SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	Dr. Rishabh Mittal
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju

Course Code	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week4 – Wednesday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches

AssignmentNumber:7.3(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	**Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs** **Lab Objectives** • To identify and correct syntax, logic, and runtime errors in Python programs using AI tools	Week4 - Wednesday

- To understand common programming bugs and AI-assisted debugging suggestions
- To evaluate how AI explains, detects, and fixes different types of coding errors
- To build confidence in using AI for structured debugging practices

---

Lab Outcomes (LOs)

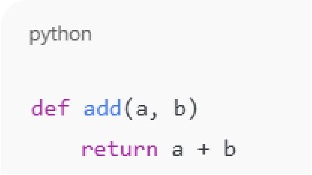
After completing this lab, students will be able to:

- Use AI tools to detect and correct syntax, logic, and runtime errors
- Interpret AI-suggested bug fixes and explanations
- Apply systematic debugging strategies using AI-generated insights
- Refactor buggy code using reliable programming patterns

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
```

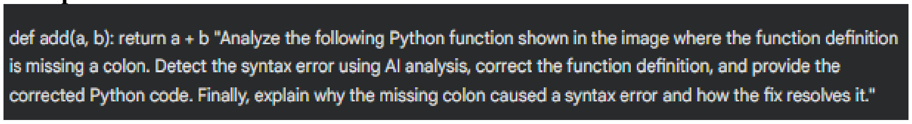
Requirements

- 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

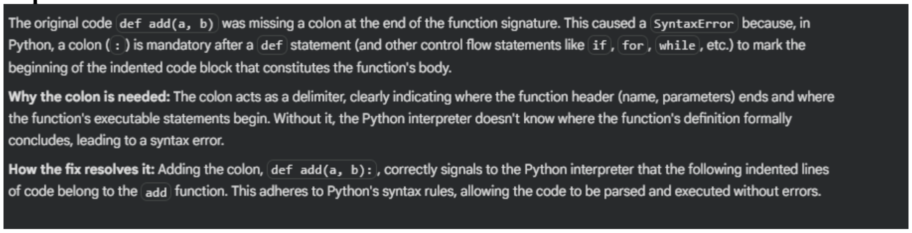
- Corrected function with proper syntax
- Syntax error resolved successfully
- AI-generated explanation of the fix

Prompt:



def add(a, b): return a + b "Analyze the following Python function shown in the image where the function definition is missing a colon. Detect the syntax error using AI analysis, correct the function definition, and provide the corrected Python code. Finally, explain why the missing colon caused a syntax error and how the fix resolves it."

Explanation of the Error:

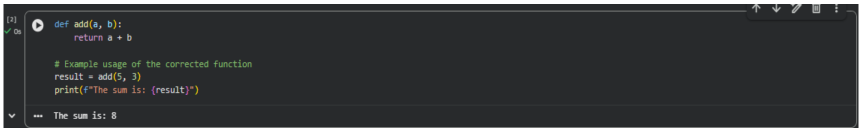


The original code `def add(a, b)` was missing a colon at the end of the function signature. This caused a `SyntaxError` because, in Python, a colon (`:`) is mandatory after a `def` statement (and other control flow statements like `if`, `for`, `while`, etc.) to mark the beginning of the indented code block that constitutes the function's body.

Why the colon is needed: The colon acts as a delimiter, clearly indicating where the function header (name, parameters) ends and where the function's executable statements begin. Without it, the Python interpreter doesn't know where the function's definition formally concludes, leading to a syntax error.

How the fix resolves it: Adding the colon, `def add(a, b):`, correctly signals to the Python interpreter that the following indented lines of code belong to the `add` function. This adheres to Python's syntax rules, allowing the code to be parsed and executed without errors.

Code:

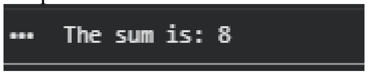


