ASSIGNMENT 10.1

NAME : K. Bhuvaneshwar Reddy.

BATCH NUMBER: 15.

HALLTICKET NUMBER:2403A52416.

SUBJECT: AI CODING

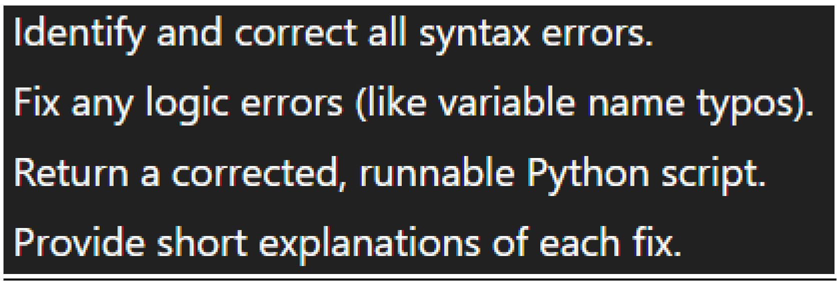
TASK-1

QUESTION:

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)
```

PROMPT:

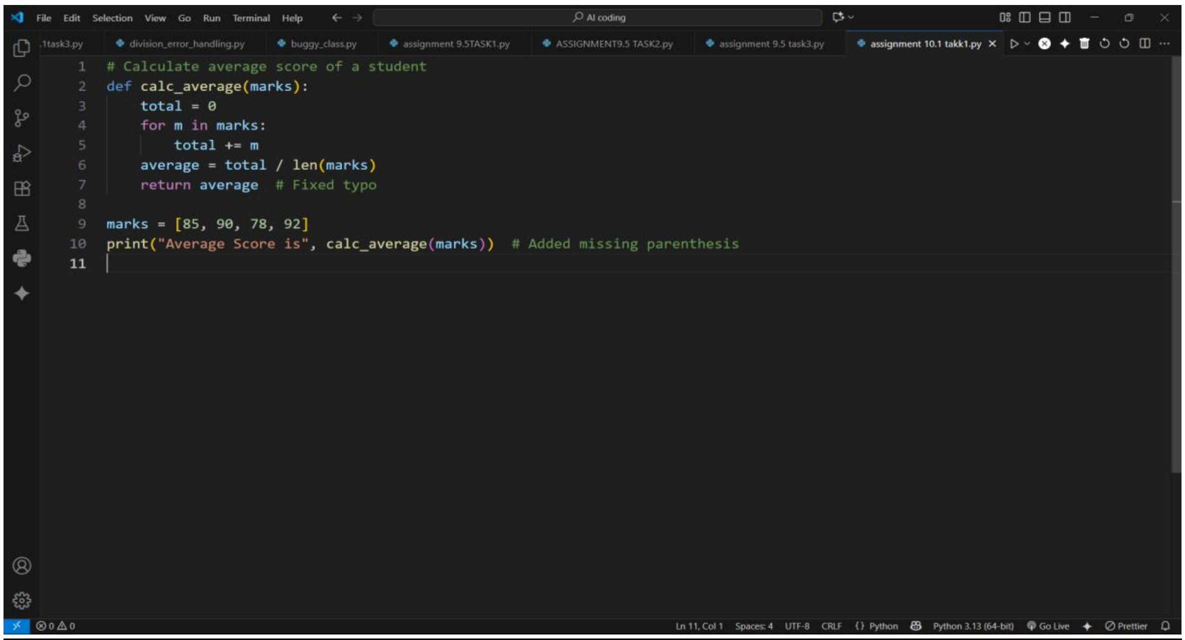

Identify and correct all syntax errors.

Fix any logic errors (like variable name typos).

Return a corrected, runnable Python script.

Provide short explanations of each fix.

CODE:

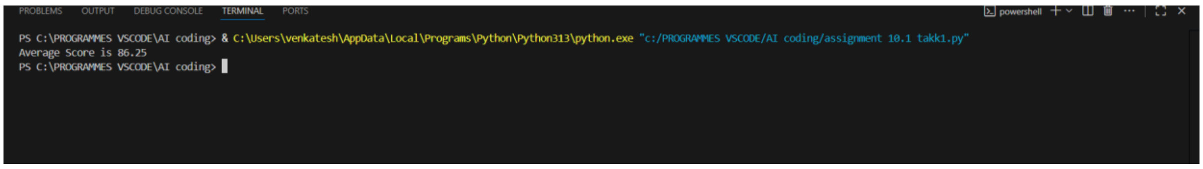


