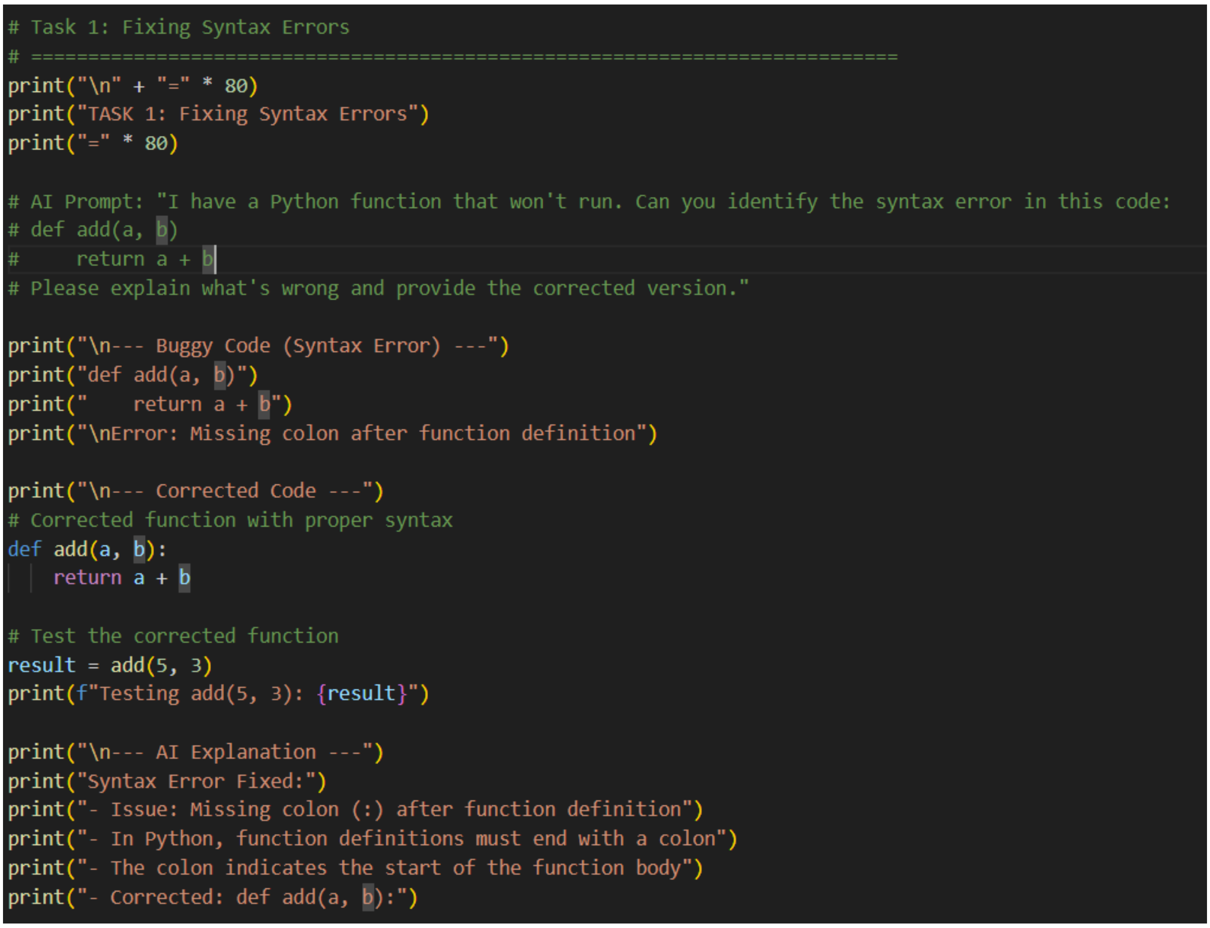
AI-ASSISSTANT-CODING-LAB-7.3

Name: G. Kruthik Roshan

Batch:41

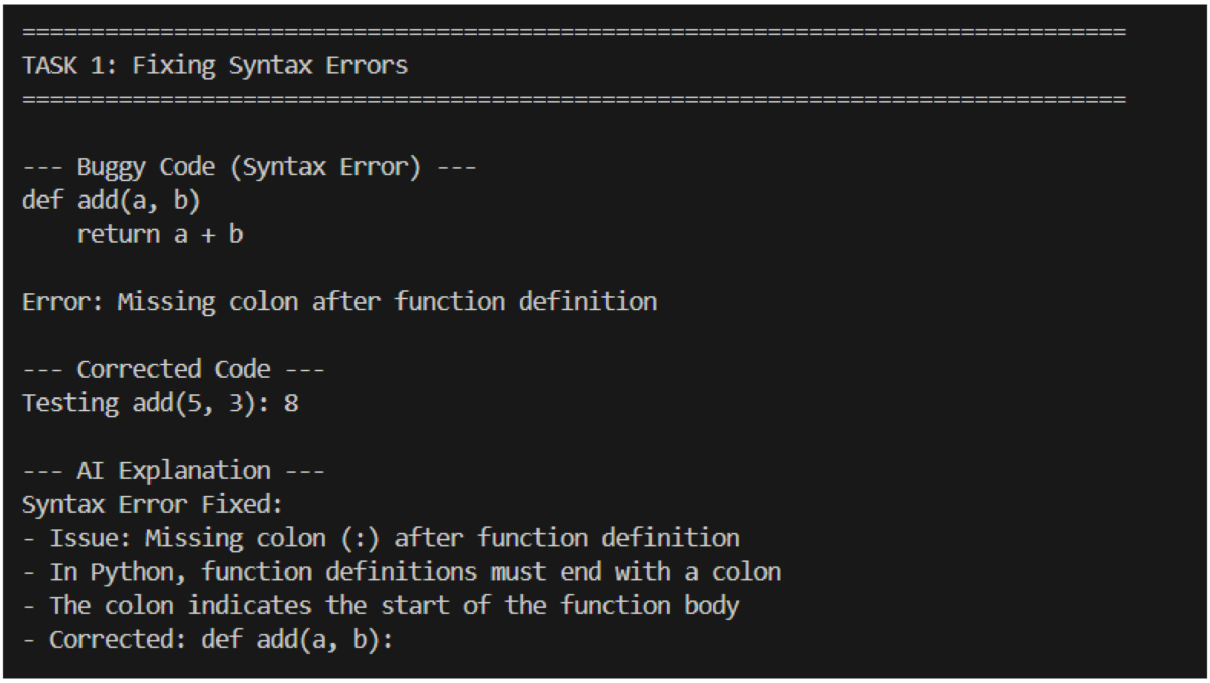
Roll-No:2303A52339

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
Prompt: I have a Python function that won't run. Can you identify the syntax error in this code:
Code:



```python
# Task 1: Fixing Syntax Errors
# ============================================================================
print("\n" + "=" * 80)
print("TASK 1: Fixing Syntax Errors")
print("=" * 80)

# AI Prompt: "I have a Python function that won't run. Can you identify the syntax error in this code:
# def add(a, b)
#     return a + b
# Please explain what's wrong and provide the corrected version."

print("\n--- Buggy Code (Syntax Error) ---")
print("def add(a, b)")
print("    return a + b")
print("\nError: Missing colon after function definition")

print("\n--- Corrected Code ---")
# Corrected function with proper syntax
def add(a, b):
    return a + b

# Test the corrected function
result = add(5, 3)
print(f"Testing add(5, 3): {result}")

print("\n--- AI Explanation ---")
print("Syntax Error Fixed:")
print("- Issue: Missing colon (:) after function definition")
print("- In Python, function definitions must end with a colon")
print("- The colon indicates the start of the function body")
print("- Corrected: def add(a, b):")
```

OUTPUT:



```
================================================================================
TASK 1: Fixing Syntax Errors
================================================================================

--- Buggy Code (Syntax Error) ---
def add(a, b)
    return a + b

Error: Missing colon after function definition

--- Corrected Code ---
Testing add(5, 3): 8

--- AI Explanation ---
Syntax Error Fixed:
- Issue: Missing colon (:) after function definition
- In Python, function definitions must end with a colon
- The colon indicates the start of the function body
- Corrected: def add(a, b):
```

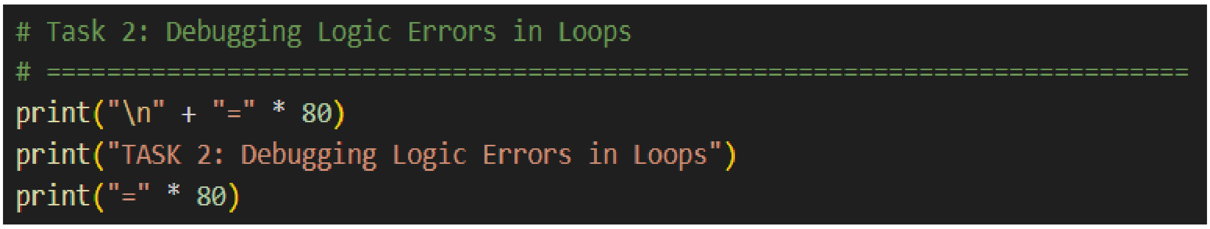
Justification:

The program failed because Python requires a colon (:) at the end of a function definition. This colon signals the start of a code block. Without it, Python cannot parse the structure of the program and stops execution before running any code. Fixing it ensures the interpreter understands the function body.