```python
def add(a, b):
    return a + b

# Example usage of the corrected function
result = add(5, 3)
print(f"The sum is: {result}")
```

```
... The sum is: 8
```

Output:

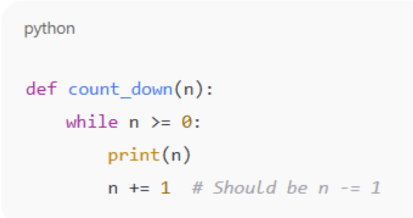


```
... The sum is: 8
```

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
```

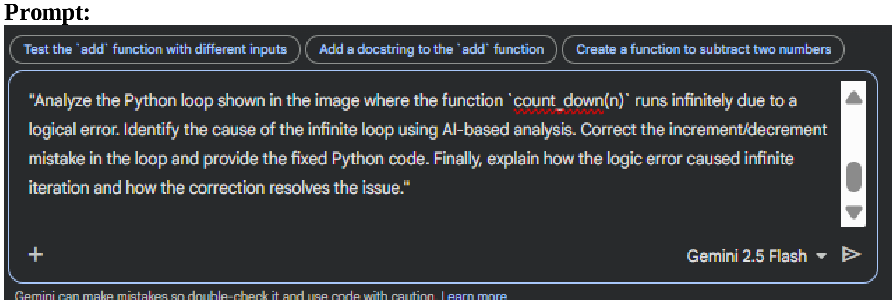
Requirements

- 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

Prompt:

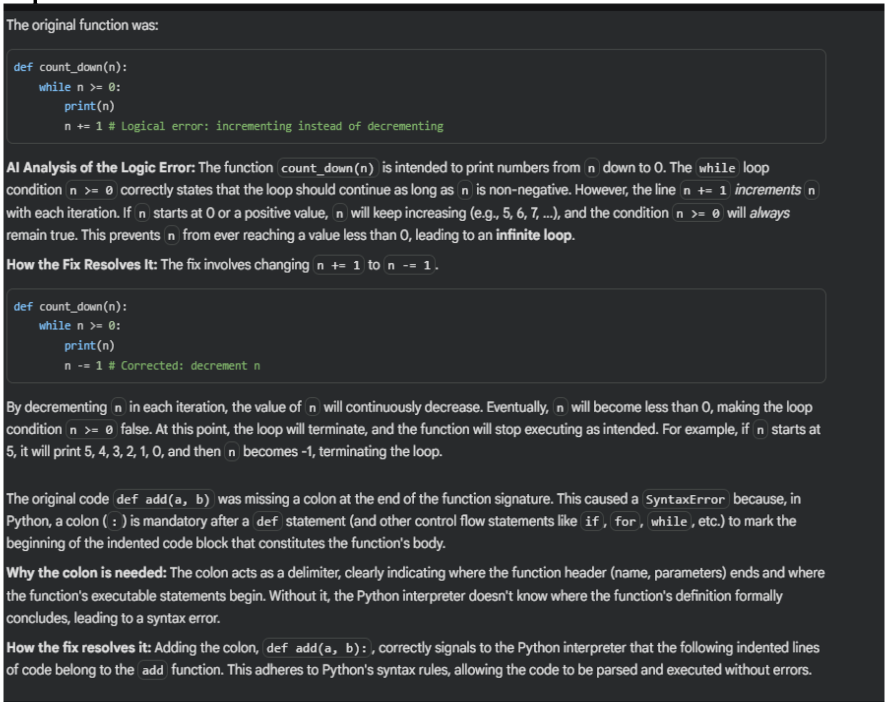


Test the `add` function with different inputs
Add a docstring to the `add` function
Create a function to subtract two numbers
+
Gemini 2.5 Flash

"Analyze the Python loop shown in the image where the function `count_down(n)` runs infinitely due to a logical error. Identify the cause of the infinite loop using AI-based analysis. Correct the increment/decrement mistake in the loop and provide the fixed Python code. Finally, explain how the logic error caused infinite iteration and how the correction resolves the issue."