```python
# Calculate average score of a student
def calc_average(marks):
    total = 0
    for m in marks:
        total += m
    average = total / len(marks)
    return average  # Fixed typo

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

```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI coding/assignment 10.1 takk1.py"
Average Score is 86.25
PS C:\PROGRAMMES VSCODE\AI coding>
```

CONCLUSION:

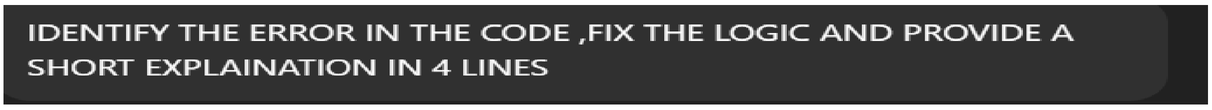
The faulty Python script had issues with indentation, a typo in the return variable, and a missing parenthesis.
After corrections, the code runs successfully and calculates the average of the given marks.
The function now correctly returns the computed average.
Final output displays: **Average Score is 86.25**.

TASK-2

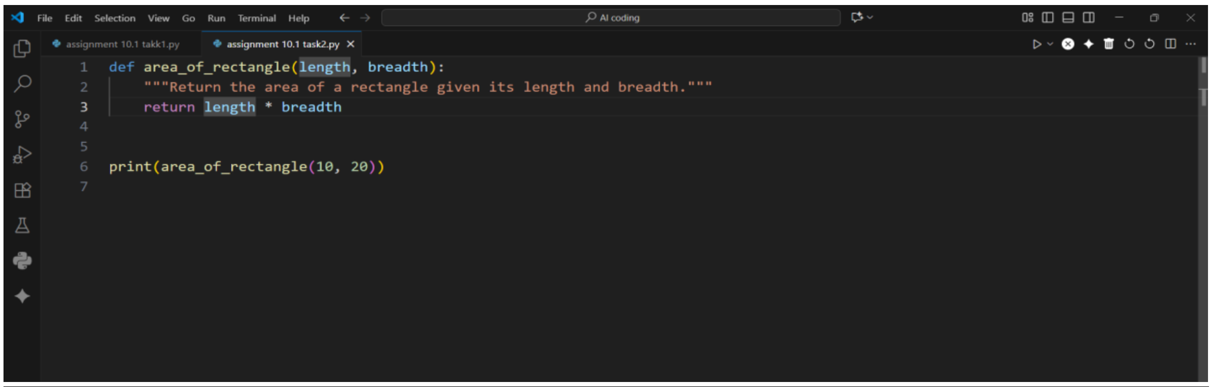
QUESTION:

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

PROMPT:


