ASSIGNMENT – 10

NAME : SHAIK SALIHA

ROLL NO: 2503A52L13

BATCH : 16

Task - 1: 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

PROMPT : 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)
```

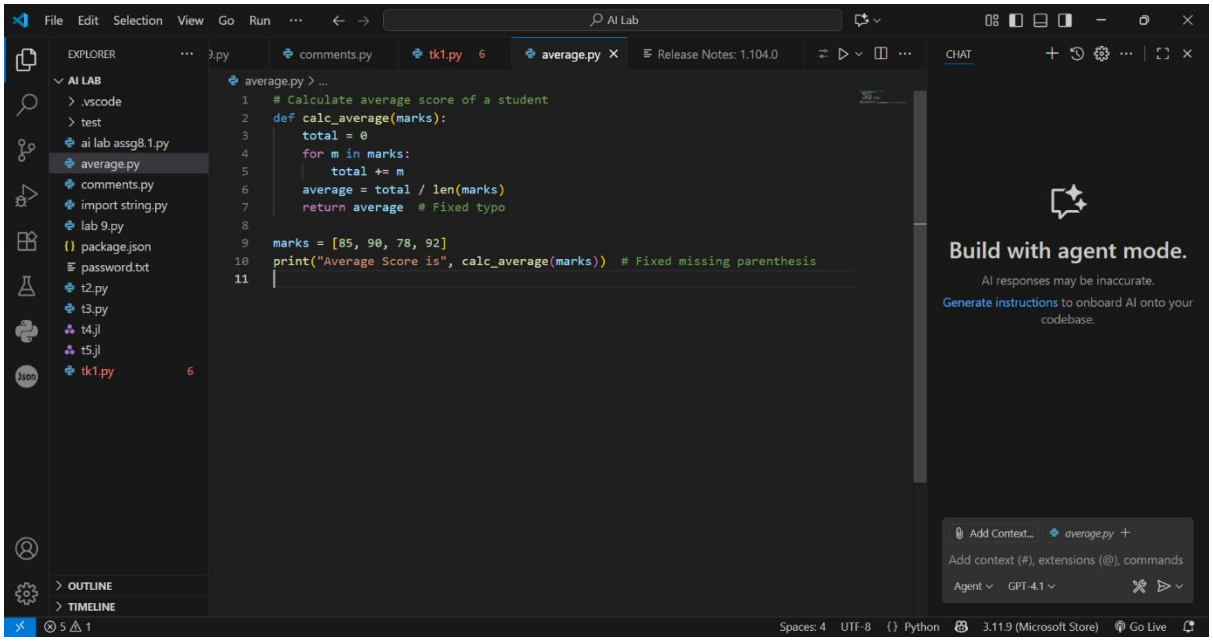
Expected Output:

• Corrected and runnable Python code with explanations of the fixes

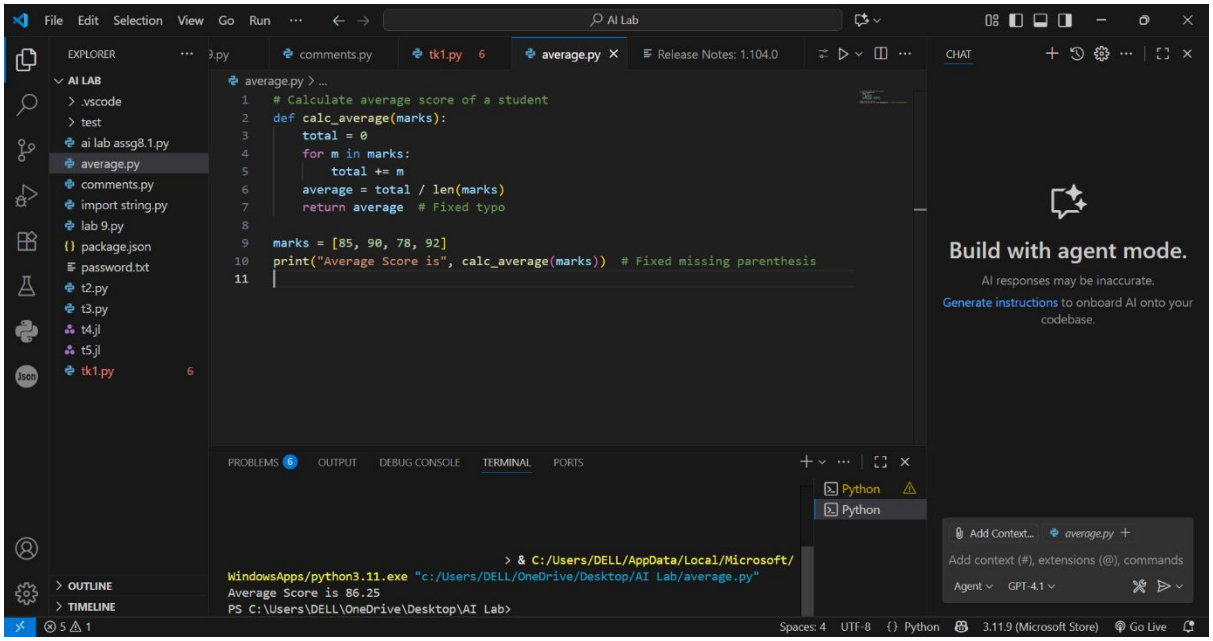
CODE :

```
# 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 average(marks))  # Fixed missing parenthesis
```



OUTPUT :



OBSERVATION : **Fixes Made**

1. **Indentation**:
   Python requires proper indentation inside functions. The contents of calc_average() were indented to be valid.
