

# AI ASSISTED CODING

## ASSIGNMENT - 7.3

ROLL NO. 2303A510J0

### Task 1: Fixing Syntax Errors

#### Scenario

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

```
python

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

#### 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

Wrong code:

```
300 / p2 / 2044b.py / add
1  def add(a, b)
2  |     return a + b
```

## Output:

```
PS C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces> & C:/Users/rohit/AppData/Local/Programs/Python/Python314/python.exe c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/r_800/p2/2044b.py
File "c:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces\r_800\p2\2044b.py", line 1
    def add(a, b)
                ^
SyntaxError: expected ':'
PS C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces> & C:/Users/rohit/AppData/Local/Programs/Python/Python314/python.exe c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/r_800/p2/2044b.py
```

## Explanation:

CHAT

SYNTAX ERROR TROUBLESHOOTING IN CODE

@workspace /explain Expected ":"

</> Workspace

> Used 1 reference

The code in `2044b.py` has a syntax error on line 1. In Python, function definitions must end with a colon ( `:` ) after the parameter list to indicate the start of the function body. Without it, the parser expects a colon and raises an error.

Here's the corrected version:

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

This follows Python's syntax rules for defining functions, ensuring the code runs without errors. If you have more context or other issues, provide details for further help.

🔄 ↶ 🍏 🗨

Grok Code Fast 1 • 1x

📎 + 2044b.py

Describe what to build next

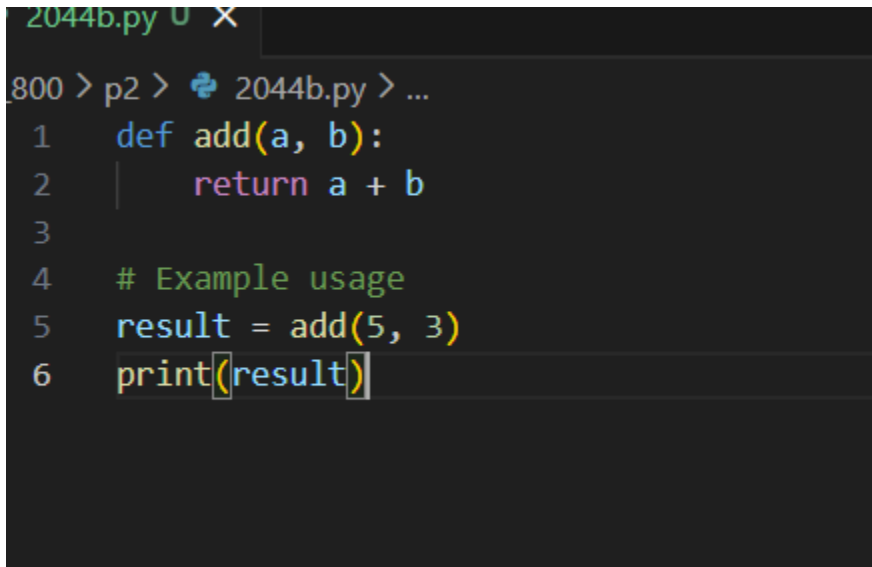
Agent ▾ Auto ▾ 🧑🏻

➡ ➤

Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Go Live 🔔

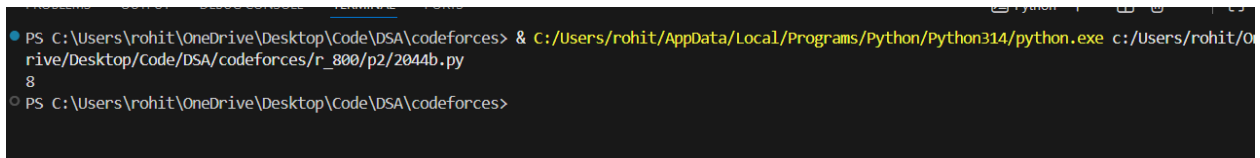
Prompt:  
Fix the code

Code:



```
2044b.py
800 > p2 > 2044b.py > ...
1  def add(a, b):
2      return a + b
3
4  # Example usage
5  result = add(5, 3)
6  print(result)
```

Output:



```
PS C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces> & c:/Users/rohit/AppData/Local/Programs/Python/Python314/python.exe c:/Users/rohit/OneDrive/Desktop/Code/DSA/codeforces/r_800/p2/2044b.py
8
PS C:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces>
```

Justification:

This task demonstrates how AI tools can quickly identify and resolve basic syntax errors in Python programs. Syntax errors, such as missing colons in function definitions, prevent code execution entirely. Using AI for detection helps reduce manual debugging time and provides clear explanations of language rules. This improves understanding of Python syntax and reinforces correct coding practices.

## Task 2: Debugging Logic Errors in Loops

### Scenario

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

python

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

### 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
- 

Wrong code:


```
def count_down(n):  
    while n >= 0:  
        print(n)  
        n += 1  
  
print(count_down(5))
```

Output: (infinite loop had to keyboard interrupt it)

```
200650
200651
200652
200653
200654
Traceback (most recent call last):
  File "c:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces\r_800\p2\2044b.py", line 6, in <module>
    print(count_down(5))
    ~~~~~^~~~~~
  File "c:\Users\rohit\OneDrive\Desktop\Code\DSA\codeforces\r_800\p2\2044b.py", line 3, in count_down
    print(n)
    ~~~~~^
KeyboardInterrupt
```



Explanation:

Gemini

 Please explain this code:

```
<> def count_down(n):
```

✦ This Python code defines a function called `count_down` that takes one argument `n`. The function contains a `while` loop that intends to print the value of `n` as long as `n` is greater than or equal to 0. However, inside the loop, `n` is incremented (`n += 1`), which means if `n` starts non-negative, the condition `n >= 0` will always remain true, leading to an infinite loop. The last line then calls this function with `n=5` and tries to print its return value, but the function will never return due to the infinite loop.



How can I install Python libraries?

Load data from Google Drive

Show an example of training a simple ML model

What can I help you build?

+

Gemini 2.5 Flash

➤

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Prompt:

Fix the logical bug in the function in the code

Code:

```
def count_down(n):  
    while n >= 0:  
        print(n)  
        n -= 1  
  
count_down(5)
```

Output:

```
... 5  
    4  
    3  
    2  
    1  
    0
```

Justification:

This task focuses on identifying logical errors that cause unintended infinite loops. Such errors are harder to detect than syntax errors because the program runs but behaves incorrectly. AI-assisted debugging helps analyze loop conditions and variable updates, ensuring proper termination. This task highlights the importance of correct logic flow and demonstrates how AI can explain and correct algorithmic mistakes.

---

### Task 3: Handling Runtime Errors (Division by Zero)

Scenario

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

```
# Debug the following code
def divide(a, b):
    return a / b

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

### 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

Wrong code:

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

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

Output:

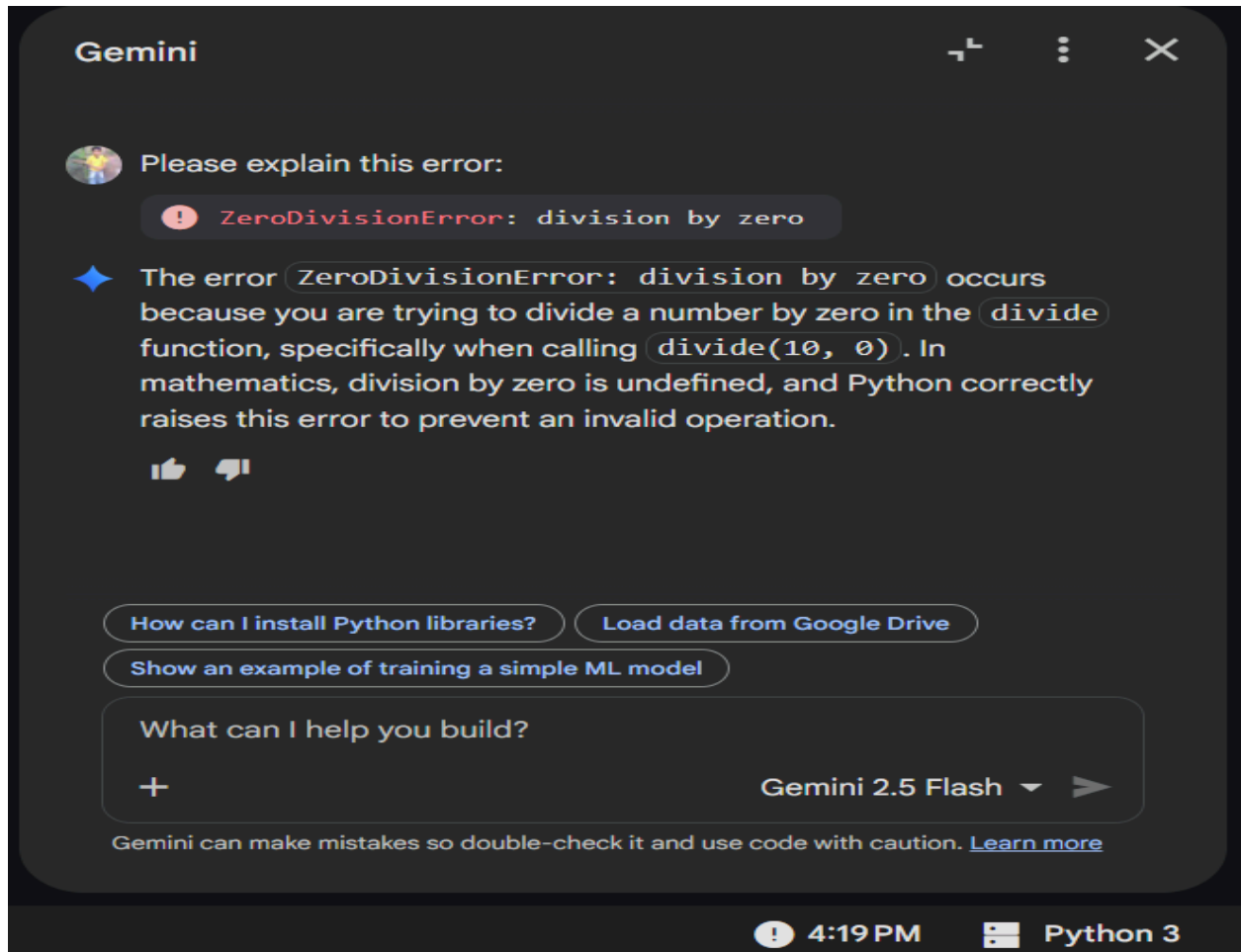
```
***
ZeroDivisionError                                Traceback (most recent call last)
/tmp/ipython-input-2311834636.py in <cell line: 0>()
      2     return a / b
      3
----> 4 print(divide(10, 0))

/tmp/ipython-input-2311834636.py in divide(a, b)
      1 def divide(a, b):
----> 2     return a / b
      3
      4 print(divide(10, 0))

ZeroDivisionError: division by zero

Next steps: Explain error
```

Explanation:



Prompt:  
Fix the logical bug in the code

Fixed code:

```
def divide(a, b):  
    return a / b  
  
try:  
    print(divide(10, 0))  
except ZeroDivisionError:  
    print("Error: Cannot divide by zero")
```

Output:

```
... Error: Cannot divide by zero
```



Justification:

Runtime errors occur during program execution and can cause unexpected crashes. This task illustrates how AI detects runtime exceptions such as division by zero and suggests structured error handling using try-except blocks. Implementing exception handling improves program robustness and reliability. AI explanations help understand defensive programming techniques and safe execution strategies.

---

#### Task 4: Debugging Class Definition Errors

##### Scenario

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

```
python
```

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

##### 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
-

Wrong code:

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

r1 = Rectangle(5, 6)
```


Output:


```
-----
TypeError                                Traceback (most recent call last)
/tmp/ipython-input-159090773.py in <cell line: 0>()
      4     self.width = width
      5
----> 6 r1 = Rectangle(5, 6)

TypeError: Rectangle.__init__() takes 2 positional arguments but 3 were given
```

Explanation:

Gemini

 Please explain this error:

 `TypeError: Rectangle.__init__() takes 2 positional argu`

✦ The error `TypeError: Rectangle.__init__() takes 2 positional arguments but 3 were given` means that the `__init__` method of your `Rectangle` class was called with an unexpected number of arguments. In Python, the `__init__` method, like all instance methods, automatically receives the instance itself as its first argument, which is conventionally named `self`. Your current `__init__` definition `def __init__(length, width):` expects two arguments after `self` is implicitly passed.


Fix the Rectangle class `__init__` method

Add `self` to the Rectangle class constructor

Instantiate the Rectangle class with `length` and `width`

What can I help you build?

+

Gemini 2.5 Flash 

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Prompt:

Fix the class in the code

Fixed code:

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

r1 = Rectangle(5, 6)
print(r1.length)
print(r1.width)
```

Output:

```
... 5
     6
```

Justification:

This task addresses common object-oriented programming mistakes, specifically incorrect constructor definitions. The absence of the `self` parameter leads to runtime errors and improper object initialization. AI-assisted debugging identifies this issue and explains the role of `self` in accessing instance variables. This task strengthens understanding of Python's class structure and object behavior.