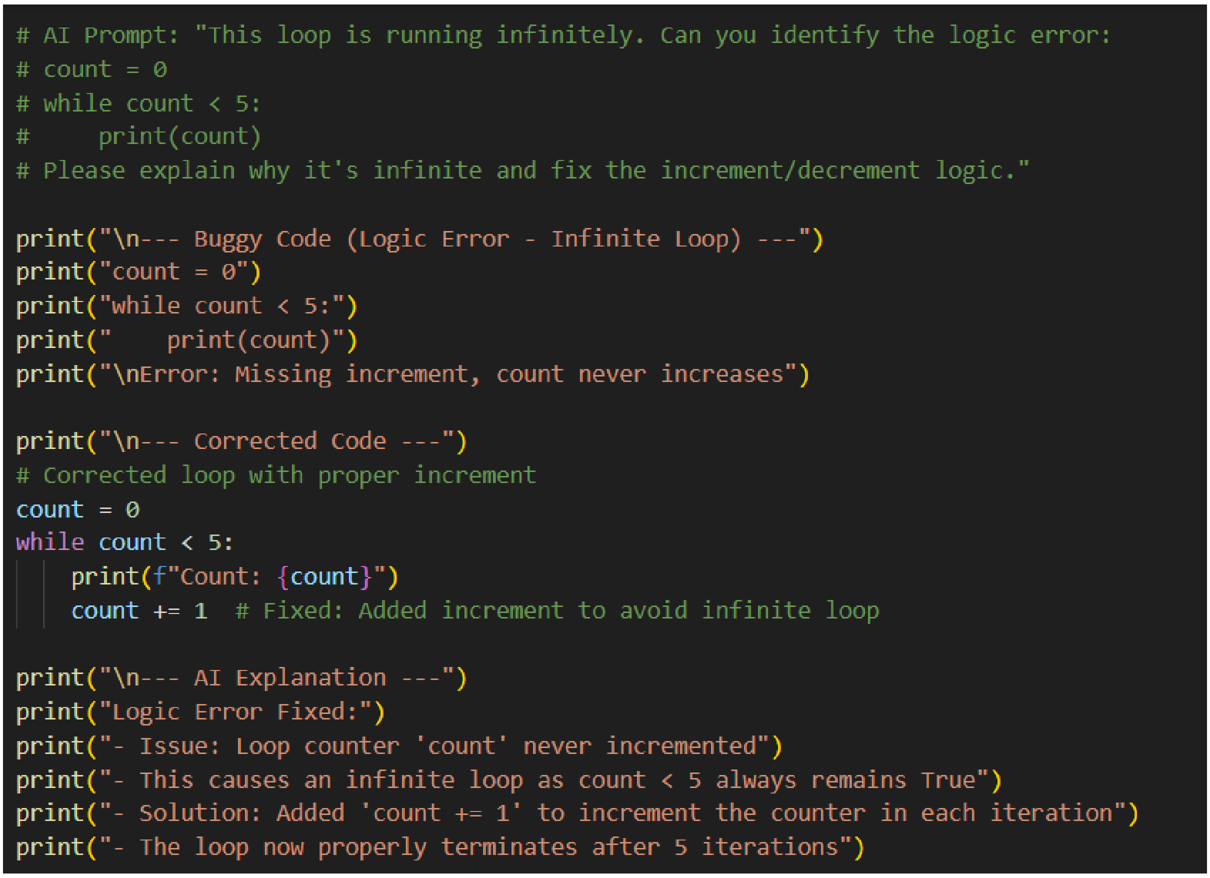
Task 2: Debugging Logic Errors in Loops
Scenario
You are debugging a loop that runs infinitely due to a logical mistake.
Requirements
• Provide a loop with an increment or decrement error
• Use AI to identify the cause of infinite iteration
• Let AI fix the loop logic
• Analyze the corrected loop behavior

Prompt: This loop is running infinitely. Can you identify the logic error:

Code:

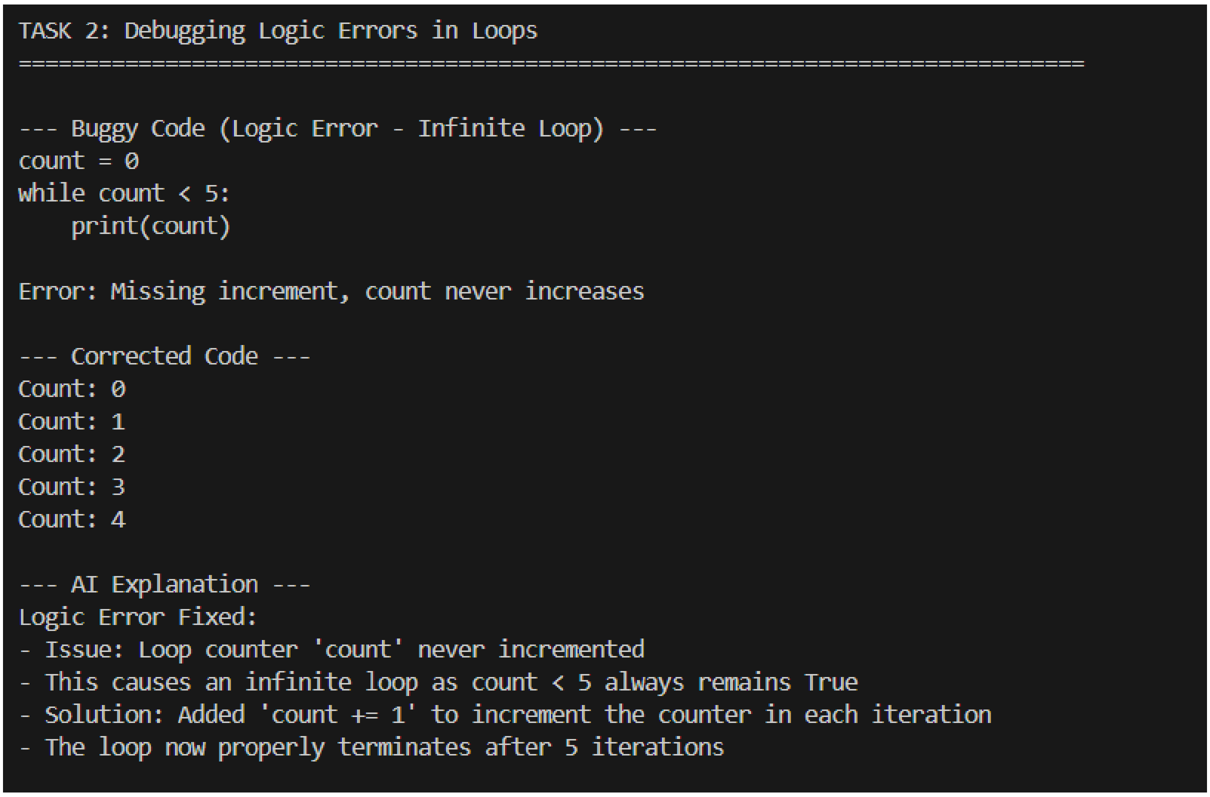


```
# Task 2: Debugging Logic Errors in Loops
# ============================================================================
print("\n" + "=" * 80)
print("TASK 2: Debugging Logic Errors in Loops")
print("=" * 80)
```



```python
# AI Prompt: "This loop is running infinitely. Can you identify the logic error:
# count = 0
# while count < 5:
#     print(count)
# Please explain why it's infinite and fix the increment/decrement logic."

print("\n--- Buggy Code (Logic Error - Infinite Loop) ---")
print("count = 0")
print("while count < 5:")
print("    print(count)")
print("\nError: Missing increment, count never increases")

print("\n--- Corrected Code ---")
# Corrected loop with proper increment
count = 0
while count < 5:
    print(f"Count: {count}")
    count += 1  # Fixed: Added increment to avoid infinite loop

print("\n--- AI Explanation ---")
print("Logic Error Fixed:")
print("- Issue: Loop counter 'count' never incremented")
print("- This causes an infinite loop as count < 5 always remains True")
print("- Solution: Added 'count += 1' to increment the counter in each iteration")
print("- The loop now properly terminates after 5 iterations")
```