2. **Typo in return statement**:
   - Original: return avrage
   - Fixed: return average
3. **Missing parenthesis in print function**:
   - Original: print("Average Score is ", calc_average(marks)
   - Fixed: print("Average Score is", calc_average(marks))
4. **Optional improvement**:
   Removed the unnecessary space after "Average Score is" since print adds a space by default between arguments.

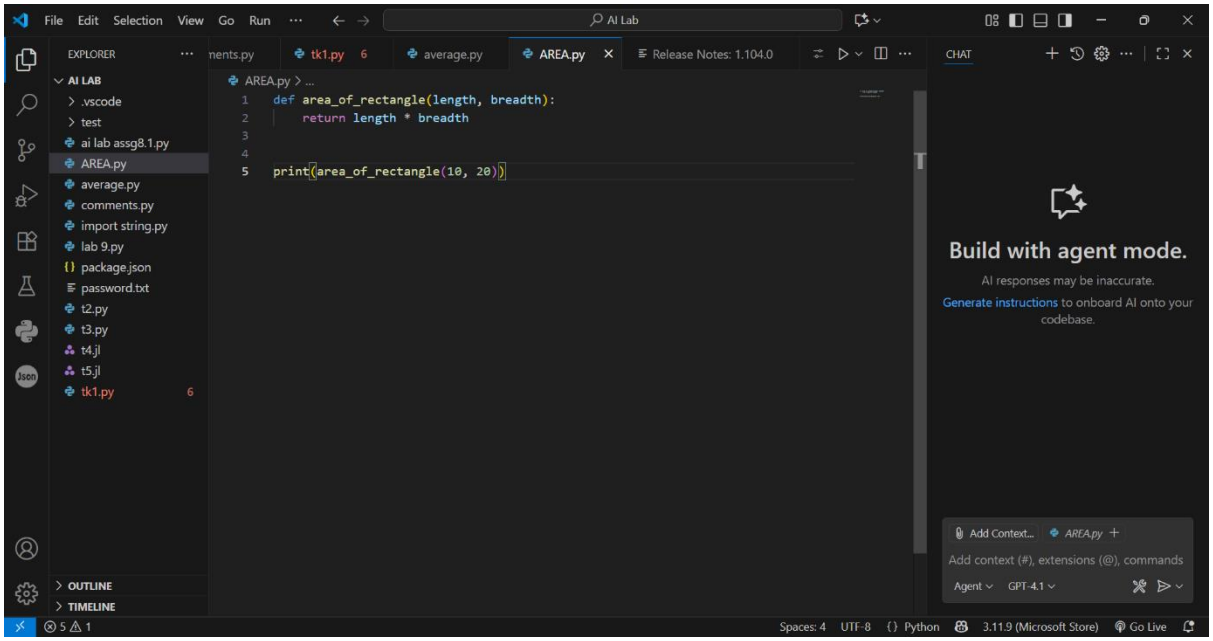
Task - 2: Use AI to refactor Python code to follow PEP 8 style guidelines.
