Assignment-7.3

2303A51304

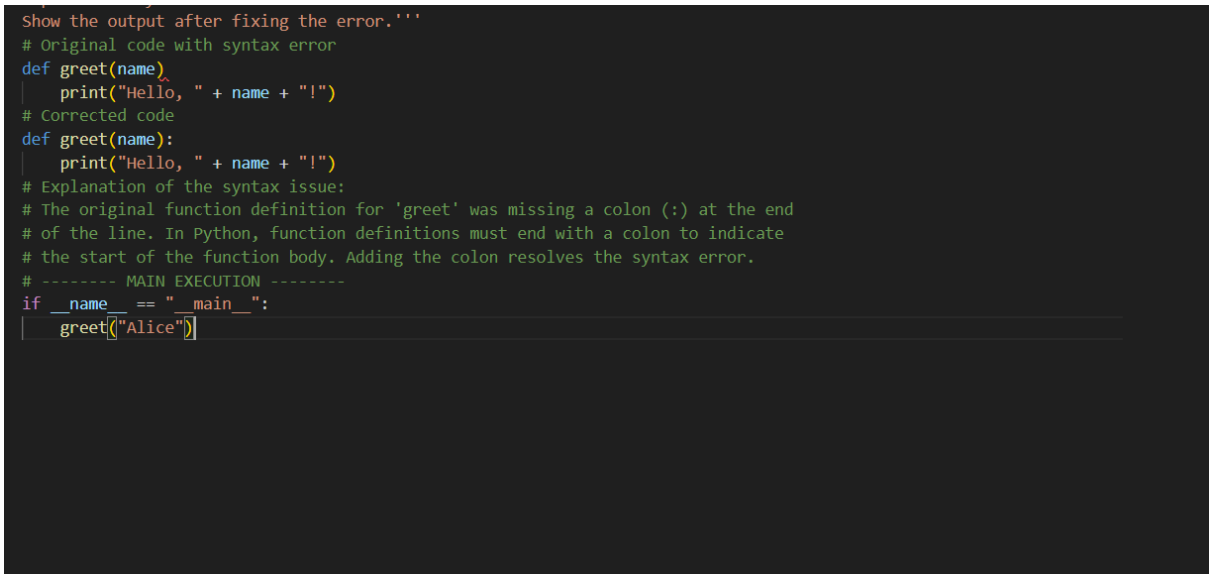
Batch-05

Task-01:

Prompt:

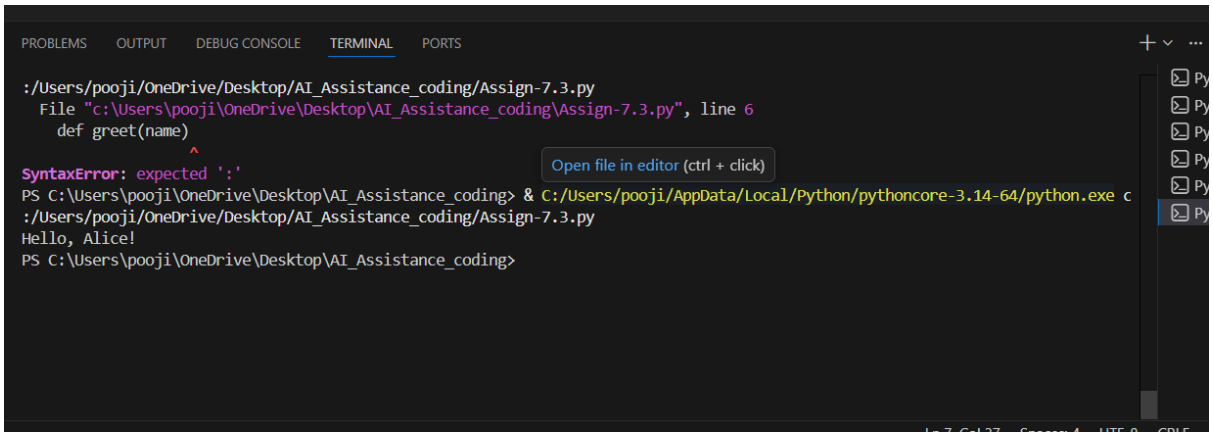
Detect and fix the syntax error in the following Python function where the function definition is missing a colon.
Correct the code so that it runs successfully.
Explain the syntax issue and how it was resolved.
Show the output after fixing the error.

