ASSIGNMENT-7.3

(Error Debugging with AI: Systematic approaches to finding and fixing bugs)

HTNO:2303A51905

BATCH NO:28

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

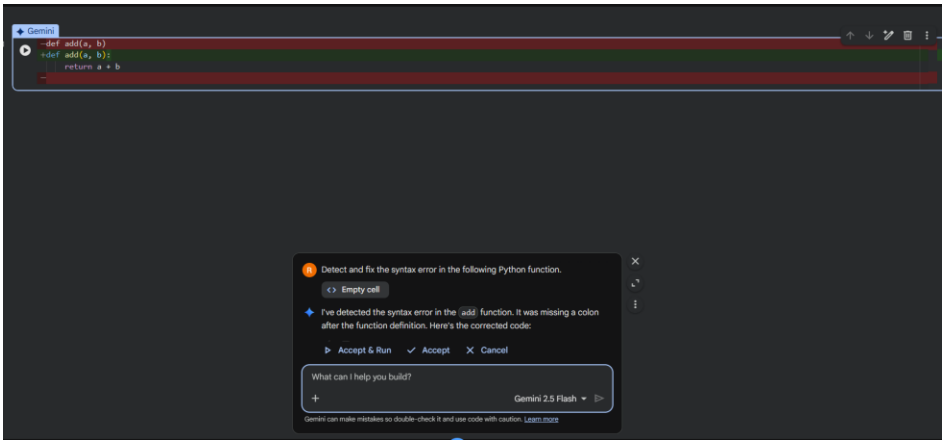
Prompt:

"Detect and fix the syntax error in the following Python function"

Buggy Code

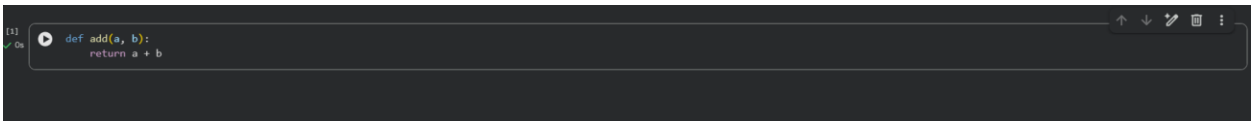
```
def add(a, b)
    return a + b
```

AI-Detected Issue



- Missing colon (:) at the end of the function definition.

Corrected code:



Explanation

- In Python, a colon is mandatory after function definitions.
- Without the colon, Python raises a SyntaxError.
- Adding the colon fixes the syntax issue and allows the function to execute properly.

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

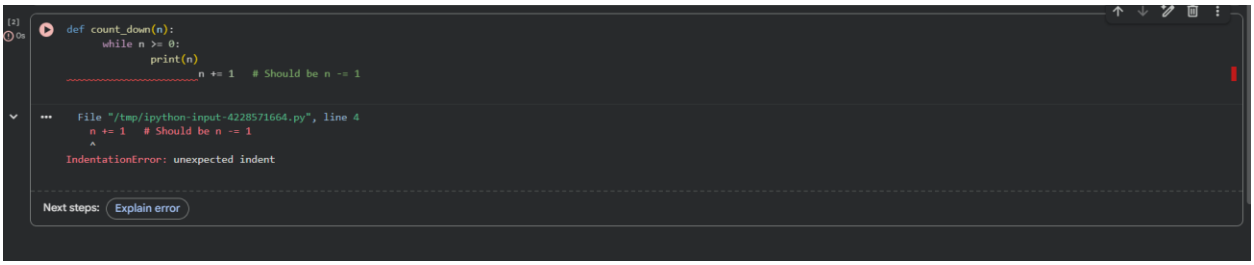
Prompt:

"Identify and fix the logical error causing an infinite loop."

