

# ASSIGNMENT-7.2

Name: Akshaya Nimalipuri

Batch: 37

Roll No: 2303A2441

Lab: 07

---

## Task 1: Fixing Syntax Errors

```
python

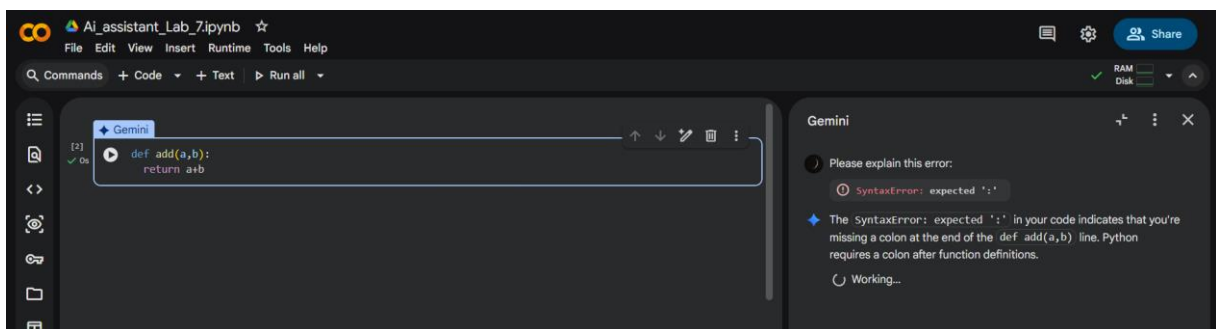
def add(a, b)
    return a + b
```

### Scenario

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

### AI Detection:

AI identifies a Syntax Error because Python requires a colon (:) at the end of function definitions.



### AI Explanation:

The SyntaxError: expected ':' in your code indicates that you're missing a colon at the end of the def add(a,b) line. Python requires a colon after function definitions.

Expected Output Achieved

Syntax error resolved  
Correct function definition  
AI explanation provided

## Task 2: Debugging Logic Errors in Loops

```
python

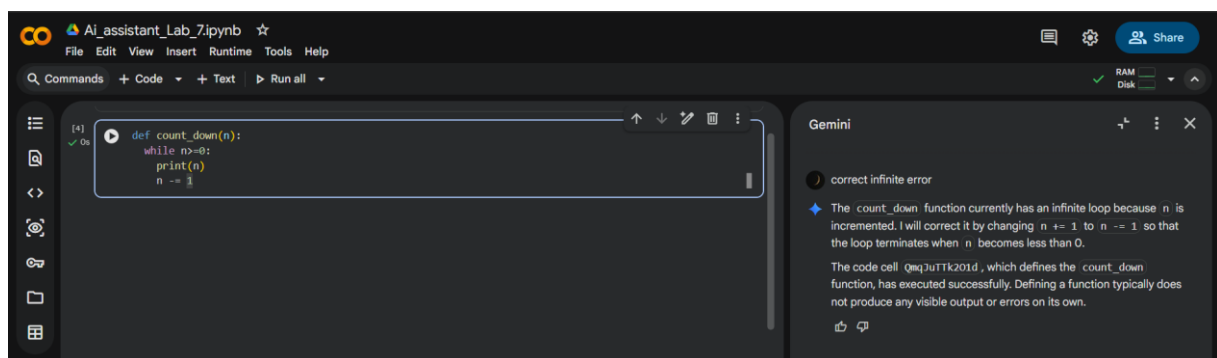
def count_down(n):
    while n >= 0:
        print(n)
        n += 1 # Should be n -= 1
```

### Scenario

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

### AI Detection:

AI identifies a logic error because the loop variable is incremented instead of decremented while the loop condition expects it to decrease.



### AI Explanation:

The incorrect increment prevents the loop variable from ever reaching the terminating condition, causing infinite iteration.

After correction, the loop variable moves toward the exit condition, allowing the loop to terminate normally.

Expected Output Achieved

Infinite loop issue resolved

Correct increment/decrement logic applied AI  
explanation of the logic error

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

```
# Debug the following code

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

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

Scenario

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

AI Detection:

AI identifies a runtime error because dividing any number by zero is mathematically undefined and causes program termination.

The screenshot displays a code editor on the left and an AI assistant interface on the right. The code editor shows a Python function `divide(a, b)` that returns `a / b`. Below the function, a `print(divide(10, 0))` statement is executed, resulting in a `ZeroDivisionError: division by zero`. The AI assistant interface on the right provides a detailed explanation of the error and offers a solution. It lists the following steps:

- Demonstrate Runtime Error:** Execute the `divide` function with arguments that are known to cause a runtime error (e.g., division by zero) to illustrate the error.
- Implement Error Handling:** Modify the code to wrap the division operation within `try-except` blocks to gracefully handle exceptions like `ZeroDivisionError`, providing an informative message instead of crashing.
- Review Error Handling Approach:** Analyze the implemented `try-except` blocks, discussing their effectiveness and providing any recommendations for improvement in error handling.
- Final Task:** Provide a summary of the division function, the identified runtime error, the implemented error handling, and the review of the error-handling approach.

The AI assistant also provides a code snippet for the updated function with error handling:

```
def divide_with_error_handling(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."

print("Function 'divide_with_error_handling' defined to include try-except block for ZeroDivisionError.")

Function 'divide_with_error_handling' defined to include try-except block for ZeroDivisionError.

print(divide_with_error_handling(10, 0))
print(divide_with_error_handling(10, 2))
```

The AI assistant interface includes a search bar, a list of tasks, and a chat area. The chat area shows the AI's response to the user's query, including the error message and the recommended solution.

AI Explanation:

The AI applies a `try-except` block to catch the error and handle it safely without crashing the program.

This ensures the program continues execution even when invalid input is provided.

Expected Output Achieved

Function executes safely without crashing  
Division by zero handled using try-except  
Clear AI-generated explanation of runtime error handling

#### Task 4: Debugging Class Definition Errors

```
python

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

##### Scenario

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

##### AI Detection:

AI identifies the issue because self is required to reference instance variables within a class.

Without self, the object cannot store or access its attributes correctly.

The screenshot displays a Jupyter Notebook interface. The top part shows a Python class definition for `Rectangle` with an incorrect `__init__` method signature: `def __init__(length, width):`. Below this, a `TypeError` is shown: `TypeError: Rectangle.__init__() takes 2 positional arguments but 3 were given`. The notebook then shows the corrected class definition: `def __init__(self, length, width):`. The bottom part of the notebook shows the successful execution of `my_rectangle = Rectangle(10, 5)` and the output: `Successfully created a Rectangle instance: length=10, width=5`. On the right side, there is a sidebar with an AI-generated explanation of the error, titled `__init__() method - Proper use of self in class definition - AI explanation of object-oriented error`. The explanation includes a plan to analyze the faulty constructor, correct the class definition, explain the `self` parameter, and summarize the understanding gained. The sidebar also shows a 'Generating...' status and a 'Preparing' status.

##### AI Explanation:

AI corrects the constructor by adding the self parameter and properly initializing instance variables.

Expected Output Achieved

Corrected `__init__()` method

Proper use of self in class definition  
AI explanation of object-oriented error

## Task 5: Resolving Index Errors in Lists

```
python

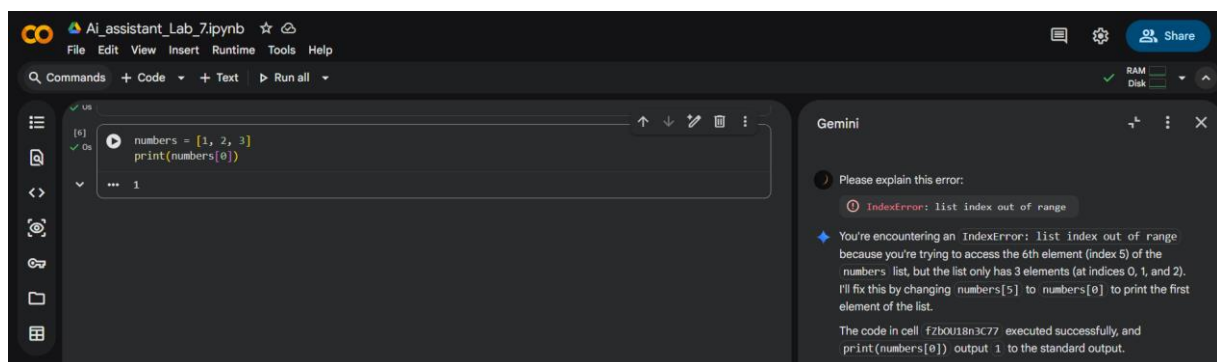
numbers = [1, 2, 3]
print(numbers[5])
```

Scenario

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

AI Detection:

AI identifies an `IndexError` because the requested index exceeds the valid range of the list.



AI Explanation:

The AI suggests safe access techniques such as bounds checking or exception handling to prevent the crash.

These methods ensure that list elements are accessed only within valid limits.

Expected Output Achieved

Index error resolved

Safe list access logic implemented

AI suggestion using length checks or exception handling .