

AI ASSISTED CODING

ASSIGNMENT-10.1

NAME: K. RUCHITHA

HT NO:2303A52069

BATCH:32

Task Description #1 – Syntax and Logic Errors

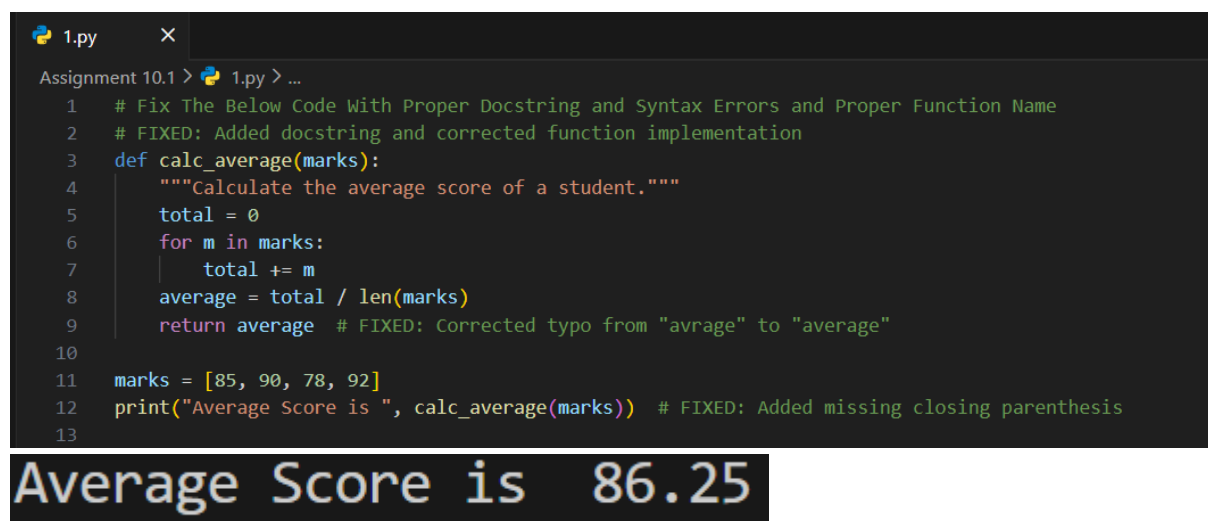
Task: Use AI to identify and fix syntax and logic errors in a faulty Python script.

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))
```

Expected Output:

- Corrected and runnable Python code with explanations of the fixes.



The screenshot shows a code editor window titled '1.py'. The code is as follows:

```
1 # Fix The Below Code With Proper Docstring and Syntax Errors and Proper Function Name
2 # FIXED: Added docstring and corrected function implementation
3 def calc_average(marks):
4     """Calculate the average score of a student."""
5     total = 0
6     for m in marks:
7         total += m
8     average = total / len(marks)
9     return average # FIXED: Corrected typo from "avrage" to "average"
10
11 marks = [85, 90, 78, 92]
12 print("Average Score is ", calc_average(marks)) # FIXED: Added missing closing parenthesis
13
```

Below the code, the output is displayed in a large, bold font: **Average Score is 86.25**

Task Description #2 – PEP 8 Compliance

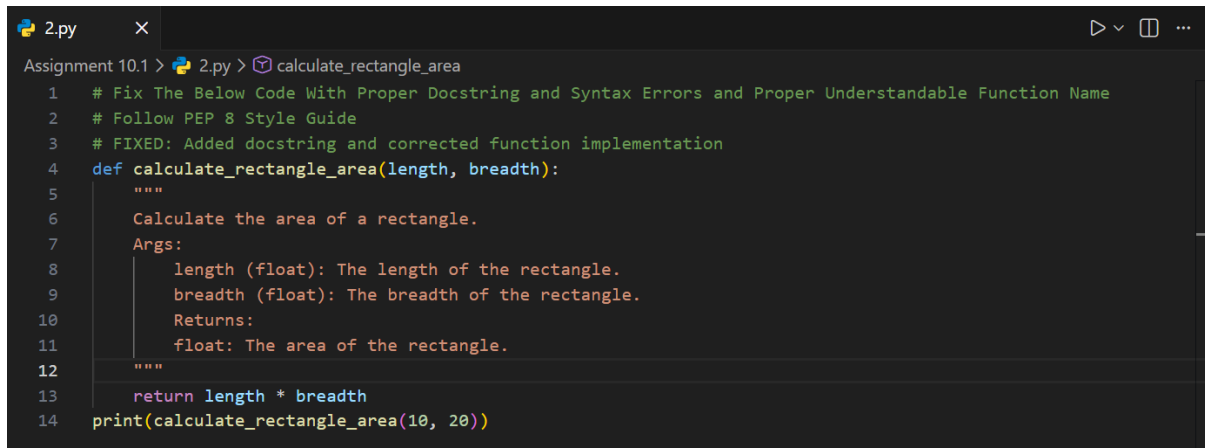
Task: Use AI to refactor Python code to follow PEP 8 style guidelines.

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.