Buggy Code

```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1   # Should be n -= 1
```

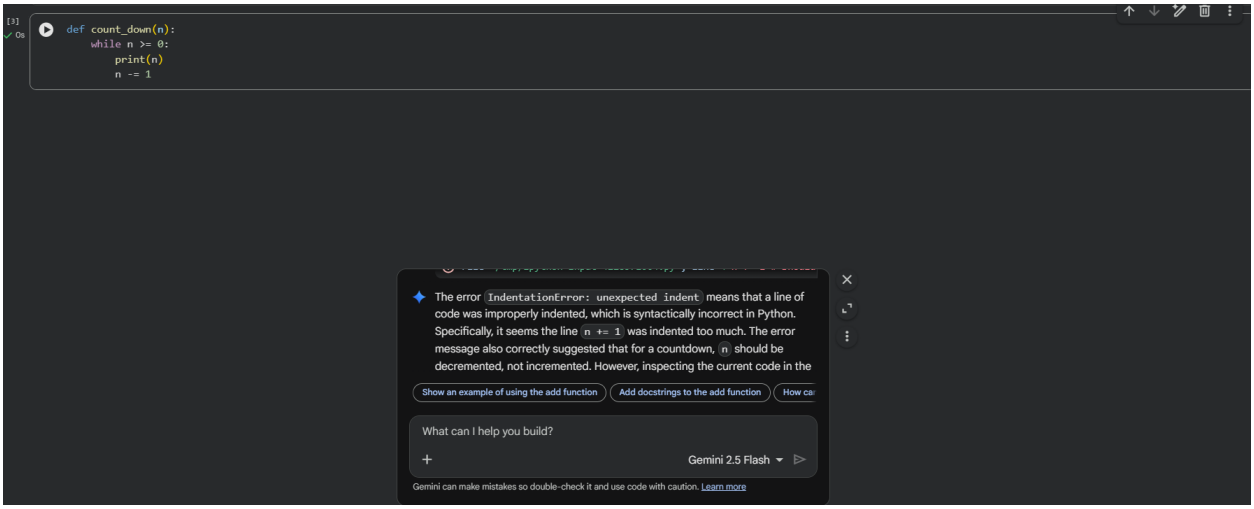
AI-Detected Issue



- IndentationError: unexpected indent

- The loop control variable `n` is **incremented instead of decremented**.
- Since the condition is `n >= 0`, increasing `n` makes the condition always true.
- This results in an **infinite loop**.

Corrected Code:



Explanation:

- The loop is intended to count down from n to 0.
- However, incrementing n prevents the loop from reaching the terminating condition.
- AI identified that decrementing n allows the loop condition to eventually become false.
- This correction stops the infinite execution.

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

Prompt:

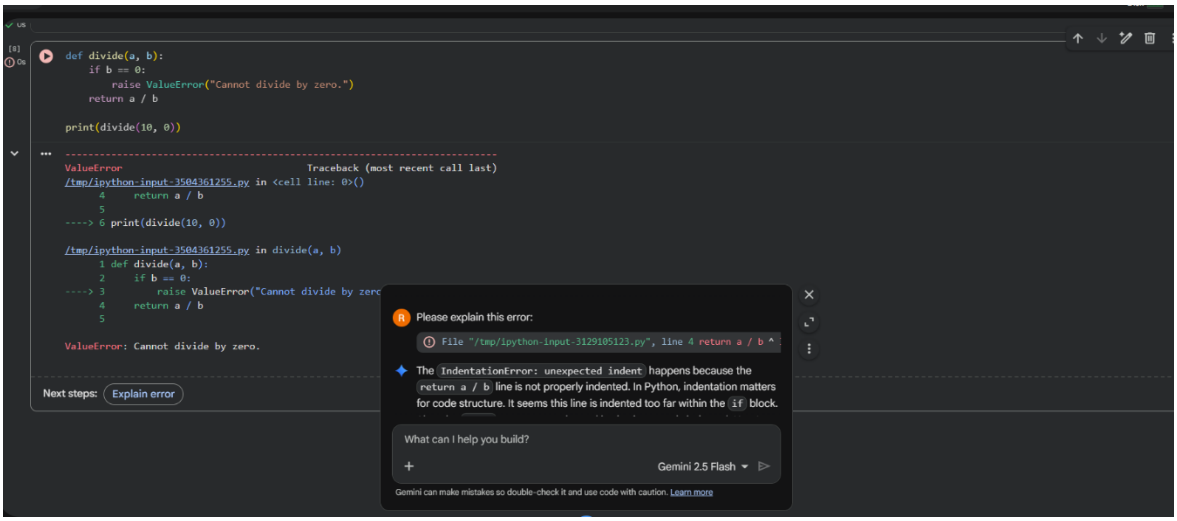
"Identify the runtime error and modify the code to prevent program crash"