OUTPUT:



```
TASK 2: Debugging Logic Errors in Loops
================================================================================

--- Buggy Code (Logic Error - Infinite Loop) ---
count = 0
while count < 5:
    print(count)

Error: Missing increment, count never increases

--- Corrected Code ---
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4

--- AI Explanation ---
Logic Error Fixed:
- Issue: Loop counter 'count' never incremented
- This causes an infinite loop as count < 5 always remains True
- Solution: Added 'count += 1' to increment the counter in each iteration
- The loop now properly terminates after 5 iterations
```

Justification:

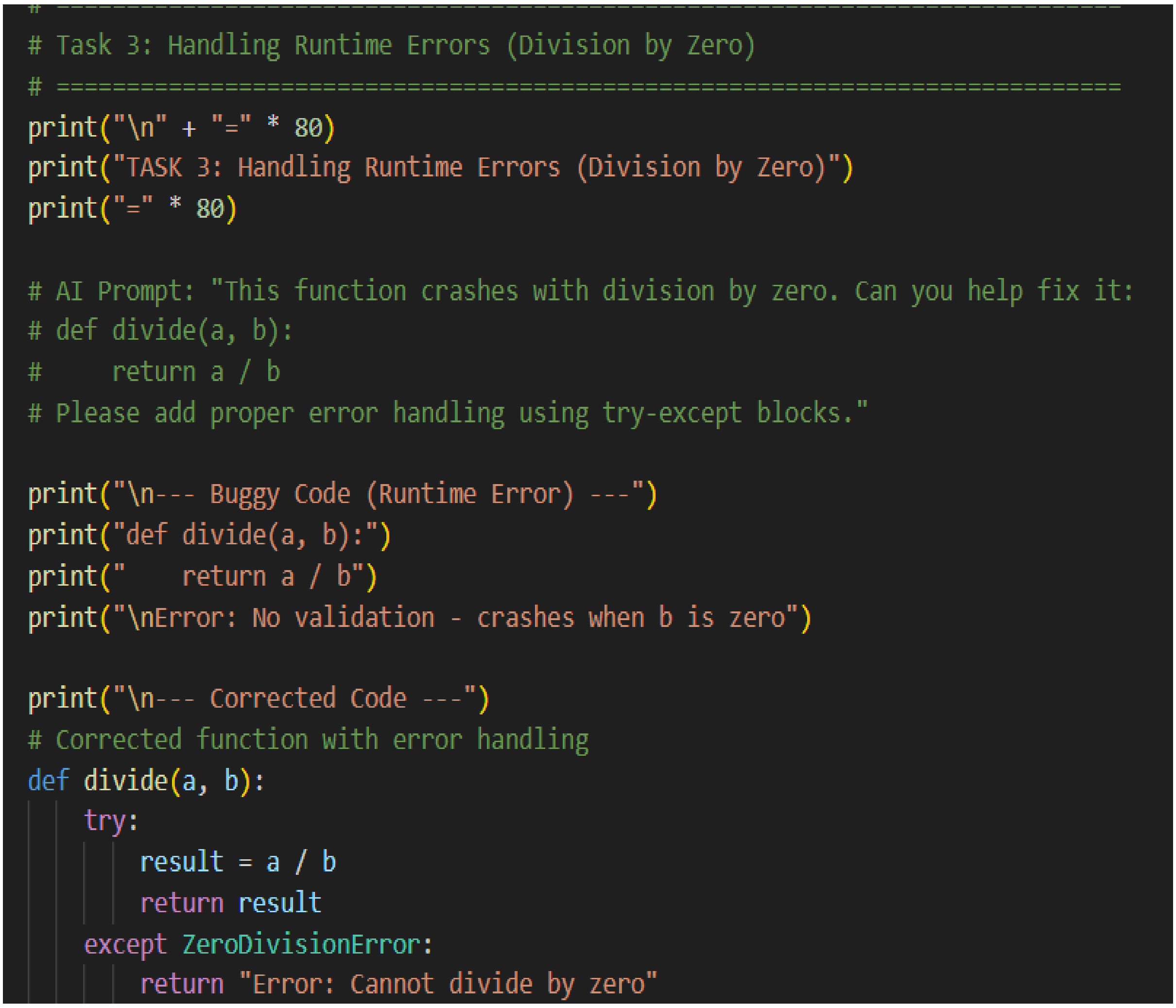
The loop condition depended on the variable count, but its value never changed. Since count remained 0, the condition count < 5 was always true, creating an infinite loop. Adding count += 1 updates the loop control variable, allowing the loop to terminate correctly after 5 iterations.

Task 3: Handling Runtime Errors (Division by Zero)
Scenario
A Python function crashes during execution due to a division by zero error.
Requirements

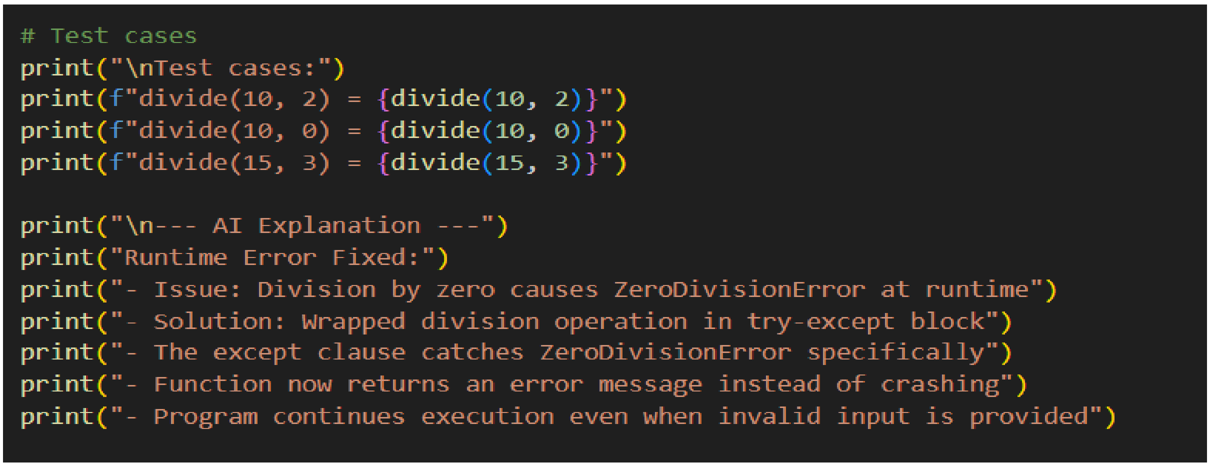
- Provide a function that performs division without validation
- Use AI to identify the runtime error
- Let AI add try-except blocks for safe execution
- Review AI's error-handling approach

Prompt: This function crashes with division by zero. Can you help fix it:

CODE:

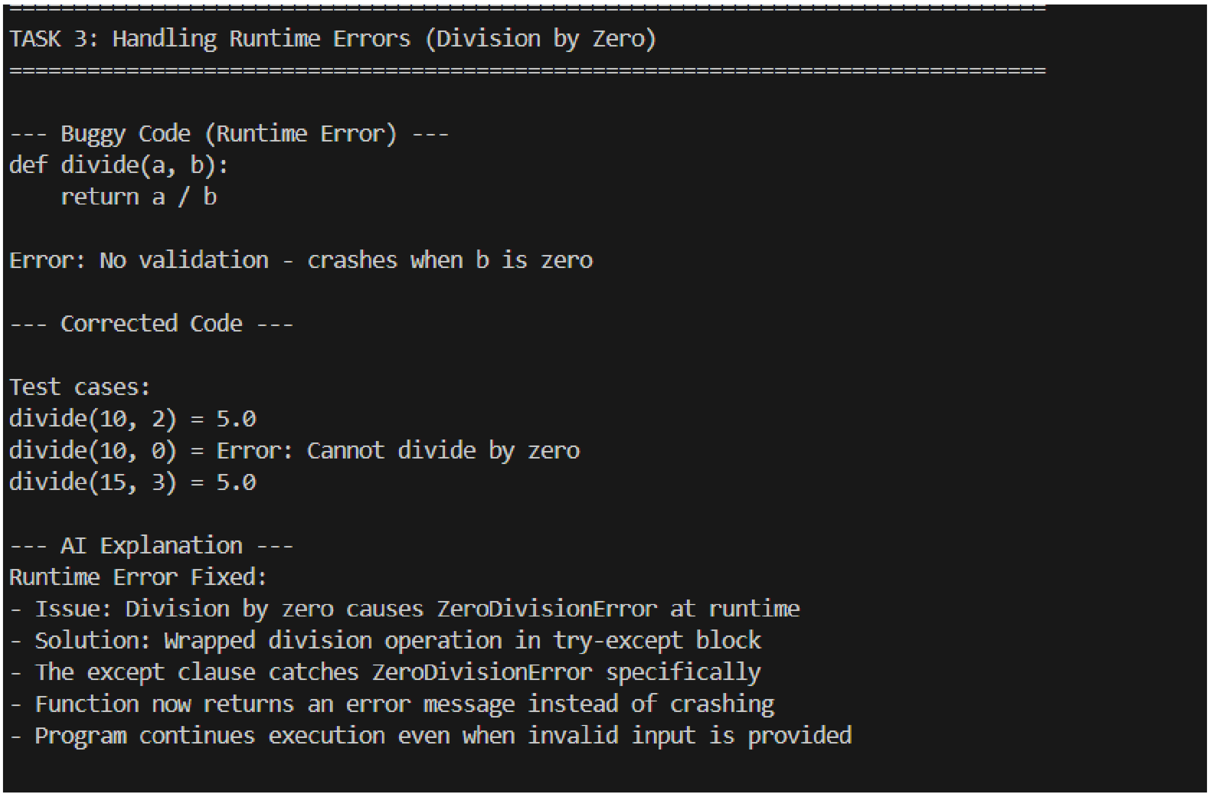


```
# ============================================================================
# Task 3: Handling Runtime Errors (Division by Zero)
# ============================================================================
print("\n" + "=" * 80)
print("TASK 3: Handling Runtime Errors (Division by Zero)")
print("=" * 80)

# AI Prompt: "This function crashes with division by zero. Can you help fix it:
# def divide(a, b):
#     return a / b
# Please add proper error handling using try-except blocks."

print("\n--- Buggy Code (Runtime Error) ---")
print("def divide(a, b):")
print("    return a / b")
print("\nError: No validation - crashes when b is zero")

print("\n--- Corrected Code ---")
# Corrected function with error handling
def divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        return "Error: Cannot divide by zero"
```



```
# Test cases
print("\nTest cases:")
print(f"divide(10, 2) = {divide(10, 2)}")
print(f"divide(10, 0) = {divide(10, 0)}")
print(f"divide(15, 3) = {divide(15, 3)}")

print("\n--- AI Explanation ---")
print("Runtime Error Fixed:")
print("- Issue: Division by zero causes ZeroDivisionError at runtime")
print("- Solution: Wrapped division operation in try-except block")
print("- The except clause catches ZeroDivisionError specifically")
print("- Function now returns an error message instead of crashing")
print("- Program continues execution even when invalid input is provided")
```

OUTPUT:



```
================================================================================
TASK 3: Handling Runtime Errors (Division by Zero)
================================================================================

--- Buggy Code (Runtime Error) ---
def divide(a, b):
    return a / b

Error: No validation - crashes when b is zero

--- Corrected Code ---

Test cases:
divide(10, 2) = 5.0
divide(10, 0) = Error: Cannot divide by zero
divide(15, 3) = 5.0

--- AI Explanation ---
Runtime Error Fixed:
- Issue: Division by zero causes ZeroDivisionError at runtime
- Solution: Wrapped division operation in try-except block
- The except clause catches ZeroDivisionError specifically
- Function now returns an error message instead of crashing
- Program continues execution even when invalid input is provided
```

Justification:

Division by zero causes a ZeroDivisionError during execution, which crashes the program. Using a try-except block catches this exception and prevents the program from stopping unexpectedly. This makes the function robust and capable of handling invalid input safely.

Task 4: Debugging Class Definition Errors

Scenario

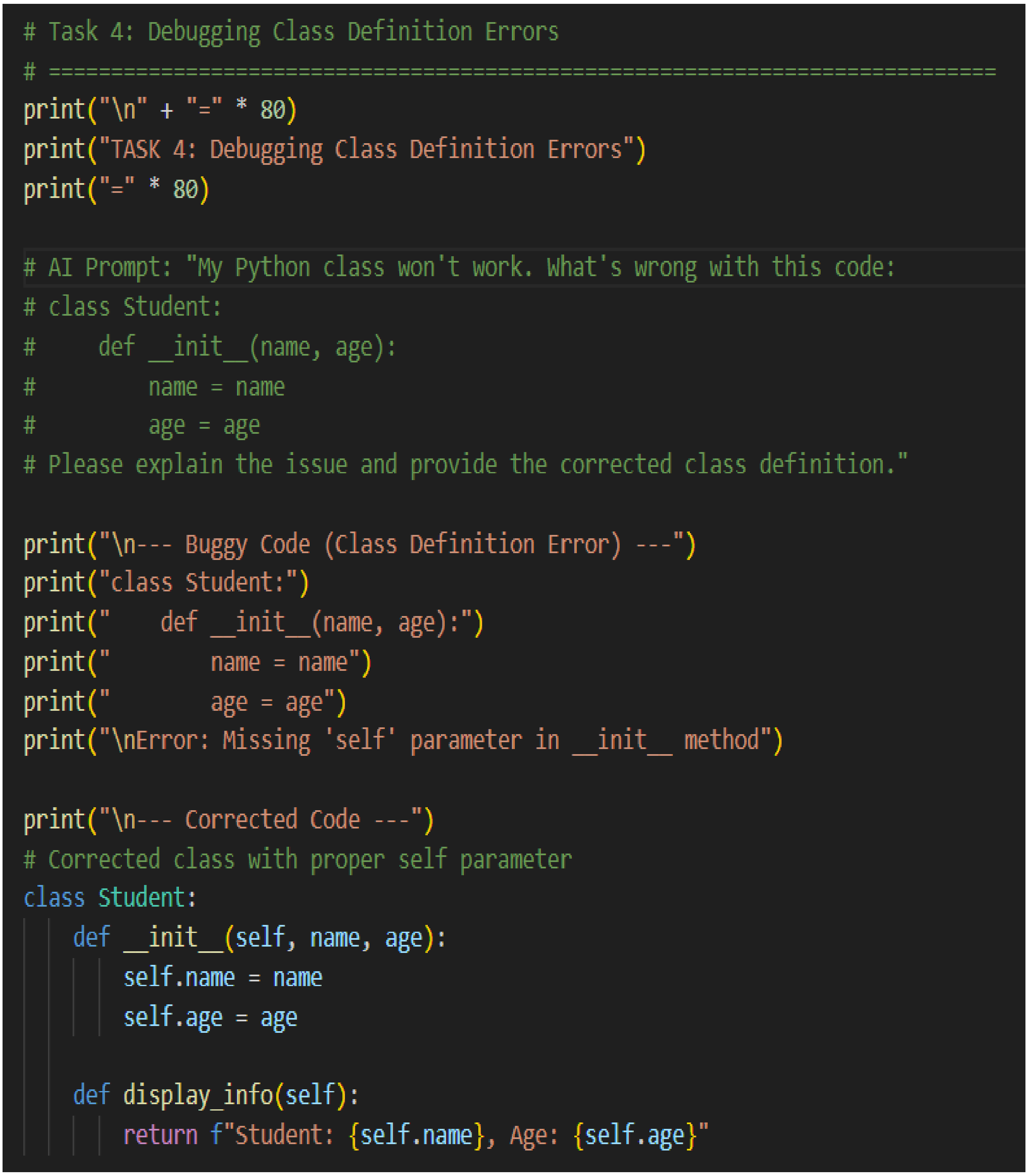
You are given a faulty Python class where the constructor is incorrectly defined.

