

# ASSIGNMENT-10.1

Name:G.Charanya

HT NO:2303A510G6

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 avgare # 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.

```
C: > Users > souky > Downloads > ✎ average.py > ...
1  # refactored code with a typo and a missing parenthesis
2
3  def calc_average(marks):
4      total = 0
5      for m in marks:
6          total += m
7      average = total / len(marks)
8      return average # Fixed typo : 'avgare' to 'average' and added missing parenthesis
9  marks = [85, 90, 78, 92]
10 print("Average Score is ", calc_average(marks))
```

```
● PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/calculatorexample.py
Average Score is 86.25
○ PS C:\Users\2303a\OneDrive\Desktop\AI>
```

## 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 window with multiple tabs at the top: 'read\_file.py', 'findmax.py', 'login.py', 'calculator.py', 'conversion.py', 'course.py', and 'rect.py'. The 'rect.py' tab is active, displaying the following Python code:

```
C: > Users > souky > Downloads > rect.py > ...  
1 def area_of_rect(l,b) : return l*b  
2 print(area_of_rect(10,20))  
3 #refactored the above code and add documentation and type hints  
4 def area_of_rect(length: float, breadth: float) -> float:  
5     """  
6         Calculate the area of a rectangle given its length and breadth.  
7     """  
8     Parameters:  
9     length (float): The length of the rectangle.  
10    breadth (float): The breadth of the rectangle.  
11  
12    Returns:  
13    float: The area of the rectangle calculated as length multiplied by breadth.  
14    Raises:  
15    ValueError: If length or breadth is negative, as dimensions cannot be negative.  
16    TypeError: If length or breadth is not a number (int or float).  
17    """  
18    if not isinstance(length, (int, float)) or not isinstance(breadth, (int, float)):  
19        raise TypeError("Length and breadth must be numbers (int or float).")  
20    if length < 0 or breadth < 0:  
21        raise ValueError("Length and breadth must be non-negative.")  
22    return length * breadth  
23 print(area_of_rect(10, 20))
```

A status bar at the bottom of the code editor shows the average score as 80.25 and the command line as PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/calculatorexample.py 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.

```
> Users > souky > OneDrive > Documents > AI Lab > 📄 read_file.py > 🗂️ calculate_percentage
1 def c(x,y):
2     return x*y/100
3 a=200
4 b=15
5 print(c(a,b))
6 #refactored the above code with descriptive variable names, inline comments, and clear formatting
7 def calculate_percentage(part: float, whole: float) -> float:
8     """
9         Calculate the percentage of a part relative to a whole.
10
11     Parameters:
12         part (float): The portion or part value.
13         whole (float): The total or whole value.
14
15     Returns:
16         float: The percentage calculated as (part / whole) * 100.
17     Raises:
18         ValueError: If the whole is zero, as division by zero is not allowed.
19         TypeError: If part or whole is not a number (int or float).
20     """
21     if not isinstance(part, (int, float)) or not isinstance(whole, (int, float)):
22         raise TypeError("Both part and whole must be numbers (int or float).")
23     if whole == 0:
24         raise ValueError("Whole cannot be zero to avoid division by zero.")
25     return (part / whole) * 100
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/calculatorexample.py
1333.333333333335
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

## 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.

```
C: > Users > souky > OneDrive > Documents > AI Lab > student.py > welcome_student
1 students = ["Alice", "Bob", "Charlie"]
2 print("Welcome", students[0])
3 print("Welcome", students[1])
4 print("Welcome", students[2])
5 #refactored code to reduce redundancy with reusable function
6 def welcome_student(student: str) -> None:
7     """
8         Print a welcome message for a student.
9
10    Parameters:
11        student (str): The name of the student to welcome.
12
13    Returns:
14        None
15    values:
16        student: A string representing the name of the student.
17        type error: If the input is not a string, a TypeError will be raised.
18
19    """
20    if not isinstance(student, str):
21        raise TypeError("Student name must be a string.")
22    print("Welcome", student)
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/calculatorex
● Welcome Alice
● Welcome Bob
● Welcome Charlie
○ PS C:\Users\2303a\OneDrive\Desktop\AI>
```

## 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.

```
pu read_file.py ● pu findmax.py ● pu login.py ● pu calculator.py ● pu conversion.py
C: > Users > souky > Downloads > pu square.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  #refactored the above code to reduce time complexity
7  nums = [i for i in range(1,1000000)]
8  squares = [n**2 for n in nums]
Q 9  print(len(squares))
```

```
● PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/calculatorexample.py
999999
999999
○ PS C:\Users\2303a\OneDrive\Desktop\AI>
```

```
rd_mc.py ● pu findmax.py ● pu login.py ● pu calculator.py ● pu conversion.py
C: > Users > souky > Downloads > pu square.py > ...
1  import time
2  time1 = time.time()
3  nums = [i for i in range(1,1000000)]
4  squares = []
5  for n in nums:
6      squares.append(n**2)
7  #print(len(squares))
8  time2 = time.time()
9  print("Time taken:", time2 - time1)
10 # refactor the above code to reduce time complexity
11 time3 = time.time()
12 nums = [i for i in range(1,1000000)]
13 squares = [n**2 for n in nums]
14 #print(len(squares))
15 time4=time.time()
16 print("Time taken:", time4 - time3)
17 time5 = time.time()
18 #print(len([n**2 for n in range(1,1000000)]))
19 time6 = time.time()
20 print("Time taken:", time6 - time5)
```

```
● PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/calculatorexample.py
Time taken: 0.1389319896697998
Time taken: 0.15295076370239258
Time taken: 1.430511474609375e-06
○ PS C:\Users\2303a\OneDrive\Desktop\AI>
```

## 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.

```
read_file.py student.py grade.py X findmax.py login.py calculator.py conversion.py co
C: > Users > souky > OneDrive > Documents > AI Lab > grade.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 #refactored code to Cleaner logic using elif or dictionary mapping.
16 def grade(score: int) -> str:
17     """
18     Return the grade based on the score.
19     Parameters:
20     score (int): Student score
21     Returns:
22     str: Grade (A, B, C, D, or F)
23     """
24     if score >= 90:
25         return "A"
26     elif score >= 80:
27         return "B"
28     elif score >= 70:
29         return "C"
30     elif score >= 60:
31         return "D"
32     else:
33         return "F"
34 print (grade(95))
35 def grade(score: int) -> str:
36     """
37     Return the grade based on the score using dictionary mapping.
38     """
39     grade_map = {
40         90: "A",
41         80: "B",
42         70: "C",
43         60: "D",
44         0: "F"
45     }
46     for cutoff, letter in grade_map.items():
47         if score >= cutoff:
48             return letter
49 print (grade(85))
50
```

```
● PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/calculatorexample.py
A
B
○ PS C:\Users\2303a\OneDrive\Desktop\AI>
```