:\Users\Shivani T\OneDrive\Desktop\AI-assisted>
```

The screenshot shows the VS Code interface with the terminal tab active. The command `python -m venv env` was run, and the output shows the creation of a virtual environment named `env`. The terminal also displays the execution of the Python script `Ass-10.1.py`, which prints the average score as 86.25.

Explanation: The original code had a typo in the return statement

(avrageinstead of average).

The print statement was missing a closing parenthesis.

Task: Description #2 – PEP 8 Compliance 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: def area_of_rect(length, breadth):

```
    return length * breadth
#example usage
print(area_of_rect(10, 20))
```

```
sentiment_analysis_pr...
sentiment_prompt_a...
sort_comparator.py
text_file_creator.read...
units_calculator.py
user_database.db
users.db
word_frequency_coun... PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\Shivani T\OneDrive\Desktop\AI-assisted> & "c:/Users/Shivani T/AppData/Local/Python/pythoncore-3.14-64/python.exe" "c:/Users/Shivani T/OneDrive/Desktop/AI-assisted\Ass-10.1.py"
200
PS C:\Users\Shivani T\OneDrive\Desktop\AI-assisted>
```

The screenshot shows the VS Code interface with the terminal tab active. The command `python -m venv env` was run, and the output shows the creation of a virtual environment named `env`. The terminal also displays the execution of the Python script `Ass-10.1.py`, which results in an error code of 200.

Explanation: The original code had several PEP 8 compliance issues:

1. No spaces around the `=` operator in function definition.
2. No spaces around the `*` operator in the return statement.

3. No blank lines between function definition and usage.

4. No proper indentation in comments.

Task: Description #3 – Readability Enhancement 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.

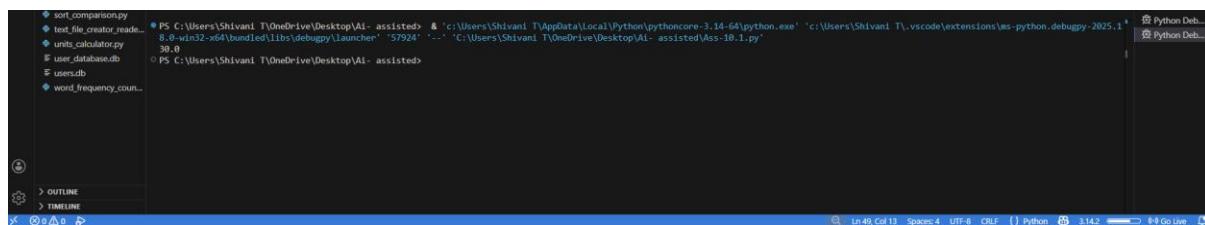
Code: def calculate_simple_interest(principal, rate):

```
    return principal * rate / 100
```

```
principal_amount = 200
```

```
interest_rate = 15
```

```
print(calculate_simple_interest(principal_amount, interest_rate))
```



Explanation: The original code had non-descriptive variable names (c, x, y, a, b).

The refactored code uses descriptive variable names (calculate_simple_interest, principal, rate, principal_amount, interest_rate) and includes an inline comment for clarity.

Task: Description #4 – Refactoring for Maintainability 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.

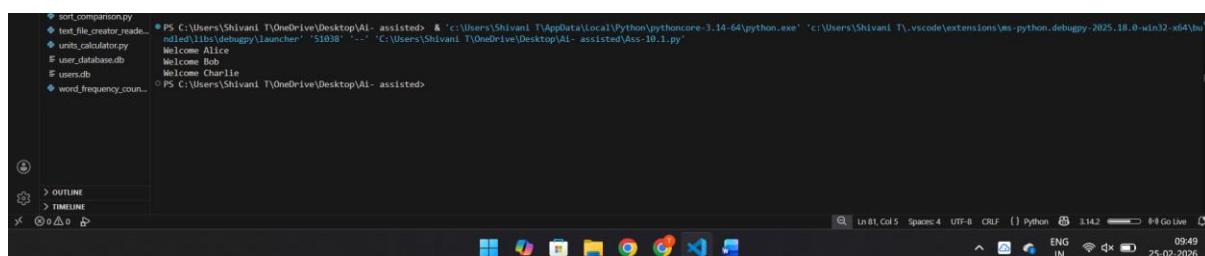
Code: def welcome_student(student_name):

```
    print("Welcome", student_name)
```

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

for student in students:

```
    welcome_student(student)
```



Explanation: The original code had repetitive print statements for each student.

By creating a function `welcome_student`, we can reuse it for each student in the list, making the code more maintainable and cleaner.

Task: Description #5 – Performance Optimization. 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))
```

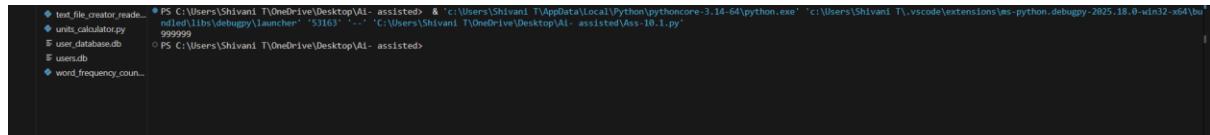
Expected Output:

- Optimized code using list comprehensions or vectorized operations.

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

```
squares = [n**2 for n in nums]
```

```
print(len(squares))
```



Explanation: The original code used a for loop to append squares to a list, which is less efficient.

The refactored code uses a list comprehension, which is more concise and faster for creating lists.

Task: Description #6 – Complexity Reduction.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: def grade(score):

```
if score >= 90:
```

```
    return "A"
```

```
elif score >= 80:
```

```
    return "B"
```

```
elif score >= 70:
```

```
    return "C"
```

```
elif score >= 60:
```

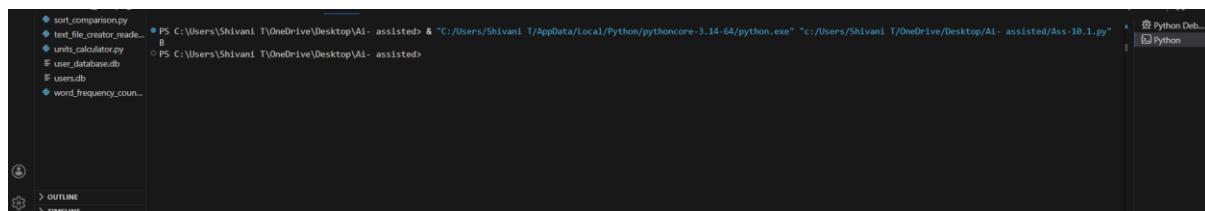
```
    return "D"
```

```
else:
```

```
    return "F"
```

#example usage

```
print(grade(85))
```



Explanation: The original code had nested if-else statements which made it less readable and more complex.

#By using elif, we can simplify the logic and improve readability.