

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

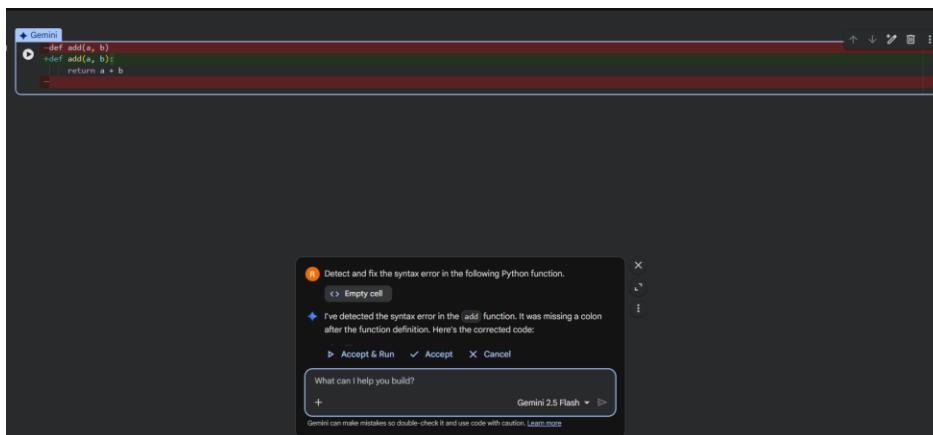
Prompt:

“Detect and fix the syntax error in the following Python function”

Buggy Code

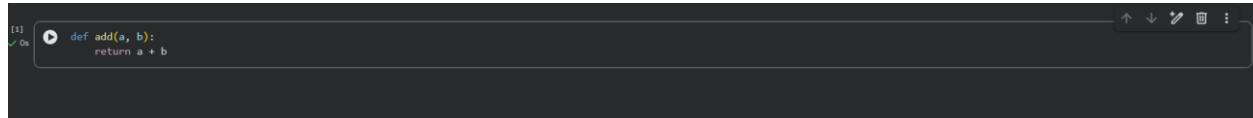
```
def add(a, b)
    return a + b
```

AI-Detected Issue



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

Corrected code:



A screenshot of a code editor window. The code in the editor is:

```
[1] 0s def add(a, b):  
    return a + b
```

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

The screenshot shows a code editor window with a dark theme. In the code area, there is a Python script named 'ipython-input-4228571664.py'. The code contains a function definition:`[2] 0s def count_down(n):
 while n >= 0:
 print(n)
 n += 1 # Should be n -= 1

...
File "/tmp/ipython-input-4228571664.py", line 4
 n += 1 # Should be n -= 1
 ^
IndentationError: unexpected indent`

Below the code, a button labeled 'Next steps: Explain error' is visible.

- **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:

The screenshot shows the same code editor window after the error was fixed. The code now correctly decrements `n`:`[3] ✓ 0s def count_down(n):
 while n >= 0:
 print(n)
 n -= 1`

A modal window titled 'The error' provides an explanation of the `IndentationError: unexpected indent` and suggests that for a countdown, `n` should be decremented, not incremented. It also notes that AI identified the correction.

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

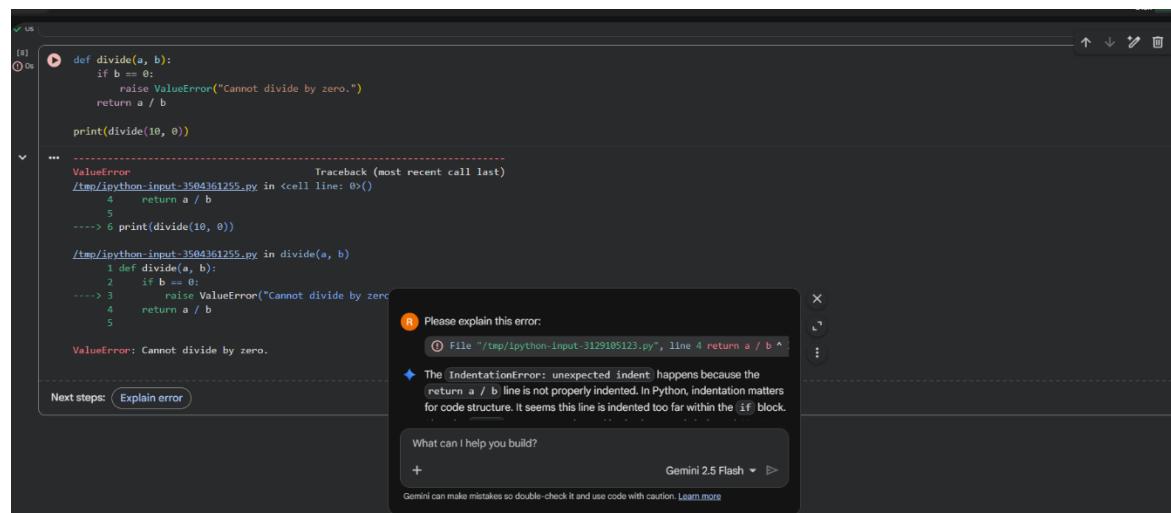
Prompt:

“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



The screenshot shows a code editor with a Python script. The script defines a function `divide` that returns the division of `a` by `b`. It then calls this function with arguments `(10, 0)`, which causes a `ValueError: Cannot divide by zero.` exception. An AI tooltip is overlaid on the code, providing an explanation of the error and suggesting a fix. The tooltip text is as follows:

Please explain this error:
File "/tmp/ipython-input-3504361255.py", line 4 return a / b :
The IndentationError: unexpected indent happens because the
return a / b line is not properly indented. In Python, indentation matters
for code structure. It seems this line is indented too far within the if block.

- 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

The screenshot shows a code editor window with a dark theme. In the center, there is a modal dialog from 'Gemini' titled 'Please explain this error'. The modal contains the following text:

```
File "/tmp/ipython-input-3084168182.py", line 4
    self.width = width
^
IndentationError: unexpected indent
```

Below this, a note says: 'The IndentationError: unexpected indent means there's an issue with how your code is indented. Normally, there has to be one tab or four spaces.'

At the bottom of the modal, there are buttons: 'Accept & Run', 'Accept', and 'Cancel'. A small note at the bottom right says: 'Gemini can make mistakes so double-check it and use code with caution. Learn more'.

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

Corrected Code:

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

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

```
numbers = [1, 2, 3]
print(numbers[5])

...
IndexError: list index out of range
```

Next steps: Explain error

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

Corrected code:

```
numbers = [1, 2, 3]
print(numbers[0])
```

Please explain this error:
IndexError: list index out of range
The IndexError: list index out of range means you're trying to access an element at an index that doesn't exist in the list. Your numbers list [1, 2, 3] only has elements at indices 0, 1, and 2. When you try to

Explanation:

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