Buggy Code

```
def divide(a, b):
    return a / b

print(divide(10, 0))
```

AI-Detected Issue



- Division by zero causes a ZeroDivisionError.

Corrected code:

```
def divide(a, b):
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b

print(divide(10, 2))
```

```
5.0
```

Explanation:

- AI identified that the runtime crash occurs due to division by zero.
- It added a **validation check** before performing division.
- Raising a ValueError prevents unsafe execution and clearly informs the user.
- This approach ensures safer and more predictable program behavior.

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

Prompt:

“Identify the issue in the class constructor and correct it”

Buggy Code

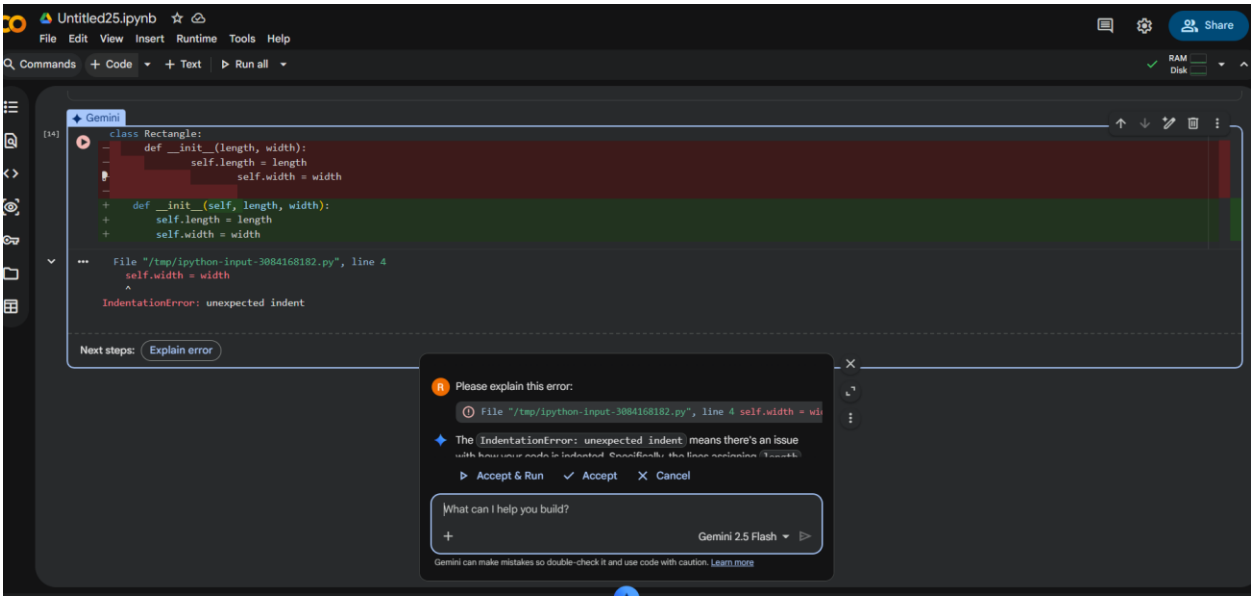
```
class Student:

    def __init__(name, roll):

        name = name

        roll = roll
```

AI-Detected Issue



- Missing self parameter in the constructor.
- Instance variables are not properly assigned.