Code:



```
Show the output after fixing the error.'''
# Original code with syntax error
def greet(name)
    print("Hello, " + name + "!")
# Corrected code
def greet(name):
    print("Hello, " + name + "!")
# Explanation of the syntax issue:
# The original function definition for 'greet' was missing a colon (:) at the end
# of the line. In Python, function definitions must end with a colon to indicate
# the start of the function body. Adding the colon resolves the syntax error.
# -------- MAIN EXECUTION --------
if __name__ == "__main__":
    greet("Alice")
```

Output:



Explanation:

The syntax error was caused by a missing colon in the function definition.
AI successfully identified and fixed the error.
The corrected function follows proper Python syntax.
The program runs without errors.
The output confirms correct functionality.

Task-02:

Prompt:

Identify and fix the logical error in the following Python loop that causes it to run infinitely.

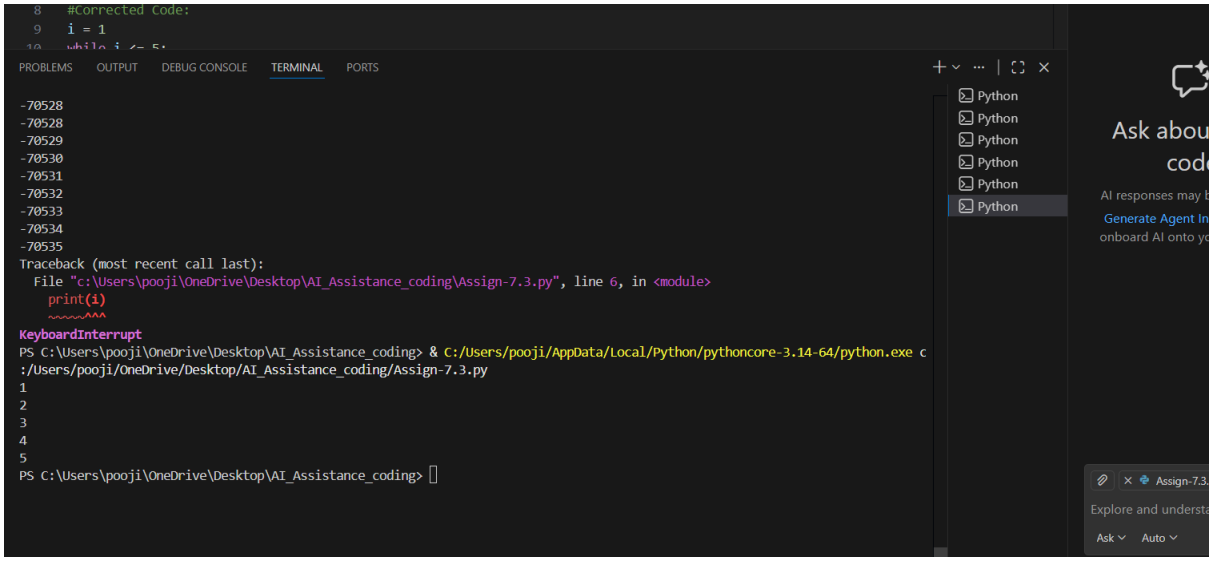
Explain why the infinite loop occurs and how the corrected logic resolves the issue.

Show the corrected code and its output.