Sample Input Code:
def area_of_rect(L,B):return L*B
print(area_of_rect(10,20))
Expected Output:
• Well-formatted PEP 8-compliant Python code

PROMPT : 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

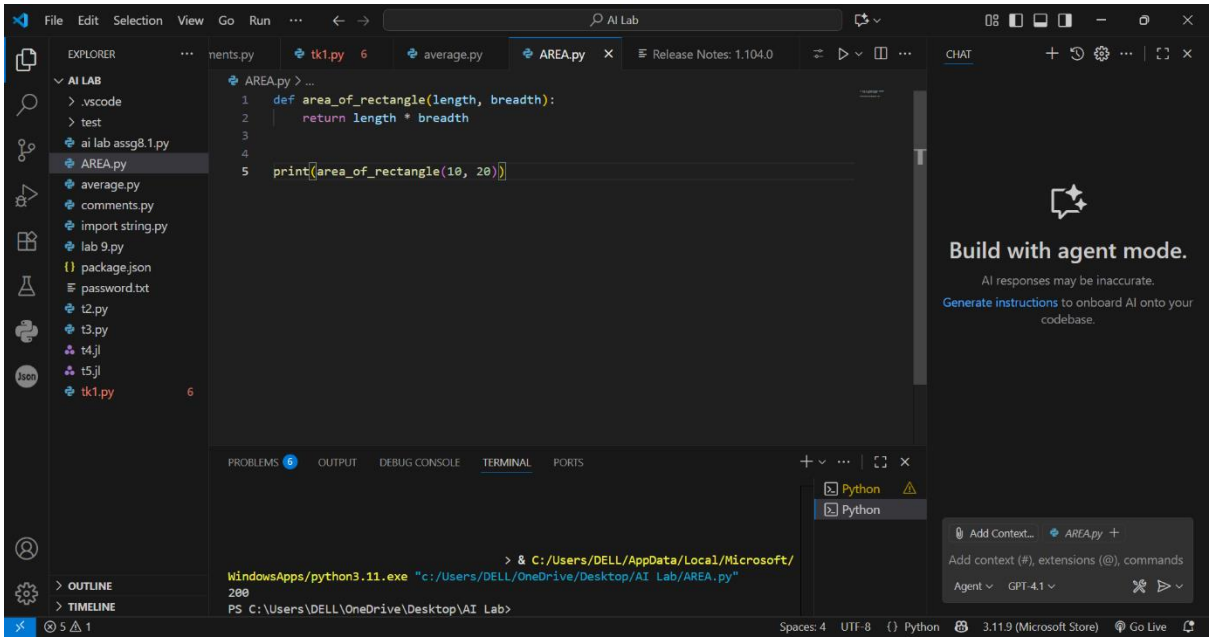
CODE : **def** area_of_rectangle(length, breadth):

  return length * breadth

print(area_of_rectangle(10, 20))



OUTPUT :



1. OBSERVATION : **Function Naming**: Changed area_of_rect → area_of_rectangle for readability and lowercase with underscores.
2. **Parameter Naming**: Changed L, B → length, breadth to avoid single-letter variables and improve clarity.
3. **Whitespace and Line Breaks**:
    - Moved return statement onto a separate indented line.
    - Added a blank line after the function for readability per PEP 8 guidelines.

Task -3: 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.
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_percentage(base_value, percentage):

"""Calculate the percentage of a given base value."""

