

# ASSIGNMENT-10.1

Name: V.Sravani

Ht .No : 2303A51923

Batch: 23

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 with the file 10.1.py open. The code has been refactored to correct a type error and a missing parenthesis. The corrected code is as follows:

```
❶ 10.1.py > ...
❷ ❸ #refactored code with a typo and a missing parenthesis
❹ ❺ def calc_average(marks):
❻ ❼     total=0
➋ ❼     for m in marks:
⌃ ❼         total += m
⌂ ❼     average = total / len(marks)
⌃ ❼     return average # fixed typo: changed "avrage" to "average"
⌂ ❼ marks = [85, 90, 78, 92]
⌃ ❼ print("Average score is:",calc_average(marks))
```

Below the code editor is a terminal window showing the execution of the script and its output. The terminal output is:

```
PS C:\Users\SRAVANI\OneDrive\Dokument\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.11-64/python.exe "C:/Users/SRAVANI/OneDrive/Dokument/AI Assist Coding/10.1 (1).py"
Average score is: 86.25
PS C:\Users\SRAVANI\OneDrive\Dokument\AI Assist Coding>
```

## 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.

The screenshot shows a code editor interface with a dark theme. The code in the editor is as follows:

```
23-02-26 10.1 (2).py > area_of_rectangle  
1 def area_of_rectangle(l, B): return l*B  
2 print(area_of_rectangle(10, 20))  
3 #refactored the above code and add documentation and type hints.  
4 def area_of_rectangle(l: float, B: float) -> float:  
5     """  
6         calculate the area of a rectangle.  
7  
8         Parameters:  
9             l (float): The length of the rectangle.  
10            B (float): The breadth of the rectangle.  
11  
12        Returns:  
13            float: The area of the rectangle.  
14            raises: ValueError: If l or B is negative.  
15            TypeError: If l or B is not a number.  
16            """  
17  
18            if l < 0 or B < 0:  
19                raise ValueError("length and breadth must be non-negative.")  
20            if not isinstance(l, (int, float)) or not isinstance(B, (int, float)):  
21                raise TypeError("Length and breadth must be numbers.")  
22            return l * B  
23  
24 print(area_of_rectangle(10, 20))
```

Below the code editor, the terminal window shows the command and the output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂  
PS C:\users\SRAWANI\OneDrive\Documents\AI Assist Coding> & C:/users/SRAWANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/users/SRAWANI/OneDrive/Documents/AI Assist Coding/23-02-26 10.1 (2).py"  
200  
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.

The screenshot shows a code editor window with a Python file named '10.2(3).py'. The code has been refactored to use descriptive variable names and include docstrings. The code is as follows:

```
1  # 10.2(3).py > calculate_percentage
2  def c(x,y):
3      return x*y/100
4
5  a=200
6  b=15
7  print(c(a,b))
8  # refactor the above code with with descriptive variable names, inline comments, and clear formatting.
9  def calculate_percentage(part: float, whole: float) -> float:
10     """
11         calculate the percentage of a part relative to a whole.
12
13     Parameters:
14         part (float): The portion or part of the whole.
15         whole (float): The total or whole amount.
16
17     Returns:
18         float: The percentage value representing the part relative to the whole.
19         raises: ValueError: If whole is zero to avoid division by zero.
20         typeerror: If part or whole is not a number.
21
22     if whole == 0:
23         raise ValueError("Whole cannot be zero to avoid division by zero.")
24     if not isinstance(part, (int, float)) or not isinstance(whole, (int, float)):
25         raise TypeError("Part and whole must be numbers.")
26
27     percentage = (part / whole) * 100
28     return percentage
29
30
31
32
33
34
35
36
37
```

Below the code, the terminal output shows the execution of the script:

```
PS C:\Users\SRawanI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAWANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/users/SRAWANI/OneDrive/Dokumen\AI Assist Coding\10.2(3).py"
30.0
PS C:\Users\SRawanI\OneDrive\Dokumen\AI Assist Coding>
```