Code:

```python
'''Identify and fix the logical error in the following Python loop that causes it to run infinitely.
Explain why the infinite loop occurs and how the corrected logic resolves the issue.
Show the corrected code and its output.'''
i = 1
while i <= 5:
    print(i)
    i -= 1   # ❌ Wrong direction (causes infinite loop)
#Corrected Code:
i = 1
while i <= 5:
    print(i)
    i += 1   # ✅ Correct direction
```

Output:



Explanation:

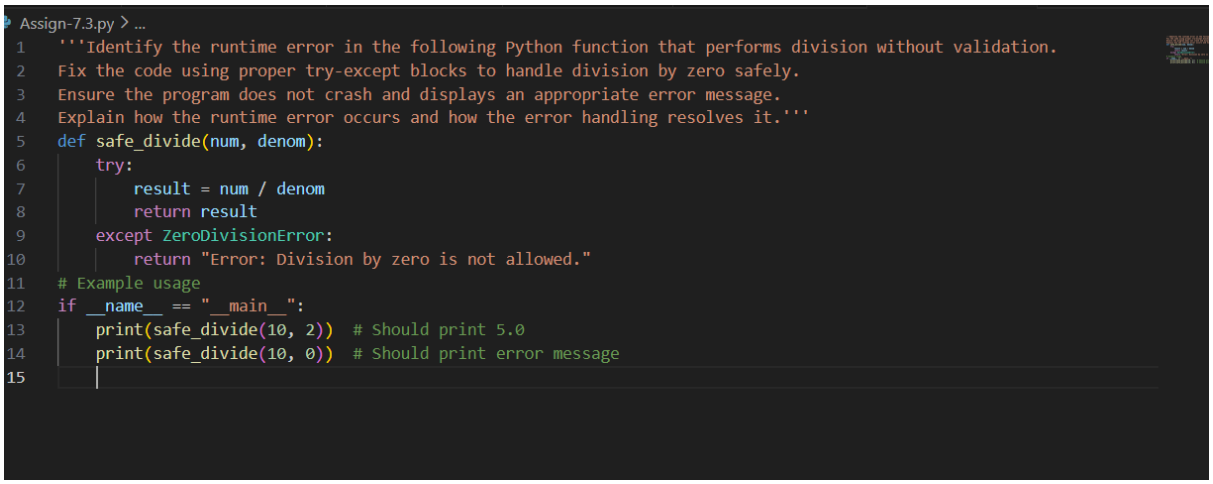
The infinite loop occurred due to an incorrect decrement operation.
AI detected the mismatch between the loop condition and update logic. Correcting the increment allowed the loop to terminate properly.
The loop now executes the expected number of times.
AI explanation clearly identifies and resolves the logical error.

Task-03:

Prompt:

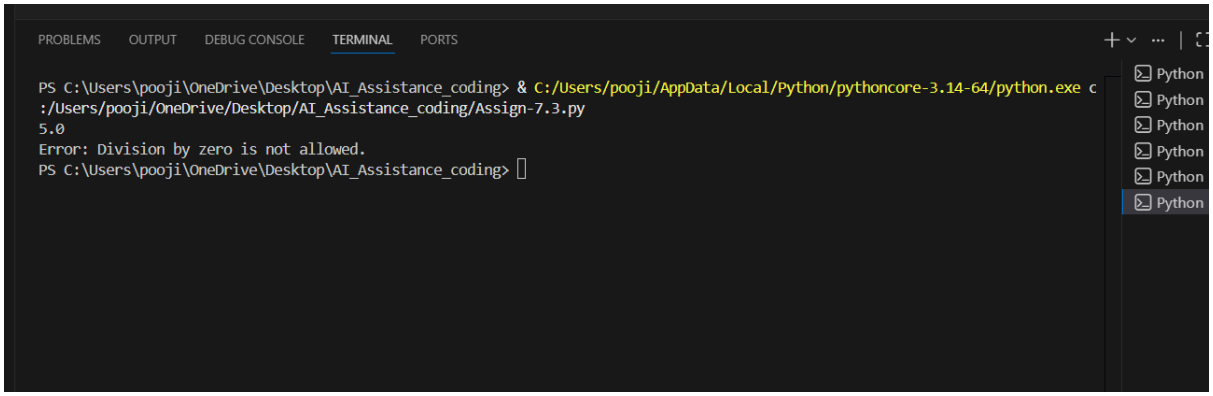
Identify the runtime error in the following Python function that performs division without validation. Fix the code using proper try-except blocks to handle division by zero safely. Ensure the program does not crash and displays an appropriate error message. Explain how the runtime error occurs and how the error handling resolves it.

