

AI-Ass-7.3

Name:T.Shivani

Ht.No:2303A51312

Batch:05

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

Code: `result = add(5, 3)`

```
print(f"The result of add(5, 3) is: {result}")
```

```
result_negative = add(-1, 10)
```

```
print(f"The result of add(-1, 10) is: {result_negative}")
```

Output:The result of add(5, 3) is: 8

The result of add(-1, 10) is: 9

1. **Explanation:** **Intentional Error:** A Python function `add(a, b)` was initially defined without the required colon, causing a `SyntaxError: expected ':' when executed.`
2. **Error Detection:** The Python interpreter explicitly reported the `SyntaxError`, clearly indicating the missing colon.

3. **Correction:** The function definition was corrected by adding the colon (def add(a, b):).
4. **Verification:** The corrected function was then successfully executed and tested with example values.

The screenshot shows a Google Colab notebook titled "Lab Assignment 7.3.ipynb". The notebook contains a section titled "Correct Function Definition" with a subtask asking for the corrected version of the `add(a, b)` function. A reasoning note states that calling the function with example values will confirm its proper functionality. Below this, a code cell shows Python code that defines the `add` function and prints its results for inputs 5 and -1, 10. The output shows the correct results: 8 and 9 respectively. The notebook also includes a "Final Task" section and a "Summary" section.

```

[4] On
result = add(5, 3)
print(f"The result of add(5, 3) is: {result}")

result_negative = add(-1, 10)
print(f"The result of add(-1, 10) is: {result_negative}")

```

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

Code: count = 0

```
while count < 5:  
    print(f"Current count: {count}")  
    count += 1
```

Output: Current count: 0

```
Current count: 1  
Current count: 2  
Current count: 3  
Current count: 4
```

The screenshot shows a Google Colab notebook titled "Ai-Ass-7.3.ipynb". The code cell contains a while loop that prints the current value of count (0 to 4) and increments count by 1. The output pane shows the repeated output of "Current count: 0" followed by "KeyboardInterrupt". Below the code, there are two sections of instructions. The first section describes the task of creating an infinite loop by forgetting to increment count. The second section provides the correct solution, which includes incrementing count by 1 inside the loop.

```
KeyboardInterrupt:  
Gemini  
[6] 0s  
count = 0  
while count < 5:  
    print(f"Current count: {count}")  
    count += 1  
...  
Current count: 0  
Current count: 1  
Current count: 2  
Current count: 3  
Current count: 4  
  
Instructions  
1. Initialize a variable, for example, count, to 0.  
2. Create a while loop with the condition that count is less than 5.  
3. Inside the loop, print the current value of count.  
4. Crucially, omit any statement that would increment or decrement the count variable, ensuring the loop condition is always true and thus creating an infinite loop.  
  
Instructions  
1. Initialize a variable, for example, count, to 0.  
2. Create a while loop with the condition that count is less than 5.  
3. Inside the loop, print the current value of count.  
4. Crucially, omit any statement that would increment or decrement the count variable, ensuring the loop condition is always true and
```

Explanation: We introduced an infinite loop by forgetting to increment the count variable in a while loop. This caused the loop condition (`count < 5`) to always be true, printing "Current count: 0" repeatedly. We then corrected this by adding `count += 1` inside the loop, which made it terminate correctly and print counts from 0 to 4. This highlights the importance of modifying loop control variables to ensure termination.

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

Code: def divide_safe(a, b):

```
try:
    result = a / b
    return result
except ZeroDivisionError:
    return "Error: Cannot divide by zero"
except TypeError:
    return "Error: Invalid input type"
```

Example calls

```
print(divide_safe(10, 2))
print(divide_safe(10, 0))
print(divide_safe(10, "a"))
```

Output:

5.0

Error: Cannot divide by zero

Error: Invalid input type

The screenshot shows a Google Colab notebook titled "Ai-Ass-7.3.ipynb". It contains two code cells. Cell [1] contains the original function definition:`def divide(a, b):
 return a / b`Cell [2] contains the refactored function definition with error handling:`def divide_safe(a, b):
 try:
 result = a / b
 return result
 except ZeroDivisionError:
 return "Error: Cannot divide by zero"
 except TypeError:
 return "Error: Invalid input type"

Example calls
print(divide_safe(10, 2))
print(divide_safe(10, 0))
print(divide_safe(10, "a"))`Output from cell [2] shows the results of the example calls:`5.0
Error: Cannot divide by zero
Error: Invalid input type`At the bottom, the status bar shows "2:18 PM" and "Python 3".

Explanation:

The original division function crashes when dividing by zero.

- AI identifies the runtime error as **ZeroDivisionError**.
- AI fixes it using try-except to prevent program failure.
- Error handling makes the code safer and more reliable.
- Using specific exceptions improves clarity and control.