```
2.py x  
Assignment 10.1 > 2.py > calculate_rectangle_area  
1 # Fix The Below Code With Proper Docstring and Syntax Errors and Proper Understandable Function Name  
2 # Follow PEP 8 Style Guide  
3 # FIXED: Added docstring and corrected function implementation  
4 def calculate_rectangle_area(length, breadth):  
5     """  
6     Calculate the area of a rectangle.  
7     Args:  
8         length (float): The length of the rectangle.  
9         breadth (float): The breadth of the rectangle.  
10    Returns:  
11        float: The area of the rectangle.  
12    """  
13    return length * breadth  
14 print(calculate_rectangle_area(10, 20))
```

200

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.

```
3.py X
Assignment 10.1 > 3.py > ...
1 # Refactor the code below make code more readable without changing its logic.
2
3 def calculate_percentage(part, whole):
4     """Calculate the percentage of a part relative to a whole.
5     Args:
6         part (float): The part value.
7         whole (float): The whole value."""
8     return (part * whole) / 100 # Calculates the percentage by multiplying the part with the whole and dividing by 100.
9
10 a=200
11 b=15
12 print(calculate_percentage(a,b))
```

30.0

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.

```
4.py
Assignment 10.1 > 4.py > ...
1 # Refactor the to code break repetitive or long code into reusable functions
2 students = ["Alice", "Bob", "Charlie"]
3
4 def welcome_student(student_name):
5     """Welcome a student by name."""
6     print("Welcome", student_name)
7
8 for student in students:
9     welcome_student(student)
```

```
Welcome Alice
Welcome Bob
Welcome Charlie
```

Task Description #5 – Performance Optimization

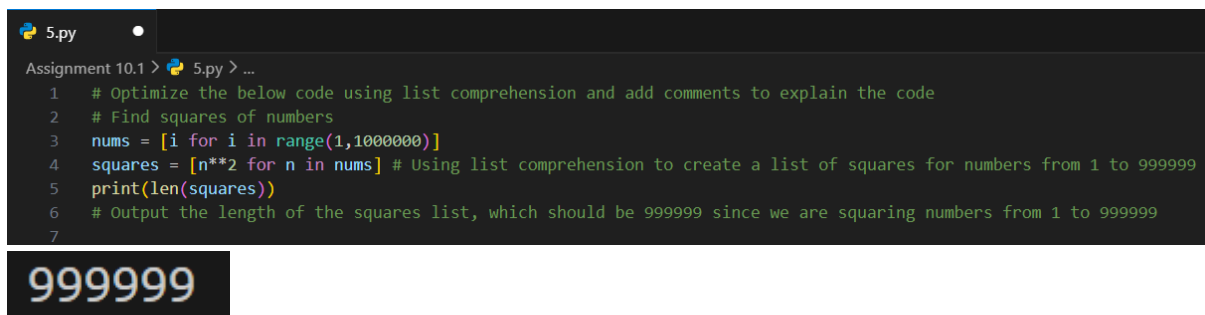
Task: Use AI to make the code run faster.

Sample Input Code:

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

Expected Output:

- Optimized code using list comprehensions or vectorized operations.

A screenshot of a code editor window titled '5.py'. The editor shows a Python script with line numbers 1 through 7. The code is as follows:

```
1 # Optimize the below code using list comprehension and add comments to explain the code
2 # Find squares of numbers
3 nums = [i for i in range(1,1000000)]
4 squares = [n**2 for n in nums] # Using list comprehension to create a list of squares for numbers from 1 to 999999
5 print(len(squares))
6 # Output the length of the squares list, which should be 999999 since we are squaring numbers from 1 to 999999
7
```

Below the code editor, the output '999999' is displayed in a large, bold, orange font.

Task Description #6 – Complexity Reduction

Task: Use AI to simplify overly complex logic.

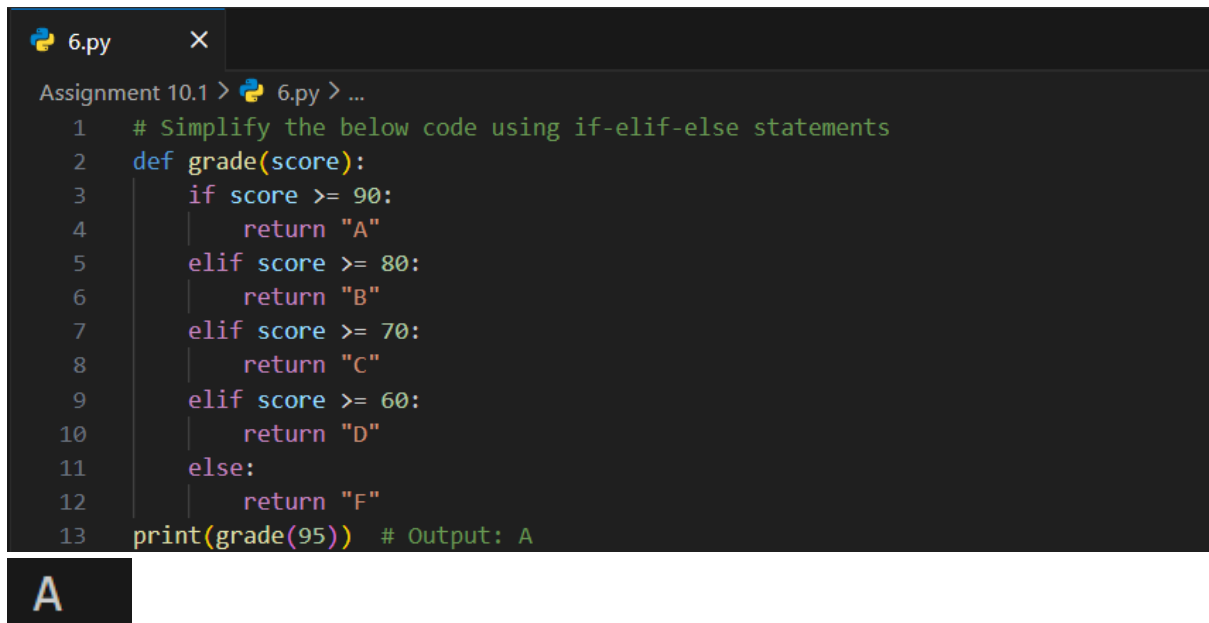
Sample Input Code:

```
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.



The screenshot shows a Python IDE window titled '6.py'. The code defines a function `grade(score)` that uses `if-elif-else` statements to return a grade based on a score. The function returns 'A' for scores 90 and above, 'B' for 80-89, 'C' for 70-79, 'D' for 60-69, and 'F' for scores below 60. The function is called with `grade(95)`, and the output 'A' is printed. The output is displayed in a separate box below the code editor.

```
Assignment 10.1 > 6.py > ...
1  # Simplify the below code using if-elif-else statements
2  def grade(score):
3      if score >= 90:
4          return "A"
5      elif score >= 80:
6          return "B"
7      elif score >= 70:
8          return "C"
9      elif score >= 60:
10         return "D"
11     else:
12         return "F"
13 print(grade(95)) # Output: A
```

A