Code:



```
Assign-7.3.py > ...
'''Identify the runtime error in the following Python function that performs division without validation.
Fix the code using proper try-except blocks to handle division by zero safely.
Ensure the program does not crash and displays an appropriate error message.
Explain how the runtime error occurs and how the error handling resolves it.'''
def safe_divide(num, denom):
    try:
        result = num / denom
        return result
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."
# Example usage
if __name__ == "__main__":
    print(safe_divide(10, 2))  # Should print 5.0
    print(safe_divide(10, 0))  # Should print error message
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/python.exe c
:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/Assign-7.3.py
5.0
Error: Division by zero is not allowed.
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>

Python
Python
Python
Python
Python
Python
```

Explanation:

Division by zero is a common runtime error in Python. AI identified the issue and added proper exception handling. The try-except block

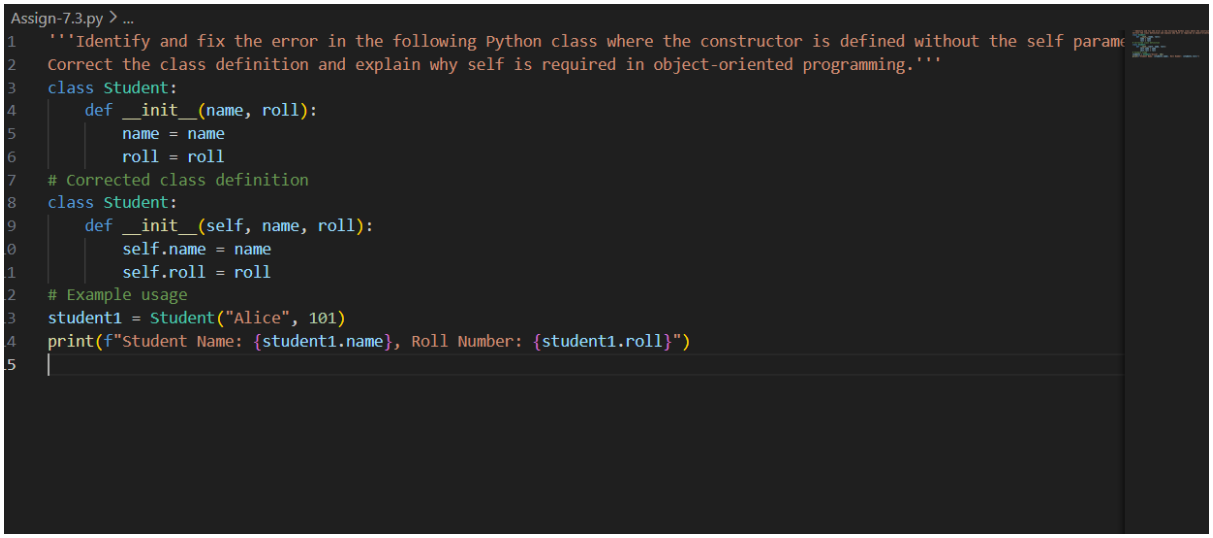
prevents program crashes. Safe execution is ensured with meaningful error messages. This improves program reliability and robustness.