IDENTIFY THE ERROR IN THE CODE ,FIX THE LOGIC AND PROVIDE A SHORT EXPLAINATION IN 4 LINES

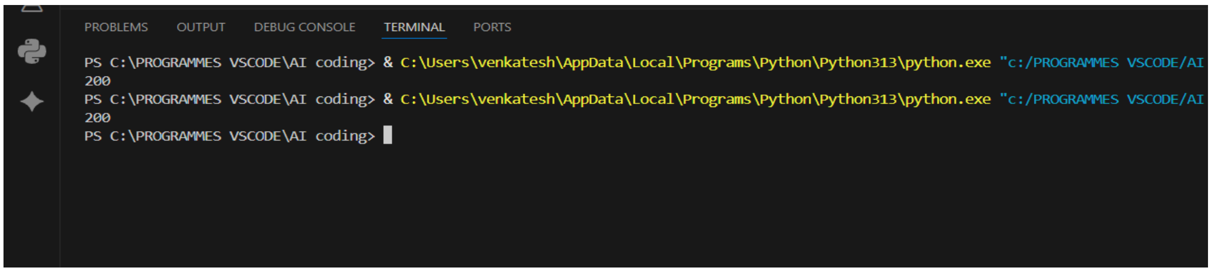
CODE:



```python
def area_of_rectangle(length, breadth):
    """Return the area of a rectangle given its length and breadth."""
    return length * breadth


print(area_of_rectangle(10, 20))
```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI
200
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI
200
PS C:\PROGRAMMES VSCODE\AI coding> 
```

CONCLUSION:

The function calculates the area of a rectangle using length and breadth.
It follows PEP 8 guidelines with proper formatting and spacing.
Descriptive names and a docstring enhance code clarity.
The program executes correctly and outputs **200**.

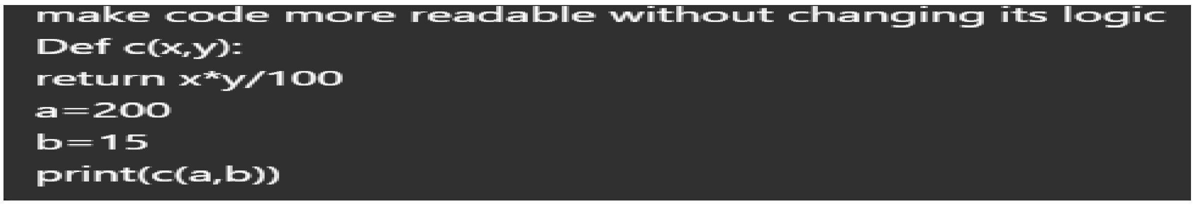
TASK-3

QUESTION:

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,

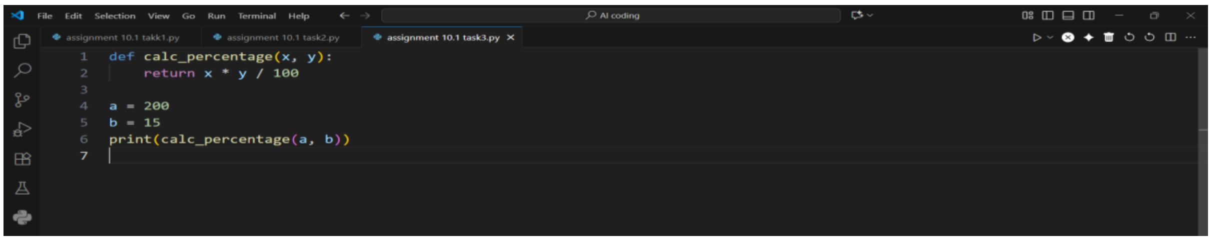
and clear formatting

PROMPT:



```
make code more readable without changing its logic
Def c(x,y):
return x*y/100
a=200
b=15
print(c(a,b))
```

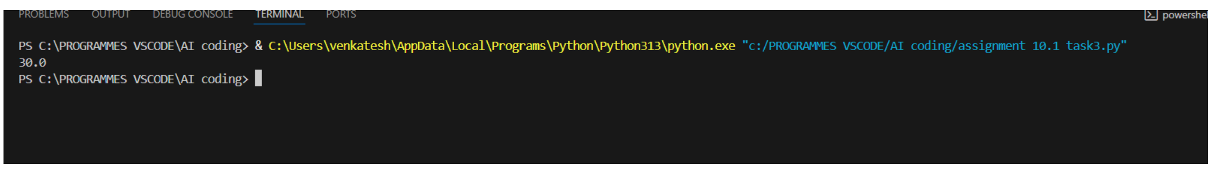
CODE:



```python
def calc_percentage(x, y):
    return x * y / 100

a = 200
b = 15
print(calc_percentage(a, b))
```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI coding/assignment 10.1 task3.py"
30.0
PS C:\PROGRAMMES VSCODE\AI coding> 
```

CONCLUSION:

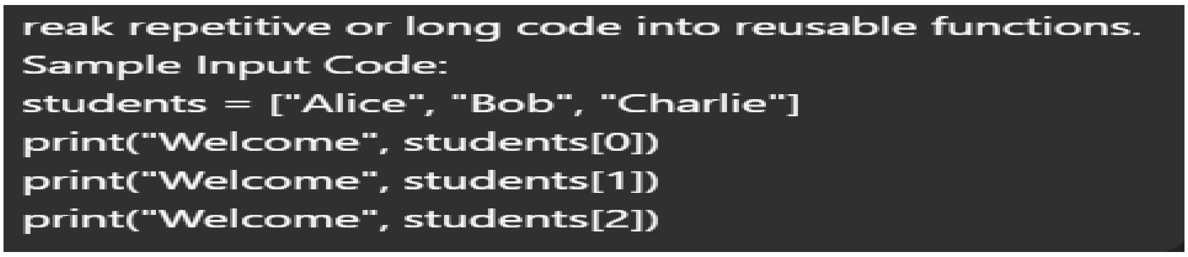
The function calc_percentage multiplies two numbers and divides by 100 to calculate a percentage.
It improves readability by using a clear function name.
The variables a and b are passed as inputs to the function.
The program runs successfully and outputs 30.0.

