AI Assisted Coding

Assignment 10.1

Name: G. Samshray

Roll no:2303A51670

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

Code:

```
# Corrected 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 here

marks = [85, 90, 78, 92]

print("Average Score is ", calc_average(marks))  # Added missing parenthesis
```

# Explanation of Fixes:

# 1. The original code had a typo in the return statement where "average" was misspelled as "avrage". This has been corrected to "average".

# 2. The print statement was missing a closing parenthesis at the end. This has been added to ensure the code runs without syntax errors.

# 3. The indentation of the code inside the function was not correct in the original snippet. Proper indentation has been applied to ensure the code is syntactically correct and runs as expected.

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:

```
# Refactored Code:

def area_of_rectangle(length, breadth):

    """Calculate the area of a rectangle."""

    return length * breadth

  print(area_of_rectangle(10, 20))
```

# Explanation of Refactoring:

# 1. The function name has been changed from "area_of_rect" to "area_of_rectangle" to be more descriptive and follow the PEP 8 naming convention for functions.

# 2. The parameter names have been changed from "L" and "B" to "length" and "breadth" to improve readability and follow PEP 8 guidelines for variable names.

# 3. A docstring has been added to the function to explain its purpose, which is a recommended practice in PEP 8 for improving code readability and maintainability.

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:

```
# refactored Code:
def calculate_percentage(value, percentage):
    """Calculate the percentage of a given value."""
    return value * percentage / 100
amount = 200
percent = 15
# Calculate and print the percentage of the amount
print(calculate_percentage(amount, percent))
```

# Explanation of Readability Enhancements:

# 1. The function name has been changed from "c" to "calculate_percentage" to be more descriptive of its purpose.

# 2. The parameter names have been changed from "x" and "y" to "value" and "percentage" to improve readability and make it clear what the function expects as input.

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:

```
# Refactored Code:

def welcome_students(student_list):

    """Print a welcome message for each student in the list."""

    for student in student_list:

        print("Welcome", student)

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

welcome_students(students)
```

# Explanation of Refactoring for Maintainability:

# 1. A new function named "welcome_students" has been created to encapsulate the logic for welcoming students. This promotes code reusability and makes the code more modular.

# 2. The function takes a list of students as an argument and iterates through it to print a welcome message for each student, eliminating the need for repetitive print statements. This makes the code easier to maintain and extend in the future if needed.

• Modular code with reusable functions.

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:

```
nums = [i for i in range(1, 1000000)]
squares = [n**2 for n in nums]
print(len(squares))
# Optimized code using vectorized operations with NumPy:
import numpy as np
nums = np.arange(1, 1000000)
squares = nums ** 2
print(len(squares))
```

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.

Code:

```
def grade(score):
    """
    Determines the letter grade based on the given score.
    Args:
        score (int): The numerical score to be graded.
    Returns:
        str: The letter grade corresponding to the score.
    """
    if score >= 90:
        return "A"
    elif score >= 80:
        return "B"
    elif score >= 70:
        return "C"
    elif score >= 60:
        return "D"
    else:
        return "F"
# Example usage
score_input = int(input("Enter the score to grade: "))
grade_result = grade(score_input)
print(f"The letter grade for a score of {score_input} is: {grade_result}")
```