## Task 5: Resolving Index Errors in Lists

### Scenario

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

```
python

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

### 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

Wrong code:

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

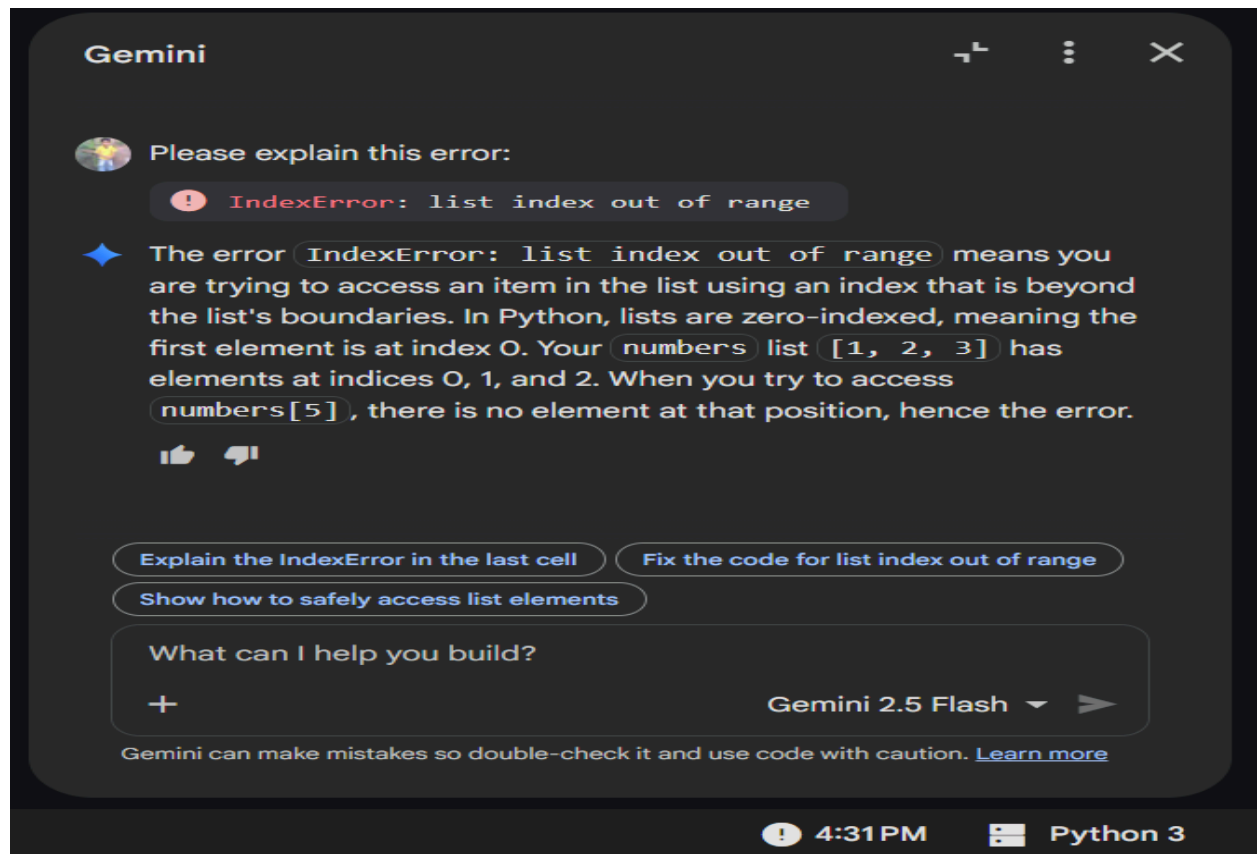
Output:

```
... -----
IndexError                                Traceback (most recent call last)
/tmp/ipython-input-1537628302.py in <cell line: 0>()
      1 numbers = [1, 2, 3]
----> 2 print(numbers[5])

IndexError: list index out of range
```

Next steps: [Explain error](#)

Explanation:



Prompt:

Fix the Error in the code

Fixed code:

```
numbers = [1, 2, 3]

try:
    print(numbers[5])
except IndexError:
    print("Error: Index is out of range for the list.")
```

## Output:

```
... Error: Index is out of range for the list.
```

## Justification:

Index errors occur when attempting to access elements outside the valid range of a list. This task demonstrates how AI identifies such errors and recommends safe access techniques like bounds checking or exception handling. Proper list handling prevents program crashes and ensures data safety. The AI explanations promote writing more reliable and error-resistant code.