Corrected Code:

Explanation:

- self is mandatory in instance methods to access object data.
- AI identified that length and width must be associated with the object.
- Adding self allows proper initialization of instance variables.

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

Prompt:

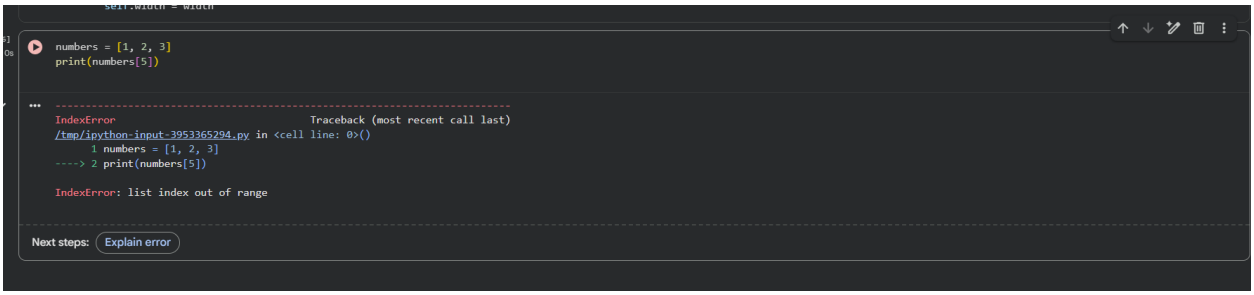
“Identify the index error and suggest a safe access method.”

Buggy Code

numbers = [1, 2, 3]

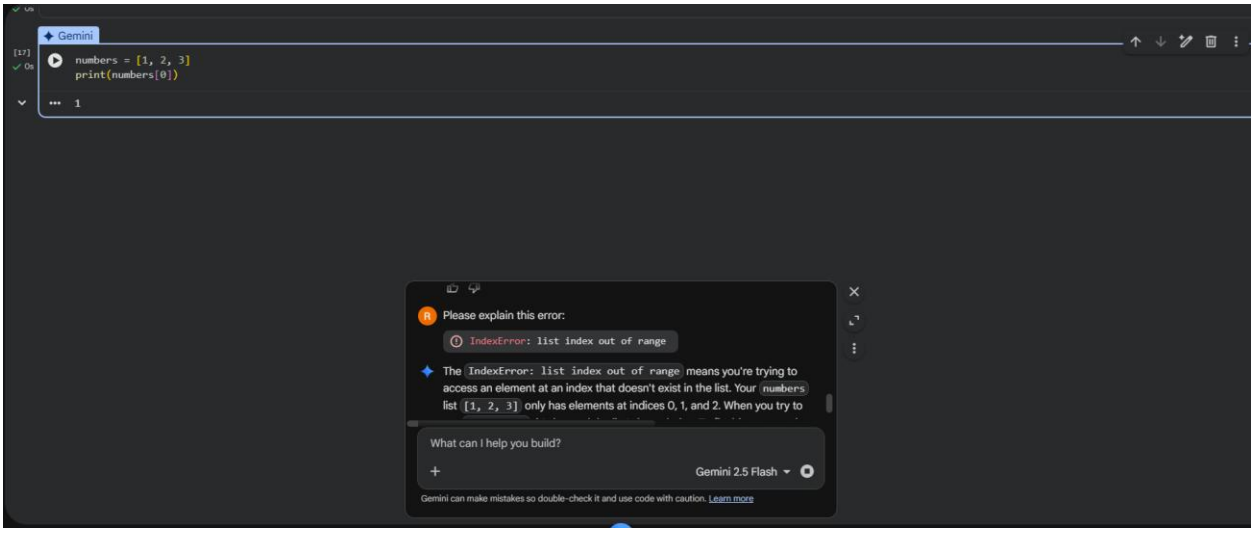
print(numbers[5])

AI-Detected Issue



- Index 5 does not exist in the list.
- Causes IndexError.

Corrected code:



Explanation:

- List indices must be within the range of list length.
- Checking bounds prevents runtime errors.