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

Code:

class Student:

```
def __init__(self, name, age):
    self.name = name
    self.age = age
```

s = Student("Ravi", 20)

print(s.name, s.age)

Output: Ravi 20

The screenshot shows a Google Colab notebook titled "Ai-Ass-7.3.ipynb". The code cell [18] contains the following Python code:

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

s = Student("Ravi", 20)
print(s.name, s.age)
```

The output cell [18] shows the result of running the code:

```
Ravi 20
```

At the top of the screen, there are tabs for "Recent Canvas Notifications", "Lab Assignment 7.3", and "Ai-Ass-7.3.ipynb - Colab". The status bar at the bottom shows "2:22PM", "Python 3", and the date "28-01-2026".

Explanation: self refers to the current object.

Without self, Python cannot store data inside the object.

AI identifies the missing self and fixes the constructor.

Using self.variable correctly binds data to the object.

Task 5 : Task 5Resolving 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

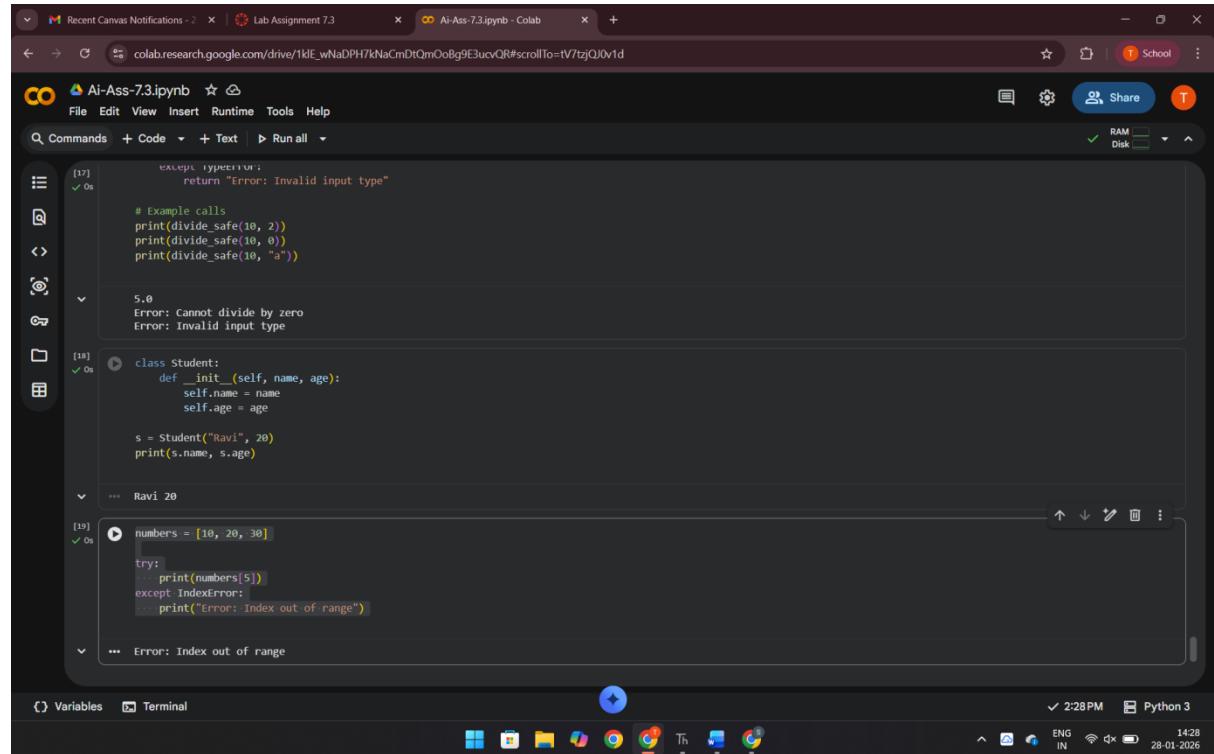
Expected Output

- Index error resolved
- Safe list access logic implemented

Code: numbers = [10, 20, 30]

try:

```
    print(numbers[5])  
  
except IndexError:  
  
    print("Error: Index out of range")
```



The screenshot shows a Google Colab notebook titled "Ai-Ass-7.3.ipynb". Cell 17 contains code for a divide-safe function and its usage. Cell 18 contains code for a Student class and its instantiation. Cell 19 contains the provided code for printing a list element. The output of cell 19 shows an IndexError: "Error: Index out of range". The status bar at the bottom indicates the time as 2:28 PM and the date as 28-01-2026.

Output: Error: Index out of range

Explanation: Accessing an invalid index causes IndexError.

AI detects the error and prevents program crash.

try-except or bounds checking ensures safe list access.

This makes the program stable and error-free