return base_value * percentage / 100

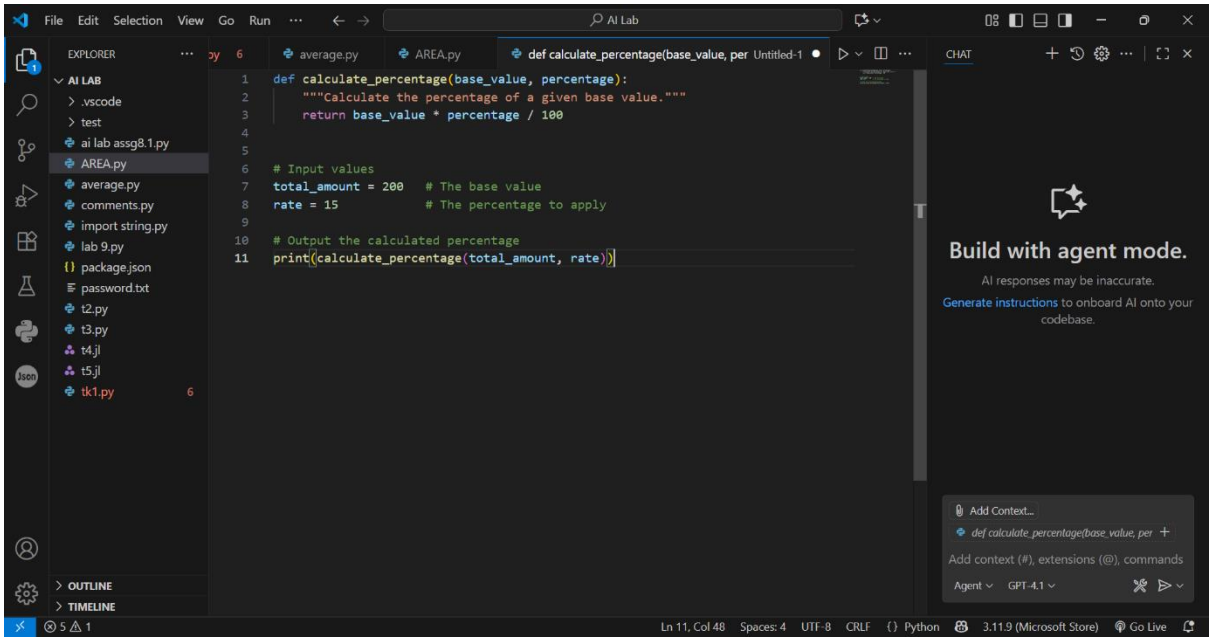
 # Input values

total_amount = 200 *# The base value*

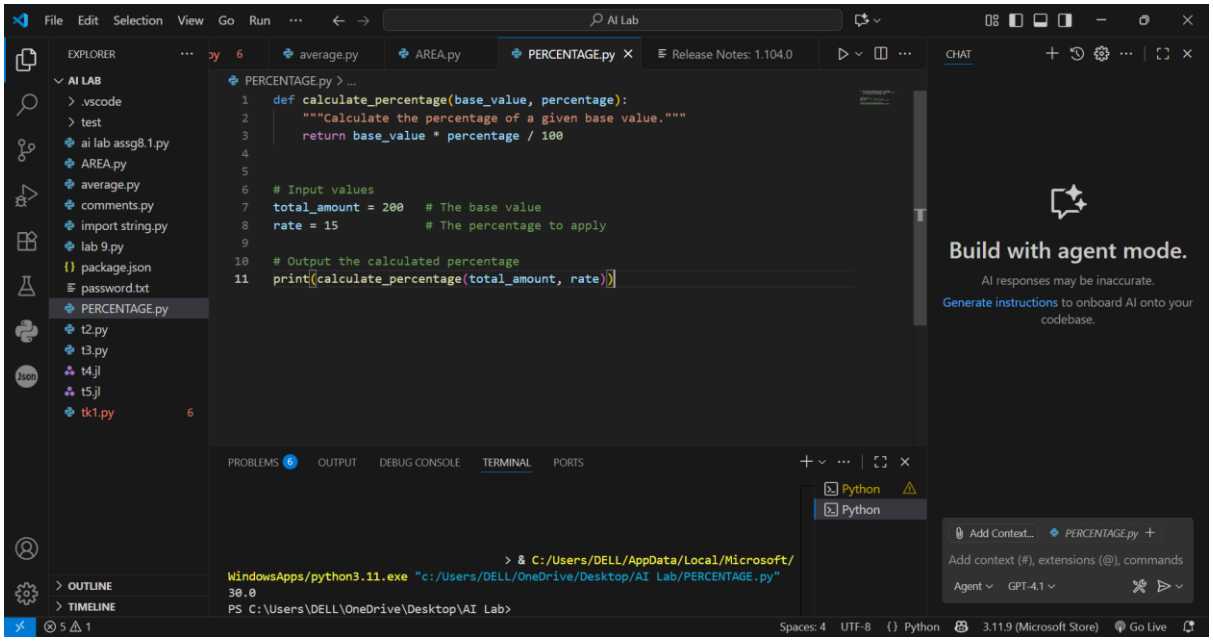
rate = 15 *# The percentage to apply*

 # Output the calculated percentage

print(calculate_percentage(total_amount, rate))



OUTPUT :



1. OBSERVATION : **Function Name**: Changed c → calculate_percentage to clearly describe its purpose.
2. **Parameter Names**: Changed x, y → base_value, percentage for clarity.
3. **Variable Names**: Changed a, b → total_amount, rate to make them meaningful.