Gemini can make mistakes so double-check it and use code with caution. Learn more

Explanation of the Error:



The original function was:

```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1 # Logical error: incrementing instead of decrementing
```

AI Analysis of the Logic Error: The function `count_down(n)` is intended to print numbers from `n` down to 0. The `while` loop condition `n >= 0` correctly states that the loop should continue as long as `n` is non-negative. However, the line `n += 1` *increments* `n` with each iteration. If `n` starts at 0 or a positive value, `n` will keep increasing (e.g., 5, 6, 7, ...), and the condition `n >= 0` will *always* remain true. This prevents `n` from ever reaching a value less than 0, leading to an **infinite loop**.

How the Fix Resolves It: The fix involves changing `n += 1` to `n -= 1`.

```
def count_down(n):
    while n >= 0:
        print(n)
        n -= 1 # Corrected: decrement n
```

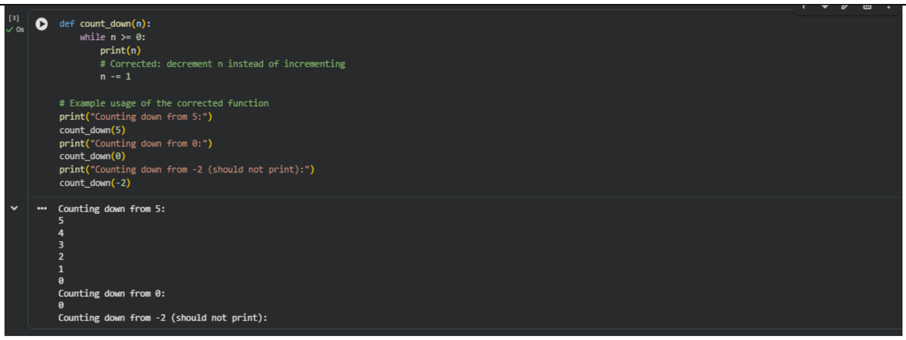
By decrementing `n` in each iteration, the value of `n` will continuously decrease. Eventually, `n` will become less than 0, making the loop condition `n >= 0` false. At this point, the loop will terminate, and the function will stop executing as intended. For example, if `n` starts at 5, it will print 5, 4, 3, 2, 1, 0, and then `n` becomes -1, terminating the loop.

The original code `def add(a, b)` was missing a colon at the end of the function signature. This caused a `SyntaxError` because, in Python, a colon (`:`) is mandatory after a `def` statement (and other control flow statements like `if`, `for`, `while`, etc.) to mark the beginning of the indented code block that constitutes the function's body.

Why the colon is needed: The colon acts as a delimiter, clearly indicating where the function header (name, parameters) ends and where the function's executable statements begin. Without it, the Python interpreter doesn't know where the function's definition formally concludes, leading to a syntax error.

How the fix resolves it: Adding the colon, `def add(a, b):`, correctly signals to the Python interpreter that the following indented lines of code belong to the `add` function. This adheres to Python's syntax rules, allowing the code to be parsed and executed without errors.

Code:

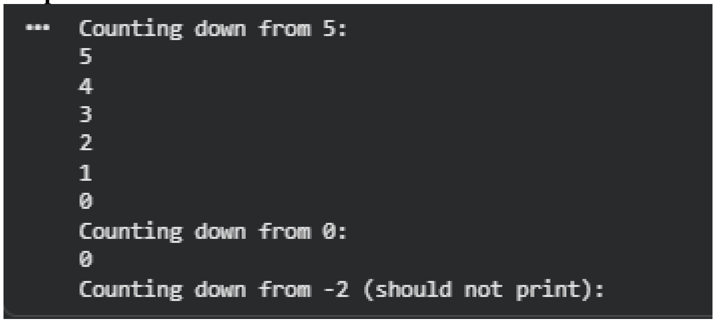


```
def count_down(n):
    while n >= 0:
        print(n)
        # Corrected: decrement n instead of incrementing
        n -= 1

# Example usage of the corrected function
print("Counting down from 5:")
count_down(5)
print("Counting down from 0:")
count_down(0)
print("Counting down from -2 (should not print):")
count_down(-2)
```

```
Counting down from 5:
5
4
3
2
1
0
Counting down from 0:
0
Counting down from -2 (should not print):
```

