# LAB Assignment - 7

Sanjana Bayya

2303A52019

Batch - 38

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

Week4 -

Wednesday

• 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 for 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 using AI-generated insights

• Refactor buggy code using reliable programming patterns

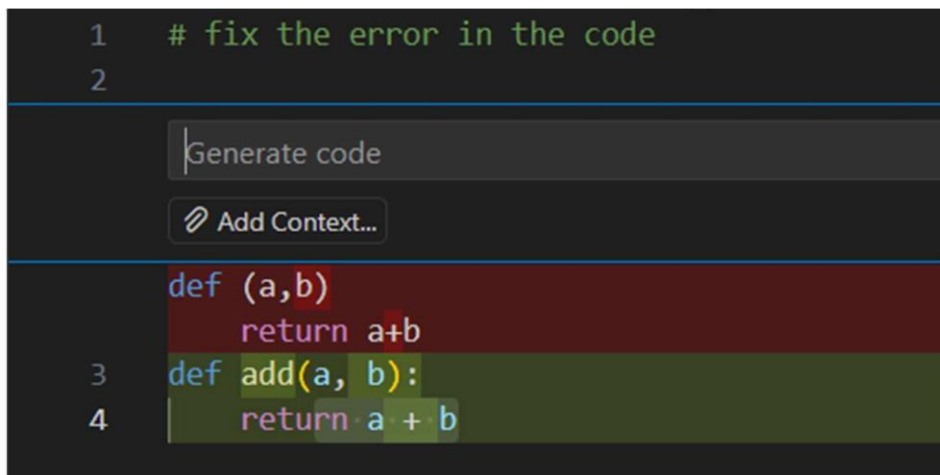Task 1: Fixing Syntax Errors

Scenario

You are reviewing a Python program where a basic function definition contains a syntax error.

Requirements

• Provide a Python function add(a, b) with a missing colon

• Use an AI tool to detect the syntax error

• Allow AI to correct the function definition

• Observe how AI explains the syntax issue

Expected Output

• Corrected function with proper syntax

• Syntax error resolved successfully

• AI-generated explanation of the fix

```
1    # fix the error in the code
2

    Generate code

    ⌀ Add Context...

def (a,b)
    return a+b
3   def add(a, b):
4   |   return a + b
```

**Explanation:**
The colon : was missing after the function definition. Adding it fixes the syntax error.


Task 2: Debugging Logic Errors in Loops

Scenario

You are debugging a loop that runs infinitely due to a logical mistake.

Requirements

• Provide a loop with an increment or decrement error

- Use AI to identify the cause of infinite iteration

- Let AI fix the loop logic

- Analyze the corrected loop behavior

Expected Output

- Infinite loop issue resolved

- Correct increment/decrement logic applied

- AI explanation of the logic error

```
1    # debug logic errors in code
2
3    def count_down(n):
4        while n>0:
5            print(n)
6            n+=1
7            ↳ -
8
```

```
1    # debug logic errors in code
2
3    def count_down(n):
4        while n>0:
5            print(n)
6            n-=1
7
8
```

**Explanation:**
The loop was running infinitely because n was increasing (n += 1) instead of decreasing.
Changing it to n -= 1 makes n move toward 0, so the loop ends correctly.

Task 3: Handling Runtime Errors (Division by Zero)

Scenario

A Python function crashes during execution due to a division by zero error.

Requirements

• Provide a function that performs division without validation

• Use AI to identify the runtime error

• Let AI add try-except blocks for safe execution

• Review AI's error-handling approach

Expected Output

• Function executes safely without crashing

• Division by zero handled using try-except

• Clear AI-generated explanation of runtime error handling

```
1    # Handle runtime errors for below code
2
3    def divide(a,b):
4 →|     return a/b              try:
5    print(divide(10,0))             return a/b
                                 except ZeroDivisionError:
                                     return "Error: Division by zero is not allowed."
```

```
1    # Handle runtime errors for below code
2
3    def divide(a,b):
4        try:
5            return a/b
6        except ZeroDivisionError:
7            return "Error: Division by zero is not allowed."
8
9    print(divide(10,0))
```

**Explanation:**
The runtime error occurs when dividing by zero.
Using a try-except block catches the ZeroDivisionError and prevents the
program from crashing.

Task 4: Debugging Class Definition Errors

Scenario

You are given a faulty Python class where the constructor is incorrectly defined.

Requirements

• Provide a class definition with missing self-parameter

• Use AI to identify the issue in the __init__() method

• Allow AI to correct the class definition

• Understand why self is required

Expected Output

• Corrected __init__() method

• Proper use of self in class definition

• AI explanation of object-oriented error

```
1    # debug class Definition errors
2    class Rectangle:
3        def __init__(self, length, width):
4            self.length=length
5            self.width=width
6
```

**Explanation:**
The constructor __init__ was missing the self parameter.
self refers to the current object and is required to store and access instance variables inside a class.

Task 5: Resolving Index Errors in Lists

Scenario

A program crashes when accessing an invalid index in a list.

Requirements

- Provide code that accesses an out-of-range list index

- Use AI to identify the Index Error

- Let AI suggest safe access methods

- Apply bounds checking or exception handling

Expected Output

- Index error resolved

- Safe list access logic implemented

- AI suggestion using length checks or exception handling

```
1    # Resolve the index error in list from the below code
2    numbers=[1,2,3]
3 →  print(numbers[5])
       ↳ try:
```

```
1    # Resolve the index error in list from the below code
2    numbers=[1,2,3]
3    try:
4        print(numbers[5])
5    except IndexError:
6        print("Index is out of bounds")
7
8
```

**Explanation:**
The error occurs because index 5 does not exist in the list.
Using try-except catches the Index Error and prevents the program from crashing.