Requirements

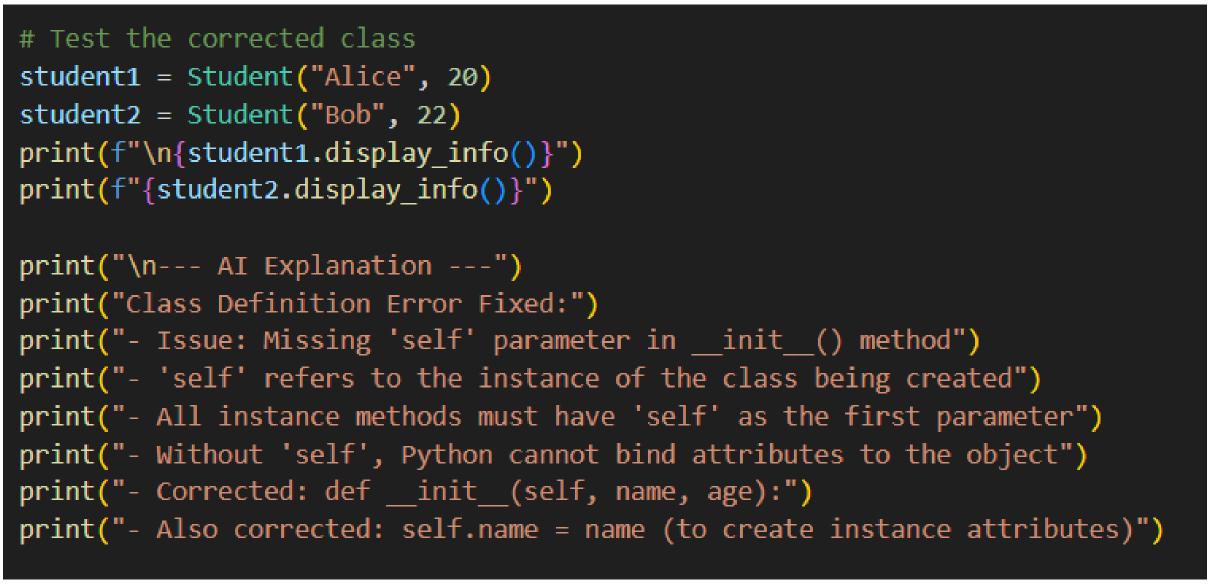
- Provide a class definition with missing self-parameter
- Use AI to identify the issue in the __init__() method
- Allow AI to correct the class definition
- Understand why self is required

Prompt: My Python class won't work. What's wrong with this code:

CODE:

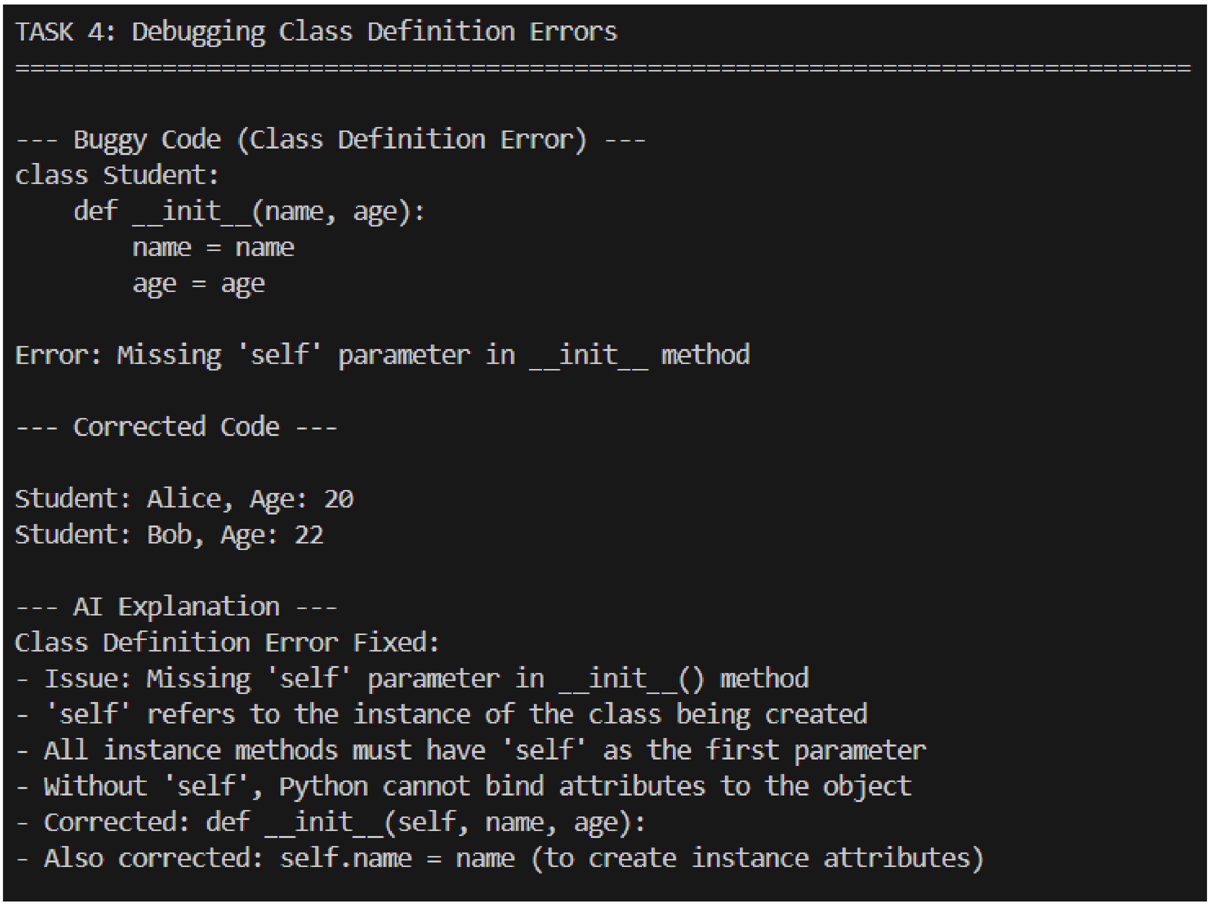


```python
# Task 4: Debugging Class Definition Errors
# ============================================================================
print("\n" + "=" * 80)
print("TASK 4: Debugging Class Definition Errors")
print("=" * 80)

# AI Prompt: "My Python class won't work. What's wrong with this code:
# class Student:
#     def __init__(name, age):
#         name = name
#         age = age
# Please explain the issue and provide the corrected class definition."

print("\n--- Buggy Code (Class Definition Error) ---")
print("class Student:")
print("    def __init__(name, age):")
print("        name = name")
print("        age = age")
print("\nError: Missing 'self' parameter in __init__ method")

print("\n--- Corrected Code ---")
# Corrected class with proper self parameter
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        return f"Student: {self.name}, Age: {self.age}"
```



```
# Test the corrected class
student1 = Student("Alice", 20)
student2 = Student("Bob", 22)
print(f"\n{student1.display_info()}")
print(f"{student2.display_info()}")

print("\n--- AI Explanation ---")
print("Class Definition Error Fixed:")
print("- Issue: Missing 'self' parameter in __init__() method")
print("- 'self' refers to the instance of the class being created")
print("- All instance methods must have 'self' as the first parameter")
print("- Without 'self', Python cannot bind attributes to the object")
print("- Corrected: def __init__(self, name, age):")
print("- Also corrected: self.name = name (to create instance attributes)")
```

OUTPUT:



```
TASK 4: Debugging Class Definition Errors
================================================================================

--- Buggy Code (Class Definition Error) ---
class Student:
    def __init__(name, age):
        name = name
        age = age

Error: Missing 'self' parameter in __init__ method

--- Corrected Code ---

Student: Alice, Age: 20
Student: Bob, Age: 22

--- AI Explanation ---
Class Definition Error Fixed:
- Issue: Missing 'self' parameter in __init__() method
- 'self' refers to the instance of the class being created
- All instance methods must have 'self' as the first parameter
- Without 'self', Python cannot bind attributes to the object
- Corrected: def __init__(self, name, age):
- Also corrected: self.name = name (to create instance attributes)
```

Justification:

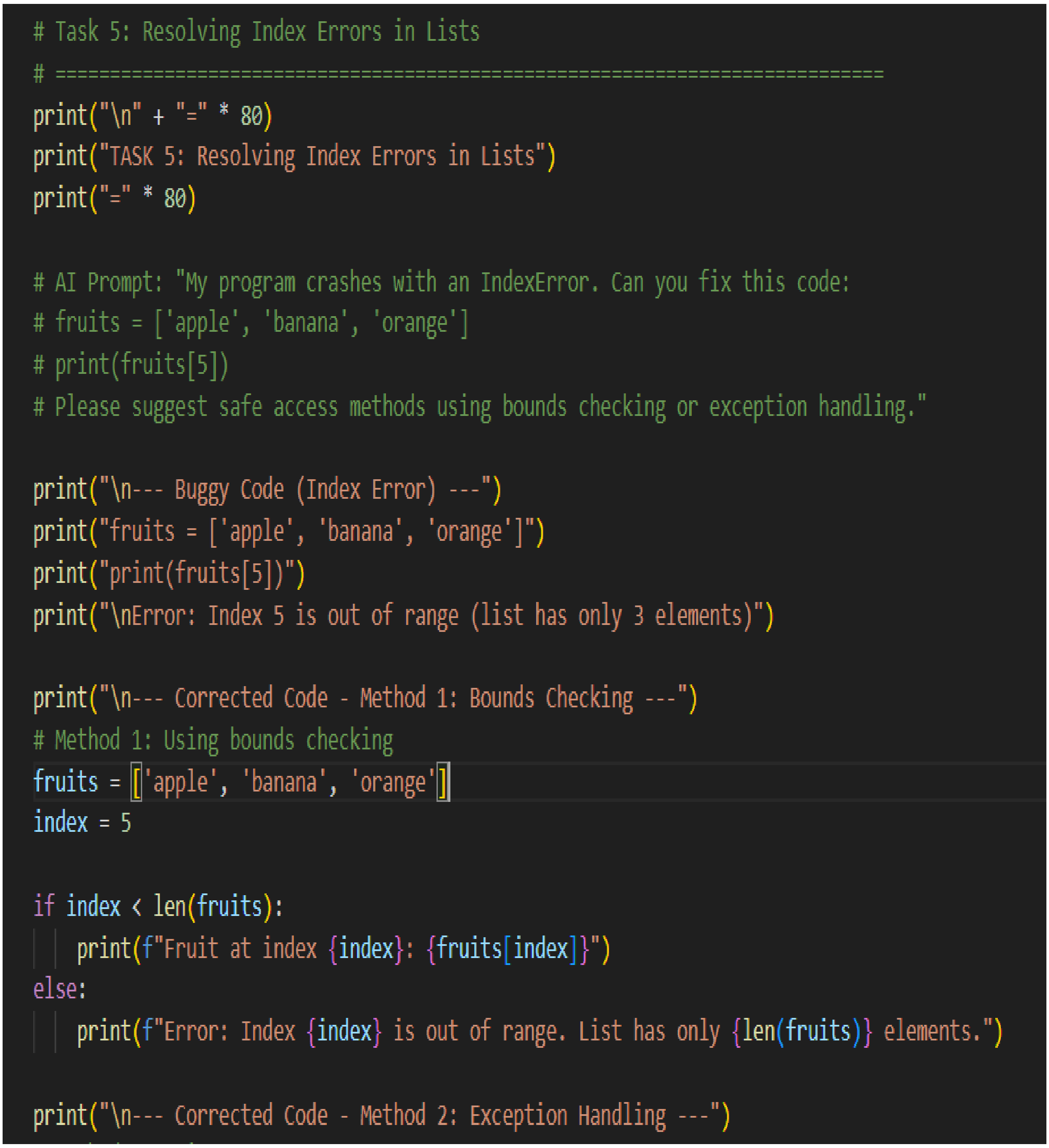
Instance methods in Python must include self as the first parameter to refer to the object being created. Without self, variables name and age are treated as local variables and not stored in the object. Adding self correctly binds attributes to the instance, enabling proper object behavior.

Task 5: Resolving Index Errors in Lists
Scenario
A program crashes when accessing an invalid index in a list.
Requirements

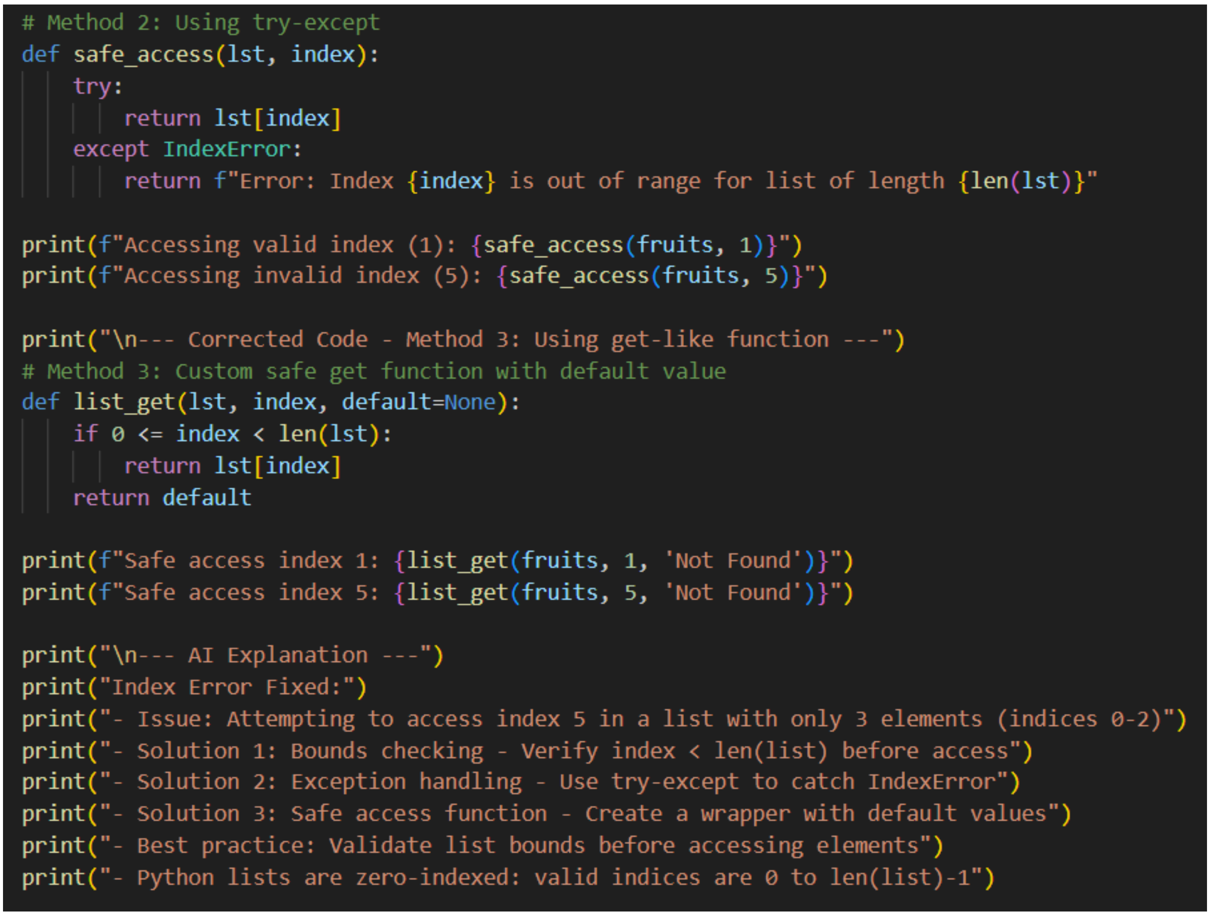
- Provide code that accesses an out-of-range list index
- Use AI to identify the Index Error
- Let AI suggest safe access methods
- Apply bounds checking or exception handling

Prompt: My program crashes with an IndexError. Can you fix this code:

CODE:

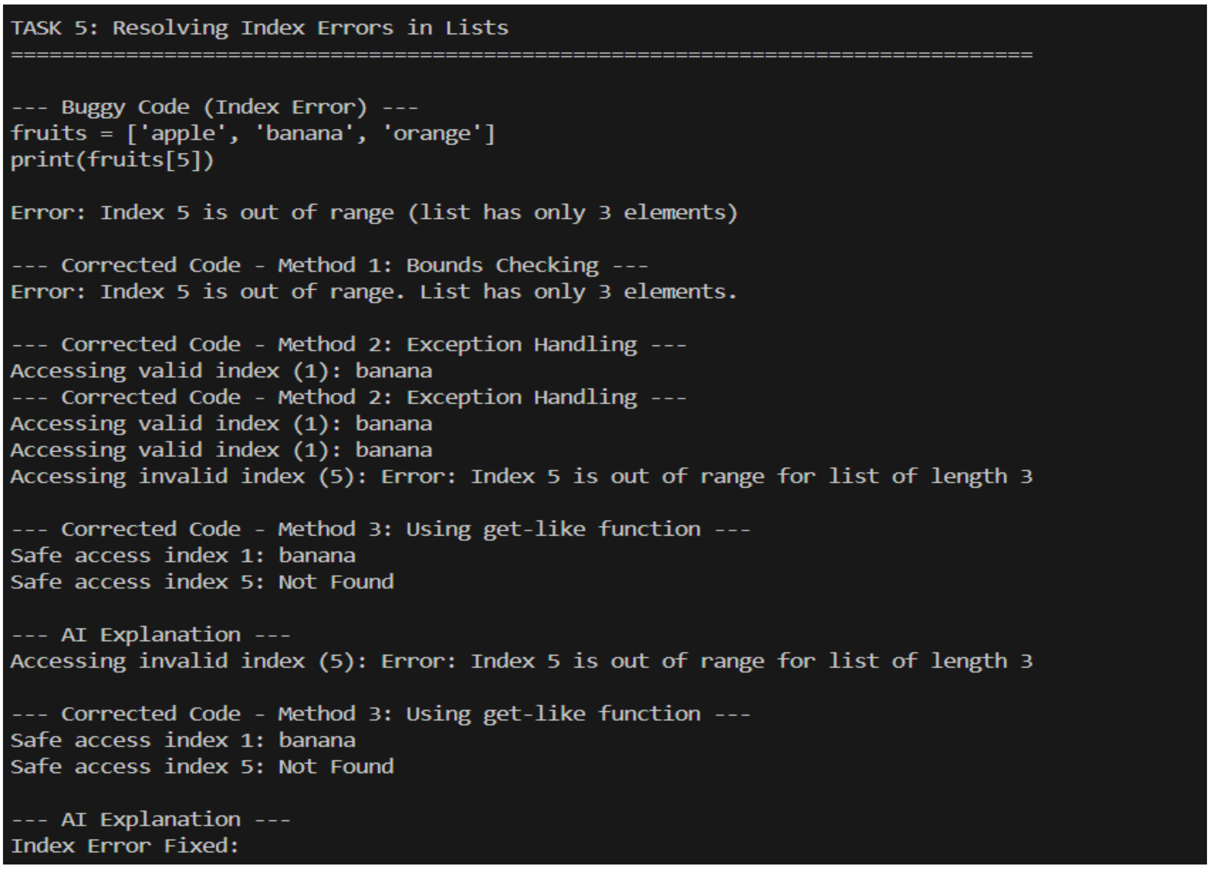


```
# Task 5: Resolving Index Errors in Lists
# ============================================================================
print("\n" + "=" * 80)
print("TASK 5: Resolving Index Errors in Lists")
print("=" * 80)

# AI Prompt: "My program crashes with an IndexError. Can you fix this code:
# fruits = ['apple', 'banana', 'orange']
# print(fruits[5])
# Please suggest safe access methods using bounds checking or exception handling."

print("\n--- Buggy Code (Index Error) ---")
print("fruits = ['apple', 'banana', 'orange']")
print("print(fruits[5])")
print("\nError: Index 5 is out of range (list has only 3 elements)")

print("\n--- Corrected Code - Method 1: Bounds Checking ---")
# Method 1: Using bounds checking
fruits = ['apple', 'banana', 'orange']
index = 5

if index < len(fruits):
    print(f"Fruit at index {index}: {fruits[index]}")
else:
    print(f"Error: Index {index} is out of range. List has only {len(fruits)} elements.")

print("\n--- Corrected Code - Method 2: Exception Handling ---")
```

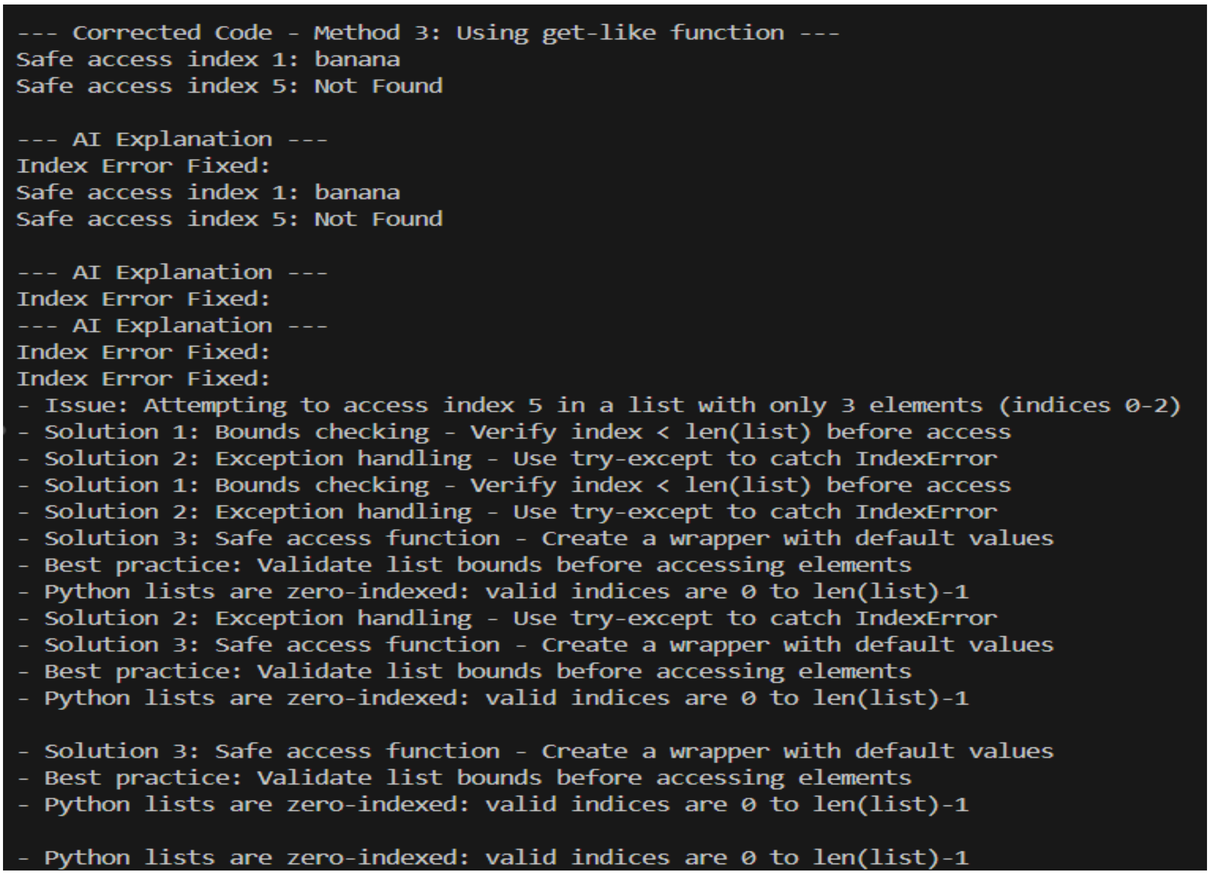


```python
# Method 2: Using try-except
def safe_access(lst, index):
    try:
        return lst[index]
    except IndexError:
        return f"Error: Index {index} is out of range for list of length {len(lst)}"

print(f"Accessing valid index (1): {safe_access(fruits, 1)}")
print(f"Accessing invalid index (5): {safe_access(fruits, 5)}")

print("\n--- Corrected Code - Method 3: Using get-like function ---")
# Method 3: Custom safe get function with default value
def list_get(lst, index, default=None):
    if 0 <= index < len(lst):
        return lst[index]
    return default

print(f"Safe access index 1: {list_get(fruits, 1, 'Not Found')}")
print(f"Safe access index 5: {list_get(fruits, 5, 'Not Found')}")

print("\n--- AI Explanation ---")
print("Index Error Fixed:")
print("- Issue: Attempting to access index 5 in a list with only 3 elements (indices 0-2)")
print("- Solution 1: Bounds checking - Verify index < len(list) before access")
print("- Solution 2: Exception handling - Use try-except to catch IndexError")
print("- Solution 3: Safe access function - Create a wrapper with default values")
print("- Best practice: Validate list bounds before accessing elements")
print("- Python lists are zero-indexed: valid indices are 0 to len(list)-1")
```