4. **Docstring**: Added a clear description of what the function does.
5. **Inline Comments**: Explained the role of input variables and output.
6. **Formatting**: Added spacing, consistent indentation, and separation for readability.

Task -4: 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 : 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(name):

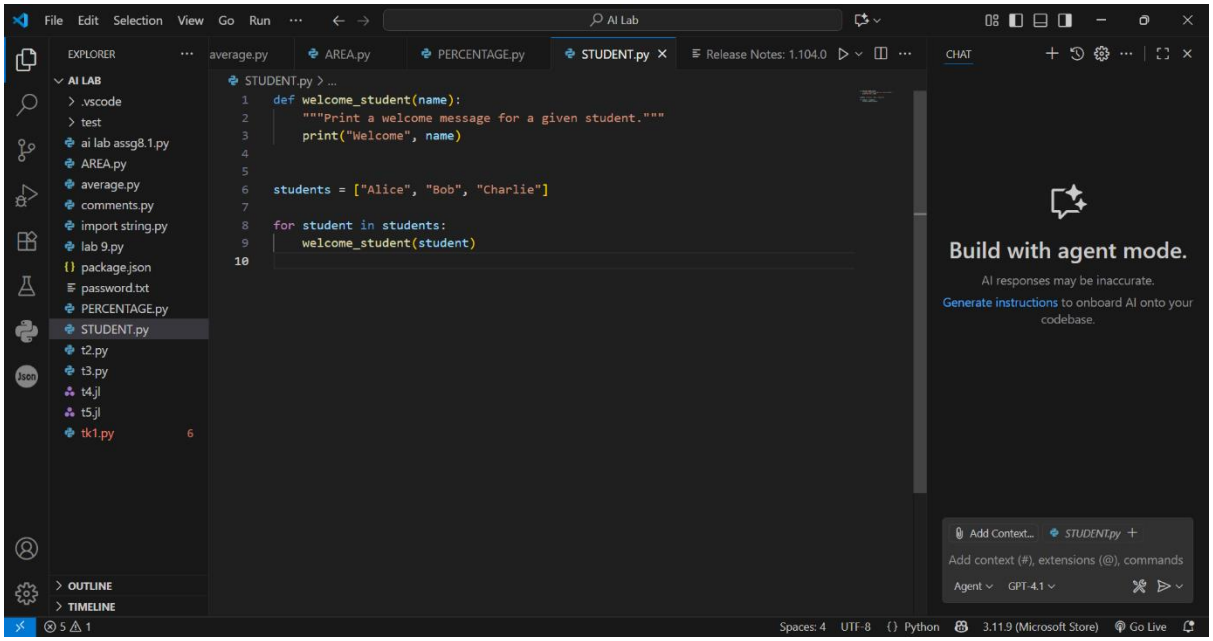
"""Print a welcome message for a given student."""

print("Welcome", name)

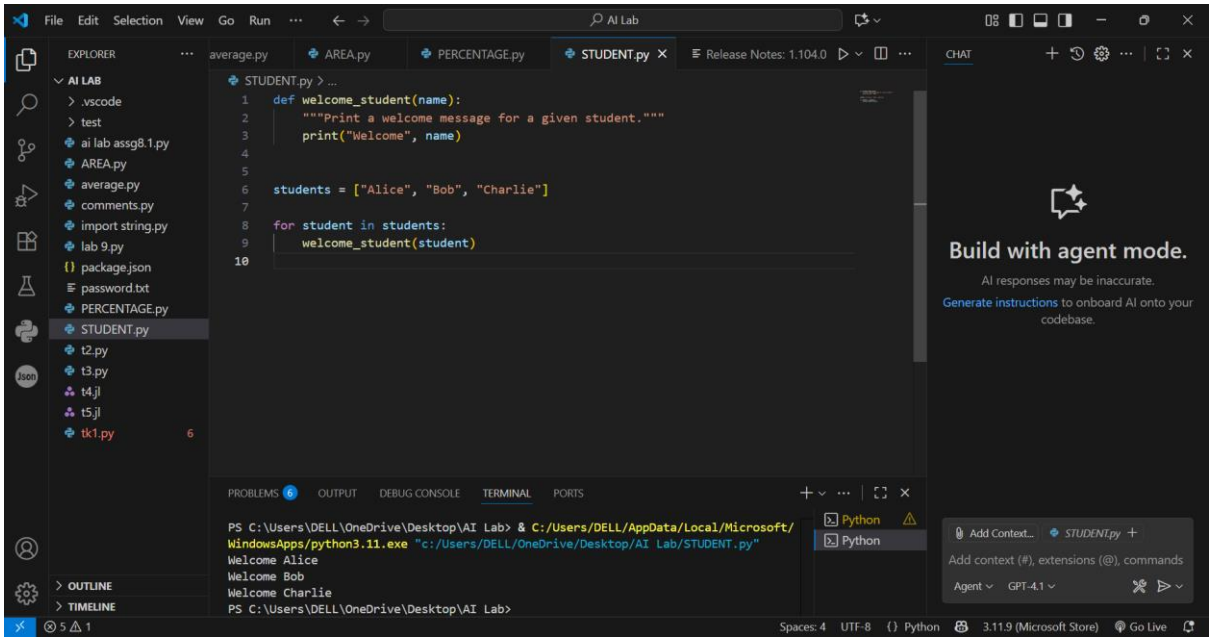
students = ["Alice", "Bob", "Charlie"]

for student **in** students:

welcome_student(student)



OUTPUT :



1. **OBSERVATION : Created a Function: Added welcome_student(name) to encapsulate the welcome logic.**
