AI ASSISTANT CODING ASSIGNMENT-7.2

1	**Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs**
	
	Lab Objectives:
	
	• To identify and correct syntax, logic, and runtime errors in Python programs using AI tools.
	• To understand common programming bugs and AI-assisted debugging suggestions.
	• To evaluate how AI explains, detects, and fixes different types of coding errors.
	• To build confidence in using AI to perform structured debugging practices.
	
	Lab Outcomes (LOs):
	After completing this lab, students will be able to:
	
	• Use AI tools to detect and correct syntax, logic, and runtime errors.
	• Interpret AI-suggested bug fixes and explanations.
	• Apply systematic debugging strategies supported by AI-generated insights.
	• Refactor buggy code using responsible and reliable programming patterns.
	
	Task Description
	
	Task 1 – Runtime Error Due to Invalid Input Type
	
	• A Python program accepts user input and performs arithmetic operations. However, the program throws a runtime error because the input is treated as a string instead of a numeric type.
	
	Example (Buggy Code):
	
	`num = input("Enter a number: ")`
	`result = num + 10`
	`print(result)`
	
	• **Task:**
		Week4 Tuesday

Use AI tools to identify the cause of the runtime error and modify the program so it executes correctly.

Expected Output -1:

- AI converts the input to the appropriate numeric type and eliminates the runtime error.

Task Description

Task 2 – Incorrect Function Return Value

A function is designed to calculate the square of a number, but it does not return the computed result properly.

Example (Buggy Code):

```
def square(n):
  result = n * n
```

Task:
Use AI assistance to analyze the function and ensure the correct value is returned.

Expected Output -2:

AI fixes the missing return statement and the function returns the correct output.

Task Description

Task 3 – IndexError in List Traversal

A Python program iterates over a list using incorrect index limits, causing an IndexError.

Example (Buggy Code):

```
numbers = [10, 20, 30]
for i in range(0, len(numbers)+1):
  print(numbers[i])
```

Task:
Use AI to identify the incorrect loop boundary and correct the iteration logic.

Expected Output -3:

AI fixes the loop condition and prevents out-of-range list access.

Task Description

Task 4 – Uninitialized Variable Usage

A program uses a variable in a calculation before assigning it any value.

Example (Buggy Code):

```
if True:
    pass
print(total)
```

Task:
Use AI tools to detect the uninitialized variable and correct the program.

Expected Output -4:

AI initializes the variable correctly before it is used

Task Description

Task 5 – Logical Error in Student Grading System

A grading program assigns incorrect grades due to improper conditional logic.

Example (Buggy Code):

```
marks = 85
if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "C"
else:
    grade = "B"
print(grade)
```

Task:
Use AI to analyze the grading conditions and correct the logical flow.

Expected Output -5:

AI corrects the conditional logic so grades are assigned accurately.

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

Task 1 : Runtime Error Due to Invalid Input Type

Prompt :

Identify the errors and fix them

Fixed Code:

```
num_str = input("Enter a number: ")
num = int(num_str) # Convert the input string to an integer
result = num + 10
print(result)
```

Output:

```
Enter a number: 5
15
```

Code Explanation:

1. Identification: The error was a TypeError caused by mismatched data types (String + Integer).

2. Detection: AI tools identify that input() defaults to string output.

3. Correction: Wrapped the input in int() to ensure the variable num holds a numeric value.

4. Refinement: I added a try-except block. This is a ‘responsible programming pattern’ (as mentioned in your LOs) that prevents the program from crashing if a user accidentally types "hello" instead of a number.

Task 2 : Incorrect Function Return Value

Prompt:

Identify the errors and fix them

Fixed Code:

```
def square(n):
    result = n * n
    return result  # This sends the value back out

# Testing the function
value = 5
output = square(value)
print(f"The square of {value} is {output}")
```

Expected output:

```
The square of 5 is 25
```

Code Explanation:

1. **Identification**: The function performed a calculation but lacked an exit path for the data.
2. **Detection:** AI identifies that the function completes its execution without a return statement, leading to a NoneType result in the main program.
3. **Correction:** Added return result to pass the computed value back to the caller.
4. **Refactoring:** While result = n * n followed by return result is correct, you can refactor this for brevity to return n ** 2 or return n * n directly.

Task 3 – IndexError in List Traversal

Task:

Use AI to identify the incorrect loop boundary and correct the iteration logic.

Prompt:

Identify the errors and fix them

Fixed code:

```
numbers = [10, 20, 30]

# Fix: Remove the +1. range(3) provides indices 0, 1, and 2.
for i in range(len(numbers)):
    print(numbers[i])
```

Expected Output:

```
10
20
30
```

Code Explanation:

- Identification: The error was an IndexError, specifically ‘list index out of range’.
- Detection: AI identifies that len(numbers) + 1 creates a boundary that exceeds the list's maximum index (*length - 1*).
- Correction: Adjusted the range parameters to stop at the actual length of the list.
- Refactoring: Recommended direct iteration as a more robust pattern to avoid manual index management.

Task 4 : Uninitialized Variable Usage

Prompt:
Identify the errors and fix them

Fixed Code:

```
# Initialization: Giving the variable a starting value
total = 0

if True:
    # Now we can safely modify or use the variable
    total = total + 50

print(total)
```

Expected Output:

```
50
```

Code Explanation:

- **Identification:** The error is a NameError: name 'total' is not defined.
- **Detection:** AI tools scan the code for variable references and check if they have a preceding assignment statement in the accessible scope.
- **Correction:** Defined total = 0 at the top of the script to ensure the variable exists in memory before the print() function or any logic tries to access it.
- **Refactoring:** It is a best practice to initialize variables at the top of a function or script to make the code more readable and prevent these ‘missing name’ errors.

Task 5 : Logical Error in Student Grading System

Prompt:

Identify the errors and fix them

Fixed Code:

```
marks = 85

if marks >= 90:
    grade = "A"
elif marks >= 80:
    grade = "B"  # Corrected from 'C' to 'B'
elif marks >= 70:
    grade = "C"  # Added a proper threshold for 'C'
else:
    grade = "F"  # The 'else' should be the failing or lowest grade

print(f"Marks: {marks}, Grade: {grade}")
```

Expected Output:

```
Marks: 85, Grade: B
```

Code Explanation:

- **Identification:** This was a Logical Flaw where the code syntax was correct, but the output did not match the intended real-world rules.
- **Detection:** AI compares the mapping of values (80+ usually equals B) against the assigned strings in the code. It also notes that the else logic was mathematically backward.
- **Correction:** Reordered the grade assignments so they follow a logical descending scale (A \rightarrow B \rightarrow C \rightarrow F).
- **Refactoring:** Using an elif chain is the most efficient way to handle this, as Python stops checking conditions as soon as it finds the first ‘True’ one.