

Lab Assignment-10.1

Name:Anand

Hallticket:2303A51090

Batch-02

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:

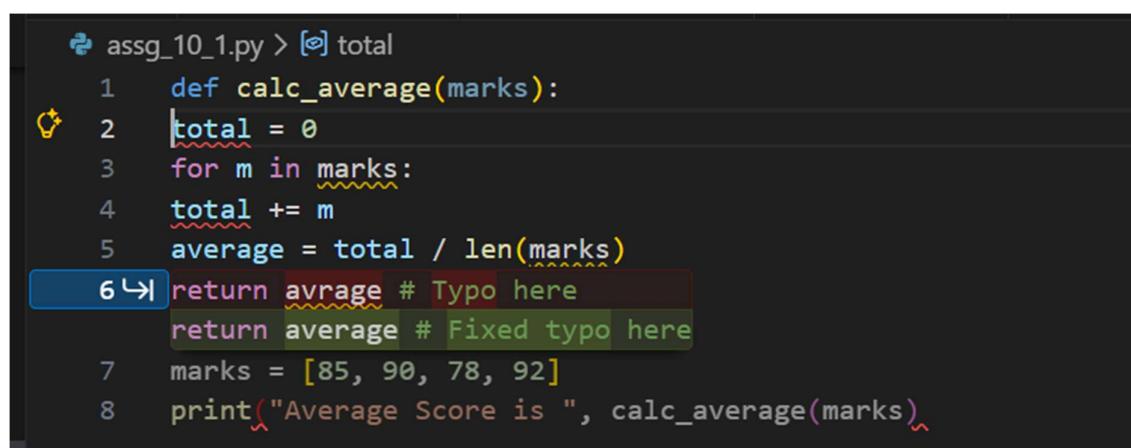
```
# Calculate average score of a student
```

```
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.

Privous code:



```
assg_10_1.py > [?] total
 1  def calc_average(marks):
 2  ↗ total = 0
 3  for m in marks:
 4  ↗ total += m
 5  average = total / len(marks)
 6 ↗ return avgare # Typo here
 7  ↗ return average # Fixed typo here
 8  marks = [85, 90, 78, 92]
 9  print("Average Score is ", calc_average(marks))
```

Corrected code:

```

🐍 assg_10_1.py > ...
1  def calc_average(marks):
2      total = 0
3      for m in marks:
4          total += m
5      average = total / len(marks)
6      return average # Fixed typo here
7  marks = [85, 90, 78, 92]
8  print("Average Score is ", calc_average(marks))

```

```

server stopped
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
Average Score is 86.25
PS C:\Users\arell\Music\aiac>

```

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.

Code:

```

9  #refactor the below code to include error handling and input validation
10 def area_of_rect(L,B) :
    if not isinstance(L, (int, float)) or not isinstance(B, (int, float)):
        raise TypeError("Length and breadth must be numbers.")
    if L < 0 or B < 0:
        raise ValueError("Length and breadth must be non-negative.")
11    return L*B
12 print(area_of_rect(10,20))
13

```

```

8     print("Average score is ", calc_average(marks))
9     #refactor the below code to include error handling and input validation
10    def area_of_rect(L,B) :
11        if not isinstance(L, (int, float)) or not isinstance(B, (int, float)):
12            raise TypeError("Length and breadth must be numbers.")
13        if L < 0 or B < 0:
14            raise ValueError("Length and breadth must be non-negative.")
15        return L*B
16    print(area_of_rect(10,20))
17

```

Output:

```

PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
200
PS C:\Users\arell\Music\aiac>

```

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.

Code:

```

18     #refacto the below code discriptive variable names and incline comments to each line and clear formatting
19
20    def c(x,y):
21        return x*y/100
22    a=200
23    b=15
24    print(c(a,b))
25
26    def calculate_percentage(value, total):
27        """Calculate percentage of a value relative to a total."""
28        if total == 0:
29            raise ValueError("Total cannot be zero.")
30        return (value / total) * 100
31
32    a = 200
33    b = 15
34    print(calculate_percentage(a, b))

```

```
18 #refactor the below code descriptive variable names and incline comments to each line and clear formatting
19
20 def calculate_percentage(value, total):
21     """Calculate percentage of a value relative to a total."""
22     if total == 0:
23         raise ValueError("Total cannot be zero.")
24     return (value / total) * 100
25
26 a = 200
27 b = 15
28 print(calculate_percentage(a, b))
29
30 |
```

Output:

```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
1333.333333333335
PS C:\Users\arell\Music\aiac> []
```

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.