Task-04:

Prompt:

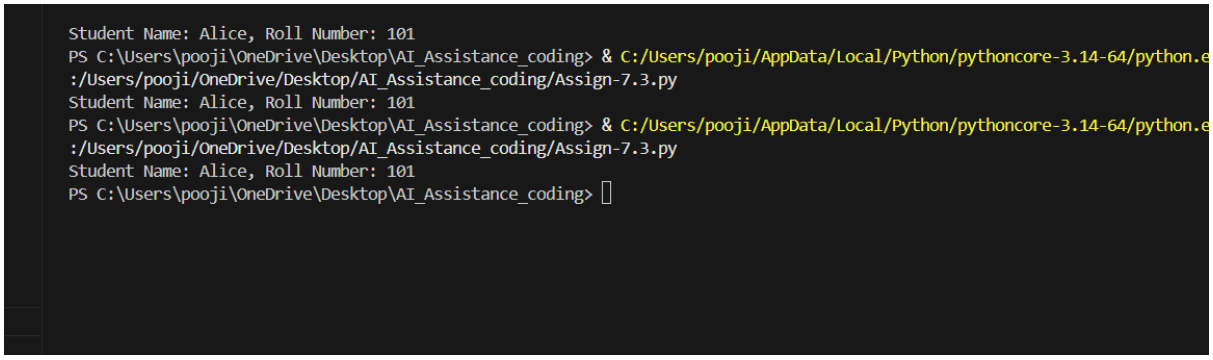
Identify and fix the error in the following Python class where the constructor is defined without the self parameter. Correct the class definition and explain why self is required in object-oriented programming.

Code:



```
Assign-7.3.py > ...
'''Identify and fix the error in the following Python class where the constructor is defined without the self parame
Correct the class definition and explain why self is required in object-oriented programming.'''
class Student:
    def __init__(name, roll):
        name = name
        roll = roll
# Corrected class definition
class Student:
    def __init__(self, name, roll):
        self.name = name
        self.roll = roll
# Example usage
student1 = Student("Alice", 101)
print(f"Student Name: {student1.name}, Roll Number: {student1.roll}")
```

Output:



```
Student Name: Alice, Roll Number: 101
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/python.e
:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/Assign-7.3.py
Student Name: Alice, Roll Number: 101
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding> & C:/Users/pooji/AppData/Local/Python/pythoncore-3.14-64/python.e
:/Users/pooji/OneDrive/Desktop/AI_Assistance_coding/Assign-7.3.py
Student Name: Alice, Roll Number: 101
PS C:\Users\pooji\OneDrive\Desktop\AI_Assistance_coding>
```

Explanation:

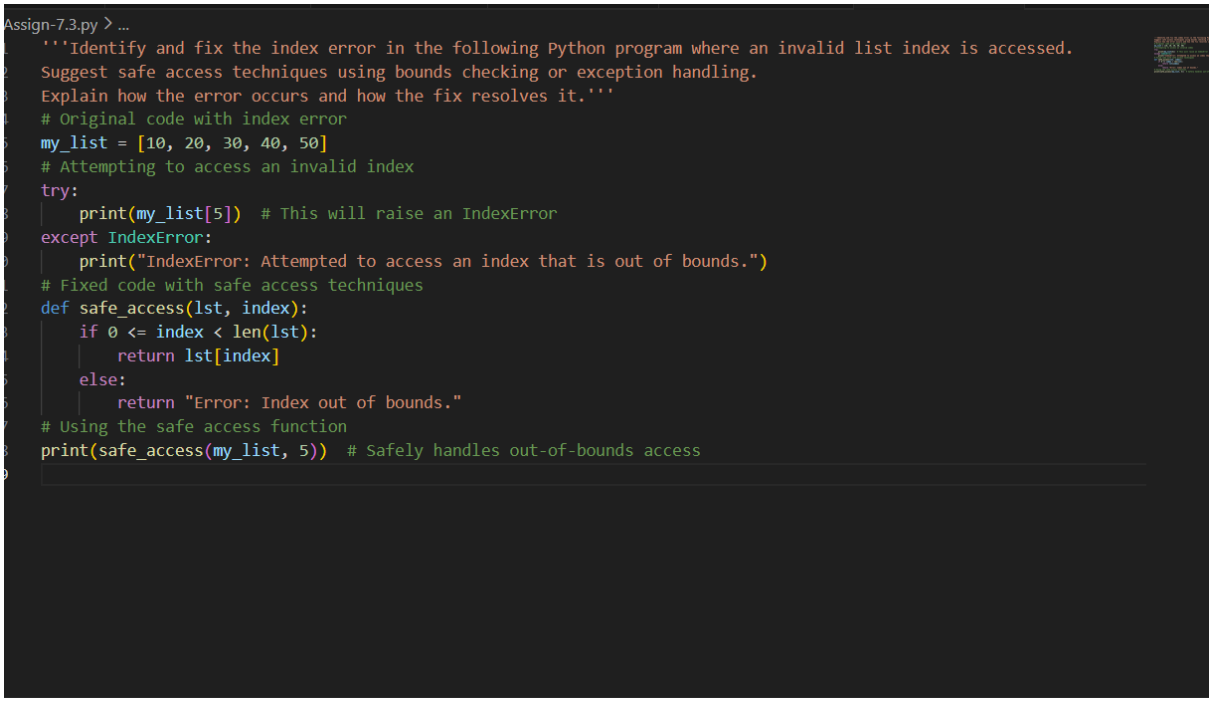
self refers to the current object instance. It allows each object to store its own data. Without self, variables become local, not object attributes. Python automatically passes the object reference as self. Using self ensures proper object-oriented behavior.

Task-05:

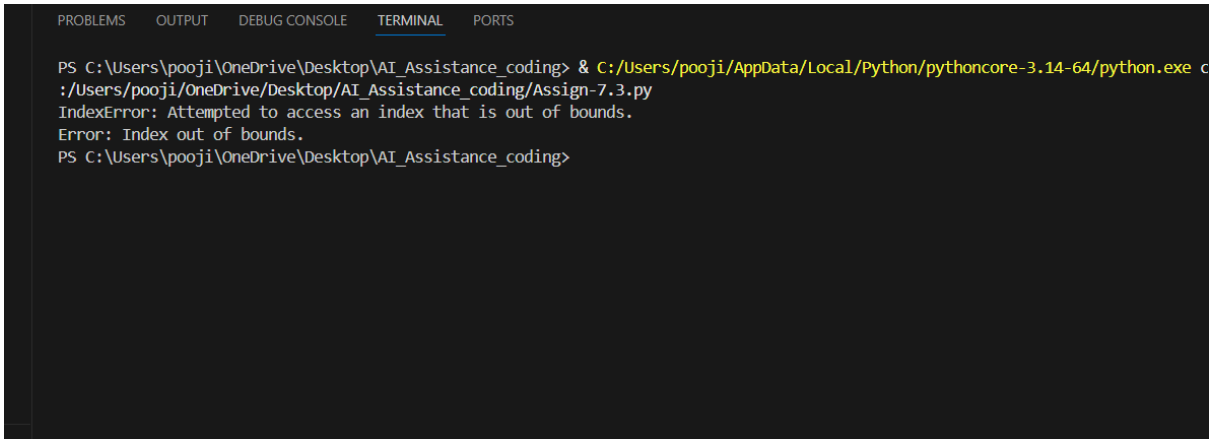
Prompt:

Identify and fix the index error in the following Python program where an invalid list index is accessed. Suggest safe access techniques using bounds checking or exception handling. Explain how the error occurs and how the fix resolves it.

Code:



Output:



Explanation:

IndexError occurs when accessing an invalid index. Length checking ensures index validity. try-except prevents program crashes. Both methods improve program robustness. Safe access is essential for reliable applications.