2. **Loop Instead of Repetition: Replaced multiple print statements with a for loop, making the code scalable for any number of students.**
3. **Docstring: Included a simple docstring for clarity.**
4. **Modularity: Logic is now reusable—function can be called with any student list.**

Task -5: 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.**

PROMPT : Make the code run faster.
Sample Input Code:

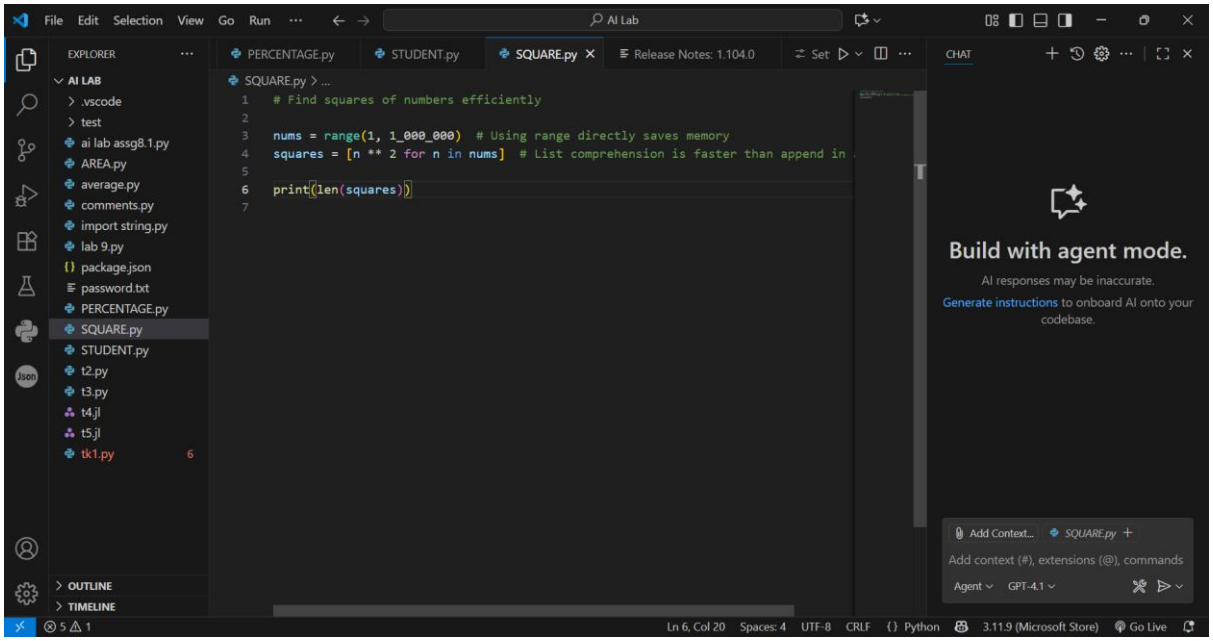
```
# Find squares of numbers
nums = [i for i in range(1,1000000)]
squares = []
for n in nums:
squares.append(n**2)
print(len(squares))
```

Expected Output:

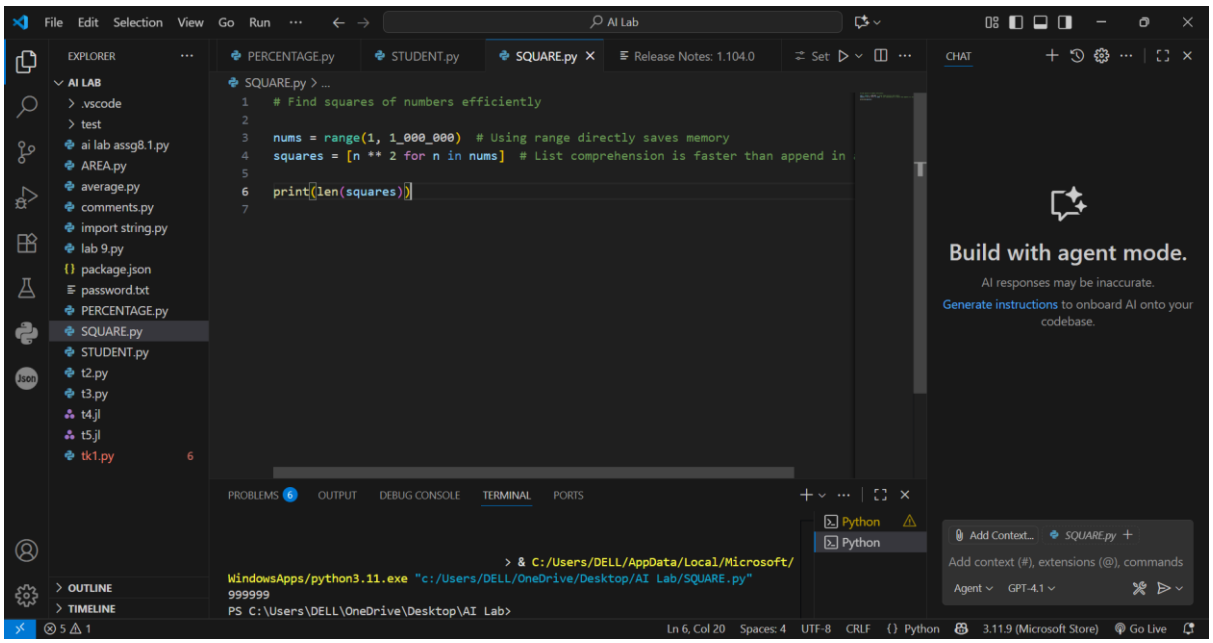
- **Optimized code using list comprehensions or vectorized operations.**

CODE :

```
# Find squares of numbers efficiently

nums = range(1, 1_000_000)  # Using range directly saves memory
squares = [n ** 2 for n in nums]  # List comprehension is faster than append in a loop

print(len(squares))
```



OUTPUT :



1. OBSERVATION : **Removed Redundant List Creation**:
    - Original used [i for i in range(...)], which materializes the list in memory immediately.
    - Replaced with range(...) which is lazy and more memory-efficient.
2. **List Comprehension**:

- Replaced the for loop with a list comprehension, which is implemented in C internally and runs faster than repeatedly calling append.

3. **Underscore in Large Number**:
    - 1_000_000 makes large numbers easier to read, while being functionally identic

Task -6: 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.