TASK-4

QUESTION:

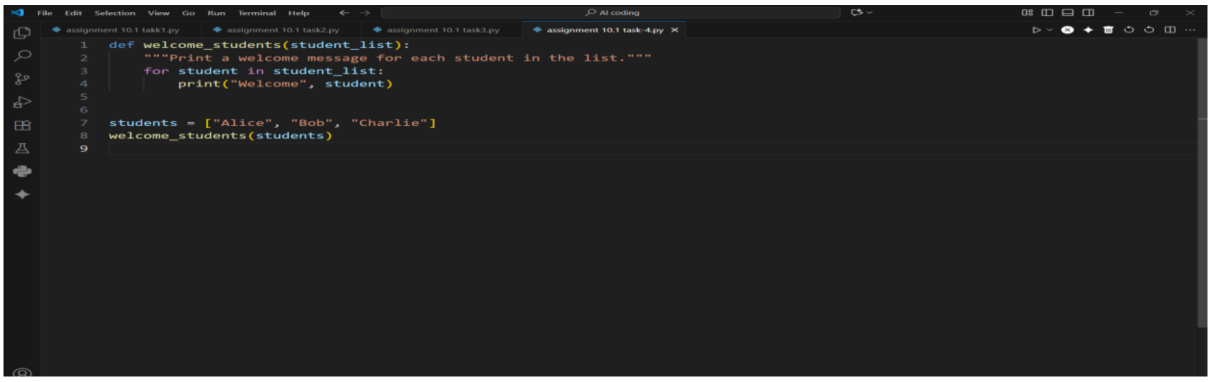
Task Description #4 – Refactoring for Maintainability
Task: Use AI to break repetitive or long code into reusable functions.
Sample Input Code:
students = ["Alice", "Bob", "Charlie"]
print("Welcome", students[0])
print("Welcome", students[1])
print("Welcome", students[2])
Expected Output:
- **Modular code with reusable functions**

PROMPT:



```
reak 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])
```

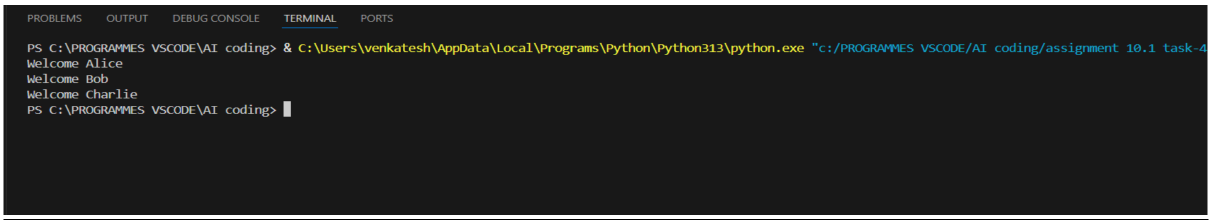
CODE:



```python
def welcome_students(student_list):
    """Print a welcome message for each student in the list."""
    for student in student_list:
        print("Welcome", student)


students = ["Alice", "Bob", "Charlie"]
welcome_students(students)
```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI coding/assignment 10.1 task-4
Welcome Alice
Welcome Bob
Welcome Charlie
PS C:\PROGRAMMES VSCODE\AI coding> 
```

CONCLUSION:

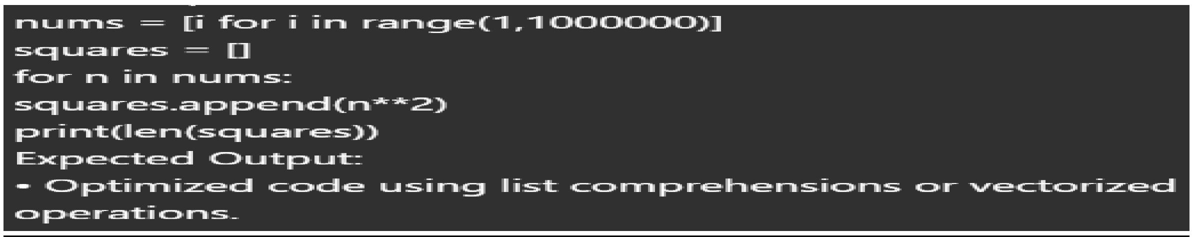
The repetitive print statements are replaced with a reusable function using a loop. This makes the code cleaner, scalable, and easier to maintain.