## 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.

```
* 10.1-4.py > welcome_student
1 students = ["Alice", "Bob", "Charlie"]
2 print("Welcome", students[0])
3 print("Welcome", students[1])
4 print("Welcome", students[2])
5 #refactor the above code to reduce redundancy with reusable function
6 def welcome_student(student_name: str) -> None:
7     """
8         Print a welcome message for a student.
9     """
10    Parameters:
11        student_name (str): the name of the student to welcome.
12    returns: None
13    raises: TypeError if student_name is not a string.
14    values:
15        student_name (str): the name of the student to welcome.
16    """
17    if not isinstance(student_name, str):
18        raise TypeError("student_name must be a string.")
19    print("Welcome", student_name)
20

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + - I S

PS C:\Users\SRawanI\OneDrive\Documents\AI Assist Coding> & C:/Users/SRAWANI/AppData/Local/Python/pythoncore-3.14-64/python.exe < c:/users/srawani/onedrive/documents/ai assist coding/10.1-4.py
Welcome Alice
Welcome Bob
Welcome Charlie
PS C:\Users\SRawanI\OneDrive\Documents\AI Assist Coding>
```

## 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.

```
◆ 10.1 (5).py > ...
1  nums = [i for i in range(1,1000000)]
2  squares = []
3  for n in nums:
4      squares.append(n**2)
5  print(len(squares))
6  #refactor the above code to reduce time complexity
7  nums = [i for i in range(1, 1000000)]
8  squares = [n**2 for n in nums]
9  print(len(squares))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14  
AT Assist Coding/10.1 (5).py"

999999  
999999

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>

```
◆ 10.1 (5).py > ...
1  import time
2
3  time1 = time.time()
4  nums = [i for i in range(1, 1000000)]
5  squares = []
6  for n in nums:
7      squares.append(n**2)
8  #print(len(squares))
9  time2 = time.time()
10 print("Time taken: ", time2 - time1)
11
12 # refactor the above code to reduce time complexity
13 time3 = time.time()
14 nums = [i for i in range(1, 1000000)]
15 squares = [n**2 for n in nums]
16 #print(len(squares))
17 time4 = time.time()
18 print("Time taken:", time4 - time3)
19
20 time5 = time.time()
21 #print(len([n**2 for n in range(1, 1000000)]))
22 time6 = time.time()
23 print("Time taken:", time6 - time5)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Py  
OneDrive/Dokumen/AI Assist Coding/10.1 (5).py"

Time taken: 0.1295607089996338  
Time taken: 0.11092209815979004  
Time taken: 2.384185791015625e-07

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>

## 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.

```
❶ 10.1.6.py >_
 1  def grade(score):
 2      if score >= 90:
 3          return "A"
 4      else:
 5          if score >= 80:
 6              return "B"
 7          else:
 8              if score >= 70:
 9                  return "C"
10              else:
11                  if score >= 60:
12                      return "D"
13                  else:
14                      return "F"
15 #refactor code to cleaner logic using elif or dictionary mapping
16 def grade(score: float) -> str:
17     """
18     return the score based on the score
19     parameters:
20     score (float): The score to evaluate.
21     returns:
22     str: grade(A, B, C, D, F) based on the score."""
23     if score >= 90:
24         return "A"
25     elif score >= 80:
26         return "B"
27     elif score >= 70:
28         return "C"
29     elif score >= 60:
30         return "D"
31     else:
32         return "F"
33 print(grade(85))
```

```
34 def grade(score: float) -> str:
35     """
36     return the grade based on score using dictionary mapping"""
37     grade_map = {
38         90: "A",
39         80: "B",
40         70: "C",
41         60: "D"
42     }
43     for threshold, grade in grade_map.items():
44         if score >= threshold:
45             return grade
46     return "F"
47 print(grade(95))
48
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PREFERENCES

```
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "C:/Users/SRAVANI/OneDrive/Dokumen/AI Assist Coding/10.1.6.py"
B
A
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>
```