```
30 #refactor the below code with modular code with reusable functions and error handling
31 def add_student(students, student_name):
32     students = ["Alice", "Bob", "Charlie"]
33     print("Welcome", students[0])
34     print("Welcome", students[1])
35     print("Welcome", students[2])
36
```

```

30     #refactor the below code with modular code with reusable functions and error handling
31
32     def validate_name(name):
33         """Validate that name is a non-empty string."""
34         if not isinstance(name, str) or not name.strip():
35             raise ValueError("Student name must be a non-empty string.")
36         return name.strip()
37
38     def welcome_student(student_name):
39         """Welcome a student by name."""
40         try:
41             validated_name = validate_name(student_name)
42             print(f"Welcome {validated_name}")
43         except ValueError as e:
44             print(f"Error: {e}")
45
46     def process_students(students):
47         """Process and welcome a list of students."""
48         if not isinstance(students, list):
49             raise TypeError("Students must be a list.")
50         for student in students:
51             welcome_student(student)
52
53     # Main execution
54     students = ["Alice", "Bob", "Charlie"]
55     process_students(students)
56

```

Output:

```

PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
Welcome Alice
Welcome Bob
Welcome Charlie
PS C:\Users\arell\Music\aiac> []

```

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.

Code:

```
30  #refactor the below code and optimize the code with the list comprehension and include error handling and input validation
31  nums = [i for i in range(1,1000000)]
32 →| squares = []
33  for n in nums:
34  |   squares.append(n**2)
35  print(len(squares))
36
```

```
30  # Refactor with list comprehension, error handling, and input validation
31  def calculate_squares(start, end):
32      """Calculate squares of numbers in a range with validation."""
33      if not isinstance(start, int) or not isinstance(end, int):
34          raise TypeError("Start and end must be integers.")
35      if start < 0 or end < 0:
36          raise ValueError("Start and end must be non-negative.")
37      if start > end:
38          raise ValueError("Start must be less than or equal to end.")
39
40      # Use generator expression for memory efficiency
41      squares = [n**2 for n in range(start, end)]
42      return squares
43
44  try:
45      result = calculate_squares(1, 1000000)
46      print(len(result))
47  except (TypeError, ValueError) as e:
48      print(f"Error: {e}")
49
50
51
```

Output:

```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
999999
PS C:\Users\arell\Music\aiac> []
```

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 "A"  
    else:  
        if score >= 70:  
            return "B"  
        else:  
            if score >= 60:  
                return "C"  
            else:  
                return "D"  
        else:  
            return "F"
```

Expected Output:

- Cleaner logic using elif or dictionary mapping.

Code:

```
49 #refactor the below code and cleaner the logic using elif or dictionary mapping
50 |
51 def grade(score):
52 ↳ if score >= 90:    if score >= 90:
53     return "A"          return "A"
54     else:                elif score >= 80:
55         if score >= 80:        return "B"
56         return "B"          elif score >= 70:
57         else:                  return "C"
58         if score >= 70:          elif score >= 60:
59         return "C"          return "D"
60         else:                  else:
61         if score >= 60:          return "F"
62         return "D"
63         else:
64         return "F"
65
66
67
68
```

```
49  : Refactor with dictionary mapping for cleaner logic
50  ef grade(score):
51      if not isinstance(score, (int, float)):
52          raise TypeError("Score must be a number.")
53      if score < 0 or score > 100:
54          raise ValueError("Score must be between 0 and 100.")
55
56      # Dictionary mapping for grade ranges
57      grade_map = {
58          90: "A",
59          80: "B",
60          70: "C",
61          60: "D",
62          0: "F"
63      }
64
65      # Return grade based on score thresholds
66      for threshold in sorted(grade_map.keys(), reverse=True):
67          if score >= threshold:
68              return grade_map[threshold]
69
70  try:
71      print(grade(85))
72  except (TypeError, ValueError) as e:
73      print(f"Error: {e}")
74
```

Output:

```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_10_1.py"
B
PS C:\Users\arell\Music\aiac> []
```