TASK-5

QUESTION:

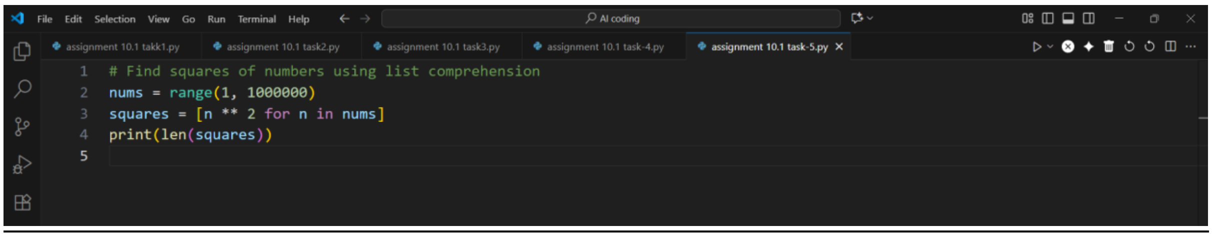
Task Description #5 – Performance Optimization
Task: Use AI to make the code run faster.
Sample Input Code:
# Find squares of numbers
nums = [i for i in range(1,1000000)]
squares = []
for n in nums:
squares.append(n2)**
print(len(squares))
Expected Output:
• Optimized code using list comprehensions or vectorized operations.

PROMPT:



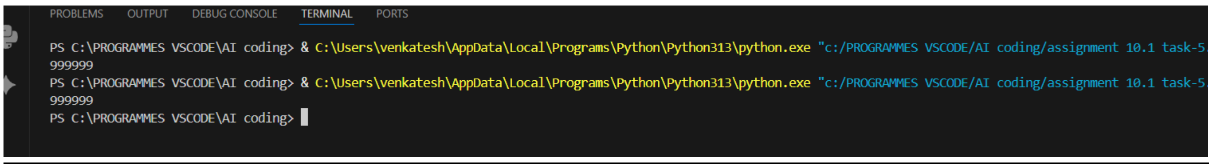
```
nums = [i for i in range(1,1000000)]
squares = []
for n in nums:
squares.append(n**2)
print(len(squares))
Expected Output:
• Optimized code using list comprehensions or vectorized
operations.
```

CODE:


File Edit Selection View Go Run Terminal Help
AI coding
assignment 10.1 takk1.py
assignment 10.1 task2.py
assignment 10.1 task3.py
assignment 10.1 task-4.py
assignment 10.1 task-5.py

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

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI coding/assignment 10.1 task-5
999999
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI coding/assignment 10.1 task-5
999999
PS C:\PROGRAMMES VSCODE\AI coding> ▌
```