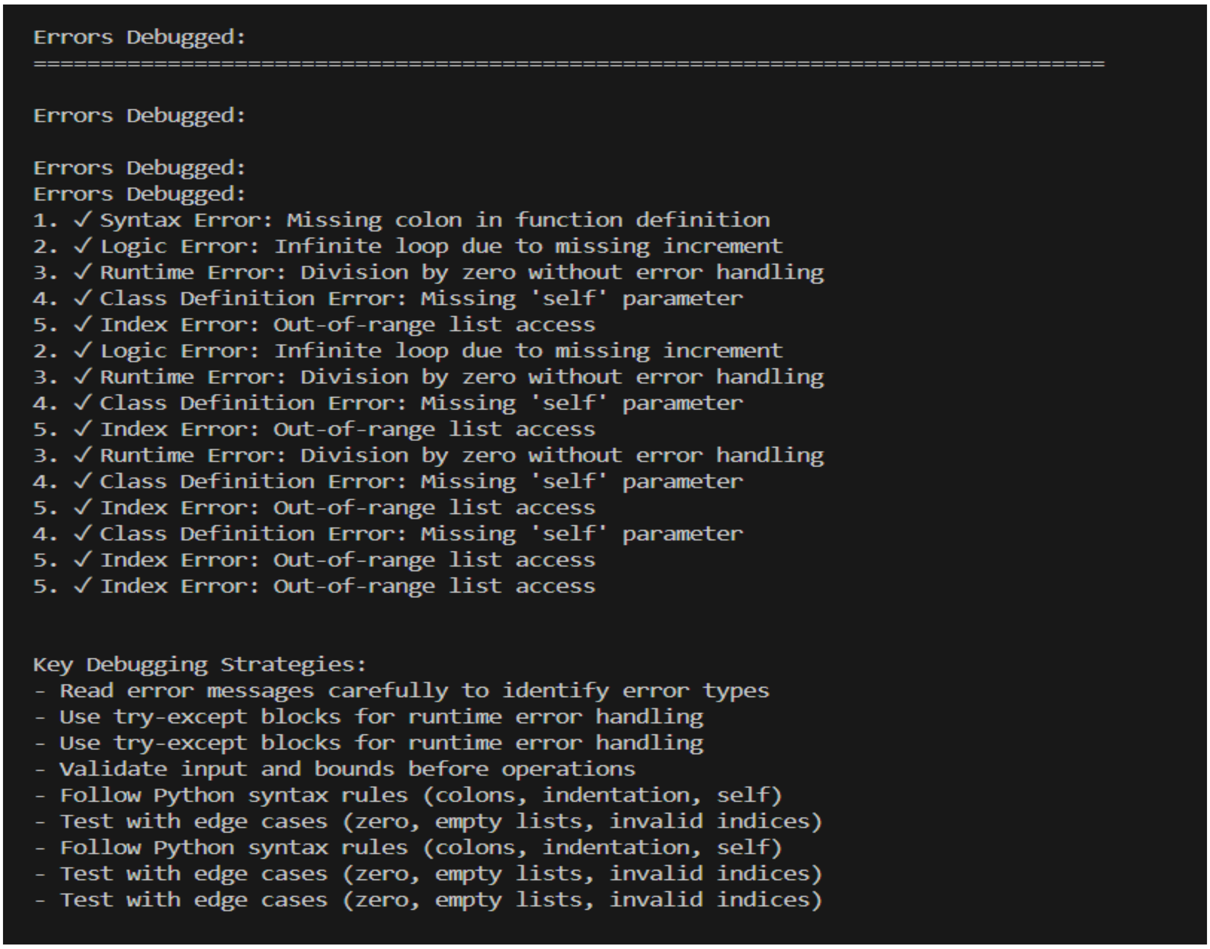
OUTPUT:



```
TASK 5: Resolving Index Errors in Lists
================================================================================

--- Buggy Code (Index Error) ---
fruits = ['apple', 'banana', 'orange']
print(fruits[5])

Error: Index 5 is out of range (list has only 3 elements)

--- Corrected Code - Method 1: Bounds Checking ---
Error: Index 5 is out of range. List has only 3 elements.

--- Corrected Code - Method 2: Exception Handling ---
Accessing valid index (1): banana
--- Corrected Code - Method 2: Exception Handling ---
Accessing valid index (1): banana
Accessing valid index (1): banana
Accessing invalid index (5): Error: Index 5 is out of range for list of length 3

--- Corrected Code - Method 3: Using get-like function ---
Safe access index 1: banana
Safe access index 5: Not Found

--- AI Explanation ---
Accessing invalid index (5): Error: Index 5 is out of range for list of length 3

--- Corrected Code - Method 3: Using get-like function ---
Safe access index 1: banana
Safe access index 5: Not Found

--- AI Explanation ---
Index Error Fixed:
```



```
--- Corrected Code - Method 3: Using get-like function ---
Safe access index 1: banana
Safe access index 5: Not Found

--- AI Explanation ---
Index Error Fixed:
Safe access index 1: banana
Safe access index 5: Not Found

--- AI Explanation ---
Index Error Fixed:
--- AI Explanation ---
Index Error Fixed:
Index Error Fixed:
- Issue: Attempting to access index 5 in a list with only 3 elements (indices 0-2)
- Solution 1: Bounds checking - Verify index < len(list) before access
- Solution 2: Exception handling - Use try-except to catch IndexError
- Solution 1: Bounds checking - Verify index < len(list) before access
- Solution 2: Exception handling - Use try-except to catch IndexError
- Solution 3: Safe access function - Create a wrapper with default values
- Best practice: Validate list bounds before accessing elements
- Python lists are zero-indexed: valid indices are 0 to len(list)-1
- Solution 2: Exception handling - Use try-except to catch IndexError
- Solution 3: Safe access function - Create a wrapper with default values
- Best practice: Validate list bounds before accessing elements
- Python lists are zero-indexed: valid indices are 0 to len(list)-1

- Solution 3: Safe access function - Create a wrapper with default values
- Best practice: Validate list bounds before accessing elements
- Python lists are zero-indexed: valid indices are 0 to len(list)-1

- Python lists are zero-indexed: valid indices are 0 to len(list)-1
```



```
 Errors Debugged:
 ============================================================================

 Errors Debugged:

 Errors Debugged:
 Errors Debugged:
 1. ✓ Syntax Error: Missing colon in function definition
 2. ✓ Logic Error: Infinite loop due to missing increment
 3. ✓ Runtime Error: Division by zero without error handling
 4. ✓ Class Definition Error: Missing 'self' parameter
 5. ✓ Index Error: Out-of-range list access
 2. ✓ Logic Error: Infinite loop due to missing increment
 3. ✓ Runtime Error: Division by zero without error handling
 4. ✓ Class Definition Error: Missing 'self' parameter
 5. ✓ Index Error: Out-of-range list access
 3. ✓ Runtime Error: Division by zero without error handling
 4. ✓ Class Definition Error: Missing 'self' parameter
 5. ✓ Index Error: Out-of-range list access
 4. ✓ Class Definition Error: Missing 'self' parameter
 5. ✓ Index Error: Out-of-range list access
 5. ✓ Index Error: Out-of-range list access


 Key Debugging Strategies:
 - Read error messages carefully to identify error types
 - Use try-except blocks for runtime error handling
 - Use try-except blocks for runtime error handling
 - Validate input and bounds before operations
 - Follow Python syntax rules (colons, indentation, self)
 - Test with edge cases (zero, empty lists, invalid indices)
 - Follow Python syntax rules (colons, indentation, self)
 - Test with edge cases (zero, empty lists, invalid indices)
 - Test with edge cases (zero, empty lists, invalid indices)
```

Justification:
The list had only three elements, but the code attempted to access index 5, which does not exist. Python raises an IndexError in such cases. Using bounds checking or exception handling ensures the program verifies index validity before access, preventing crashes and improving reliability.