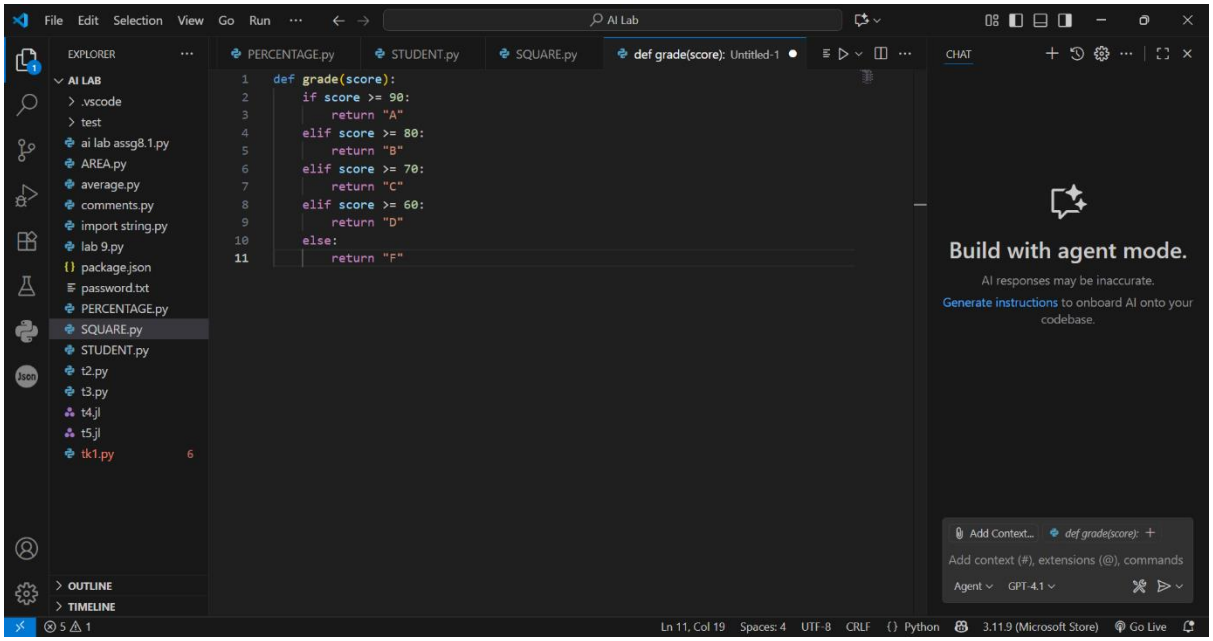
PROMPT : Simplify overly complex logic.
Sample Input Code:

```
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"
```

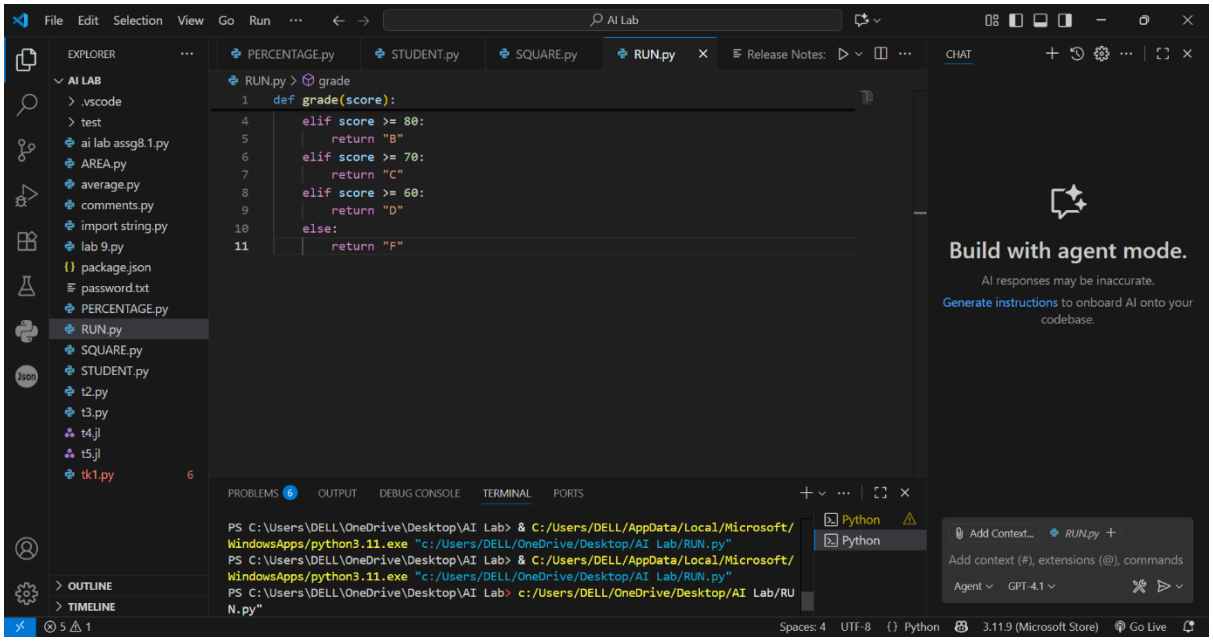
Expected Output:
• Cleaner logic using elif or dictionary mapping.

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"
```



OUTPUT :



1. OBSERVATION : **Used elif**: Replaced deeply nested if-else blocks with elif for better readability and maintainability.
2. **Cleaner Flow**: Each condition is checked in order without unnecessary nesting, making it easier to follow.