Output:



```
Counting down from 5:
5
4
3
2
1
0
Counting down from 0:
0
Counting down from -2 (should not print):
```

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))
```

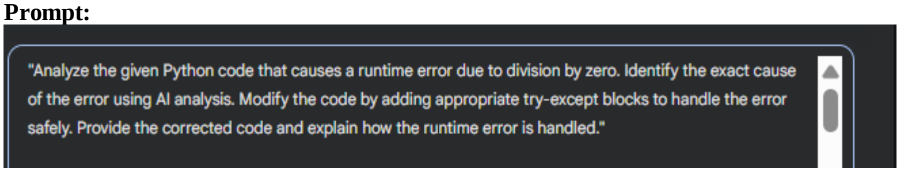
Requirements

- 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

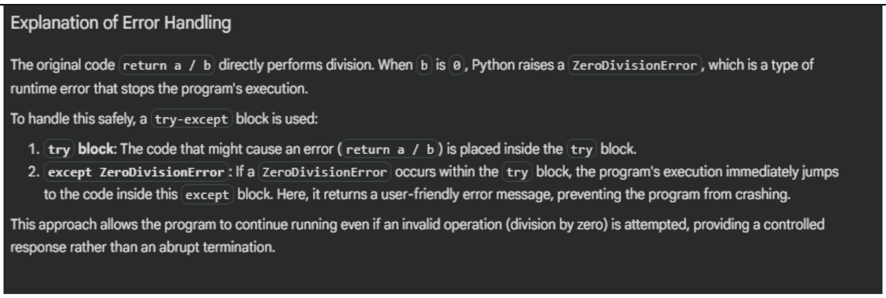
- Function executes safely without crashing
- Division by zero handled using try-except
- Clear AI-generated explanation of runtime error handling

Prompt:



"Analyze the given Python code that causes a runtime error due to division by zero. Identify the exact cause of the error using AI analysis. Modify the code by adding appropriate try-except blocks to handle the error safely. Provide the corrected code and explain how the runtime error is handled."

Explanation of the Error:



Explanation of Error Handling

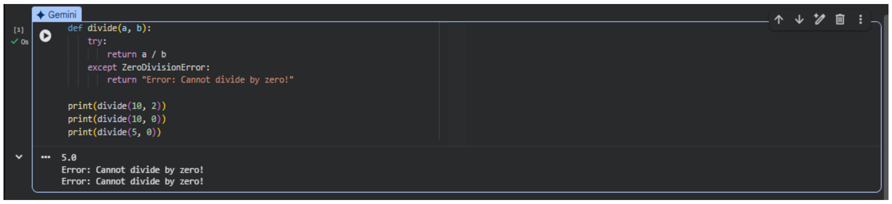
The original code `return a / b` directly performs division. When `b` is `0`, Python raises a `ZeroDivisionError`, which is a type of runtime error that stops the program's execution.

To handle this safely, a `try-except` block is used:

1. `try` **block:** The code that might cause an error (`return a / b`) is placed inside the `try` block.
2. `except ZeroDivisionError`: If a `ZeroDivisionError` occurs within the `try` block, the program's execution immediately jumps to the code inside this `except` block. Here, it returns a user-friendly error message, preventing the program from crashing.

This approach allows the program to continue running even if an invalid operation (division by zero) is attempted, providing a controlled response rather than an abrupt termination.

Code:

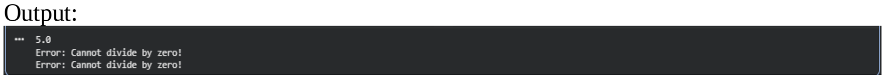


```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero!"

print(divide(10, 2))
print(divide(10, 0))
print(divide(5, 0))
```

```
5.0
Error: Cannot divide by zero!
Error: Cannot divide by zero!
```

Output:

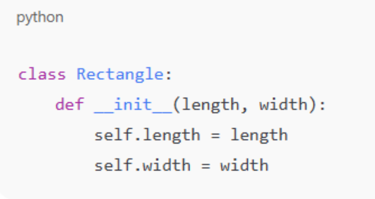