CONCLUSION:

The first optimized version replaces the loop with a list comprehension.
It runs faster and uses less code while still producing the same result.

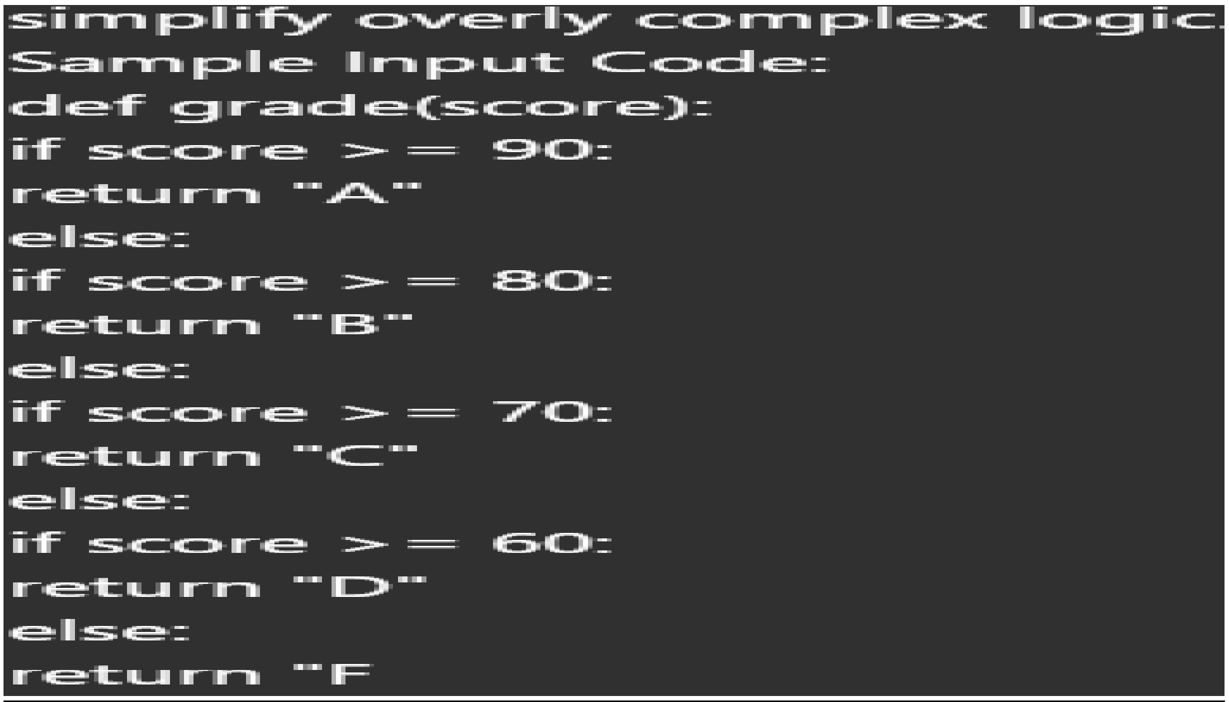
TASK-6

QUESTION:

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"

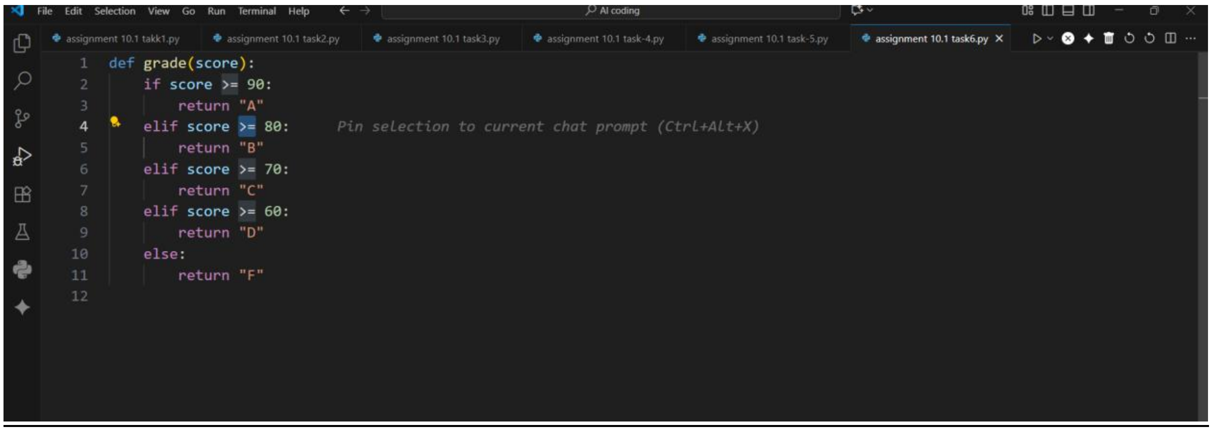
else:
if score >= 70:
return "C"
else:
if score >= 60:
return "D"
else:
return "F"
Expected Output:
• Cleaner logic using elif or dictionary mapping

PROMPT:



```
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
```

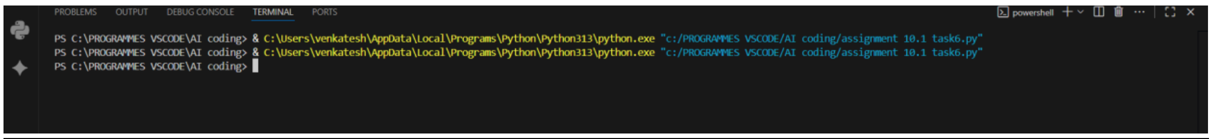
CODE:



```python
def grade(score):
    if score >= 90:
        return "A"
    elif score >= 80:      Pin selection to current chat prompt (Ctrl+Alt+X)
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"

```

OUTPUT:



```
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI coding/assignment 10.1 task6.py"
PS C:\PROGRAMMES VSCODE\AI coding> & C:\Users\venkatesh\AppData\Local\Programs\Python\Python313\python.exe "c:/PROGRAMMES VSCODE/AI coding/assignment 10.1 task6.py"
PS C:\PROGRAMMES VSCODE\AI coding> 
```

CONCLUSION:

The original code had deeply nested if-else statements, making it harder to read.
It was simplified using elif, which makes the logic clearer and easier to follow.
The function now directly checks conditions in sequence without unnecessary nesting.
It correctly assigns grades from **A to F** based on the score.