```
5.0
Error: Cannot divide by zero!
Error: Cannot divide by zero!
```

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
```

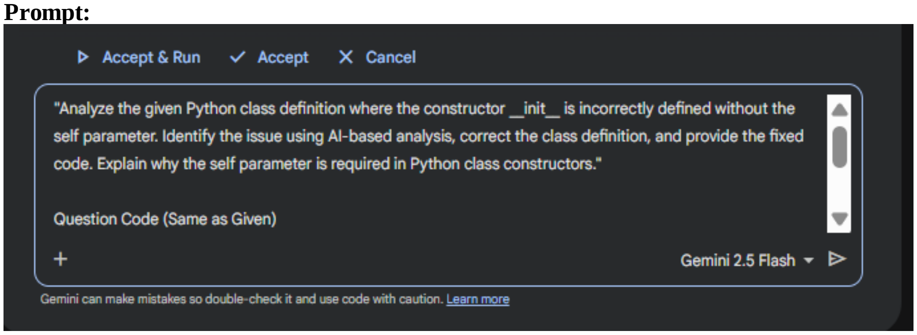
Requirements

- 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

Prompt:

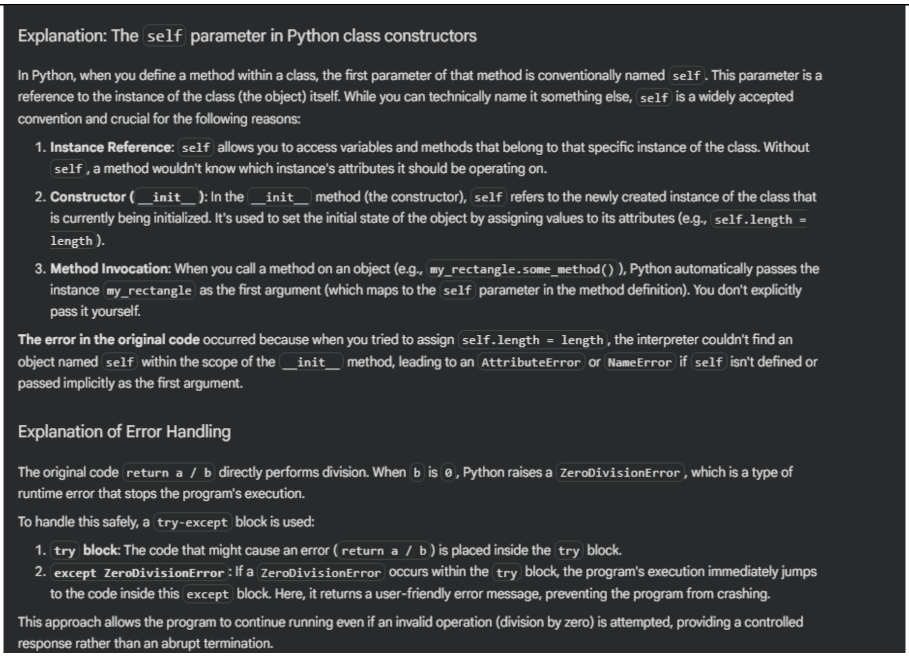


"Analyze the given Python class definition where the constructor __init__ is incorrectly defined without the self parameter. Identify the issue using AI-based analysis, correct the class definition, and provide the fixed code. Explain why the self parameter is required in Python class constructors."

Question Code (Same as Given)

Gemini can make mistakes so double-check it and use code with caution. Learn more

Explanation of the Error:



Explanation: The `self` parameter in Python class constructors

In Python, when you define a method within a class, the first parameter of that method is conventionally named `self`. This parameter is a reference to the instance of the class (the object) itself. While you can technically name it something else, `self` is a widely accepted convention and crucial for the following reasons:

1. **Instance Reference**: `self` allows you to access variables and methods that belong to that specific instance of the class. Without `self`, a method wouldn't know which instance's attributes it should be operating on.
2. **Constructor (`__init__`)**: In the `__init__` method (the constructor), `self` refers to the newly created instance of the class that is currently being initialized. It's used to set the initial state of the object by assigning values to its attributes (e.g., `self.length = length`).
3. **Method Invocation**: When you call a method on an object (e.g., `my_rectangle.some_method()`), Python automatically passes the instance `my_rectangle` as the first argument (which maps to the `self` parameter in the method definition). You don't explicitly pass it yourself.

The error in the original code occurred because when you tried to assign `self.length = length`, the interpreter couldn't find an object named `self` within the scope of the `__init__` method, leading to an `AttributeError` or `NameError` if `self` isn't defined or passed implicitly as the first argument.

Explanation of Error Handling

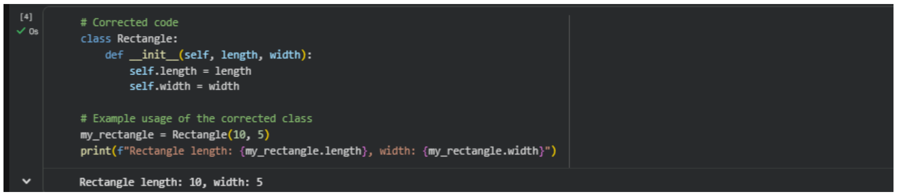
The original code `return a / b` directly performs division. When `b` is `0`, Python raises a `ZeroDivisionError`, which is a type of runtime error that stops the program's execution.

To handle this safely, a `try-except` block is used:

1. `try` **block**: The code that might cause an error (`return a / b`) is placed inside the `try` block.
2. `except ZeroDivisionError`: If a `ZeroDivisionError` occurs within the `try` block, the program's execution immediately jumps to the code inside this `except` block. Here, it returns a user-friendly error message, preventing the program from crashing.

This approach allows the program to continue running even if an invalid operation (division by zero) is attempted, providing a controlled response rather than an abrupt termination.

Code:

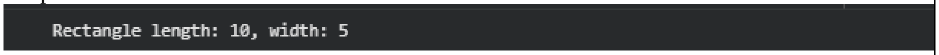


```python
# Corrected code
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

# Example usage of the corrected class
my_rectangle = Rectangle(10, 5)
print(f"Rectangle length: {my_rectangle.length}, width: {my_rectangle.width}")
```

```
Rectangle length: 10, width: 5
```

Output:

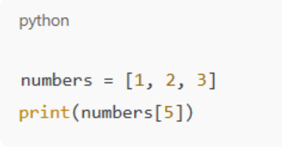


```
Rectangle length: 10, width: 5
```

Task 5: Resolving Index Errors in Lists

Scenario

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



python

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

Requirements

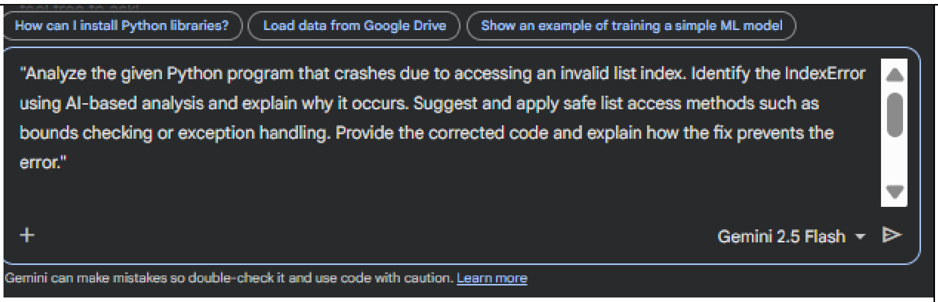
- Provide code that accesses an **out-of-range list index**
- Use AI to identify the Index Error
- Let AI suggest safe access methods
- Apply bounds checking or exception handling

Expected Output

- Index error resolved
- Safe list access logic implemented
- AI suggestion using length checks or exception handling

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Prompt:

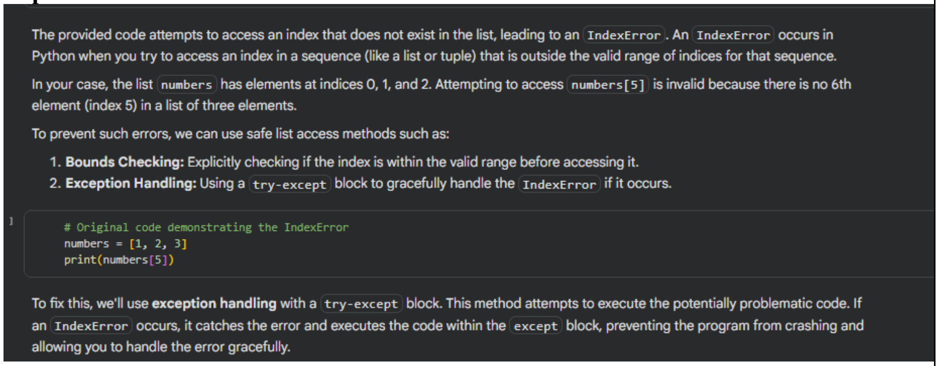

How can I install Python libraries?
Load data from Google Drive
Show an example of training a simple ML model
+

"Analyze the given Python program that crashes due to accessing an invalid list index. Identify the IndexError using AI-based analysis and explain why it occurs. Suggest and apply safe list access methods such as bounds checking or exception handling. Provide the corrected code and explain how the fix prevents the error."

Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more

Explanation of the Error:



The provided code attempts to access an index that does not exist in the list, leading to an `IndexError`. An `IndexError` occurs in Python when you try to access an index in a sequence (like a list or tuple) that is outside the valid range of indices for that sequence.

In your case, the list `numbers` has elements at indices 0, 1, and 2. Attempting to access `numbers[5]` is invalid because there is no 6th element (index 5) in a list of three elements.

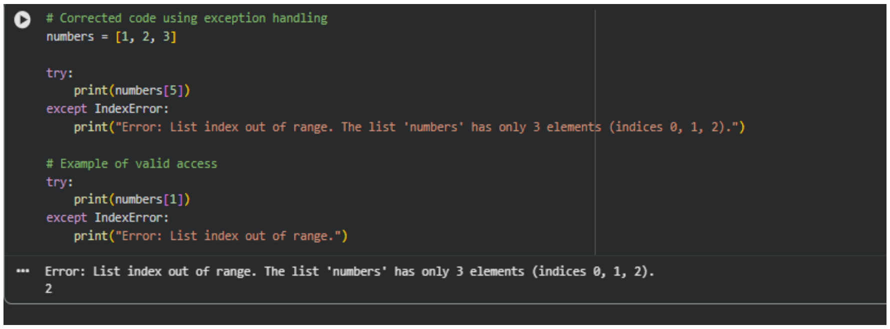
To prevent such errors, we can use safe list access methods such as:

1. **Bounds Checking:** Explicitly checking if the index is within the valid range before accessing it.
2. **Exception Handling:** Using a `try-except` block to gracefully handle the `IndexError` if it occurs.

```
# Original code demonstrating the IndexError
numbers = [1, 2, 3]
print(numbers[5])
```

To fix this, we'll use **exception handling** with a `try-except` block. This method attempts to execute the potentially problematic code. If an `IndexError` occurs, it catches the error and executes the code within the `except` block, preventing the program from crashing and allowing you to handle the error gracefully.

Code:



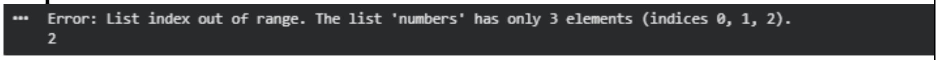
```
# Corrected code using exception handling
numbers = [1, 2, 3]

try:
    print(numbers[5])
except IndexError:
    print("Error: List index out of range. The list 'numbers' has only 3 elements (indices 0, 1, 2).")

# Example of valid access
try:
    print(numbers[1])
except IndexError:
    print("Error: List index out of range.")
```

```
Error: List index out of range. The list 'numbers' has only 3 elements (indices 0, 1, 2).
2
```

Output:



```
Error: List index out of range. The list 'numbers' has only 3 elements (indices 0, 1, 2).
2
```