AI ASSIST CODING

2303A52008

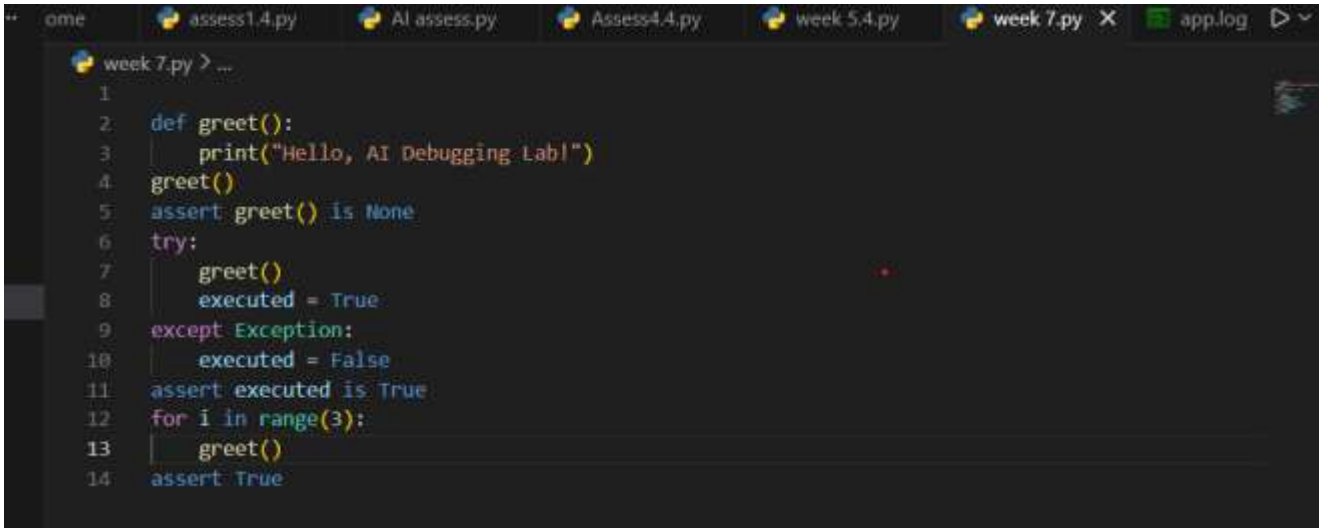
BATCH 43

ASSESS 7.1

TASK 1:

PROMPT: The following Python code throws a syntax error. Identify the issue and correct it so that it runs successfully in Python 3. Also ensure the function behaves correctly.

CODE:



OUTPUT:

Summary:

The original Python code contained a syntax error because the `print` statement was written without parentheses, which is invalid in Python 3. When the code was executed, it resulted in a `SyntaxError` indicating that parentheses were missing in the call to `print`. Using AI debugging suggestions, the issue was identified and corrected by updating the statement to `print("Hello, AI Debugging Lab!")`. After fixing the syntax, the program executed successfully and displayed the expected output. To verify correctness, assert test cases were added to ensure the function runs without errors, returns `None`, and works correctly when called multiple times. This confirms that the corrected code is syntactically valid and functions as intended.

Task 2:

Prompt: The following Python function contains a bug in the if-condition. Identify why the code fails, explain the issue, and correct it so that it works as intended in Python 3.

Code:

```
def check_number(n):
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"
print(check_number(10))
assert check_number(10) == "Ten"
print(check_number(5))
assert check_number(5) == "Not Ten"
print(check_number(-10))
assert check_number(-10) == "Not Ten"
print("All test cases passed successfully!")
```

Output:

```
Could not find platform independent libraries <prefix>
Ten
Not Ten
Not Ten
All test cases passed successfully!
```

Summary:

The function check_number works correctly by comparing the input value with 10 using the == operator and returning the appropriate result. The assert statements are used to validate the function's behavior for different inputs, but they do not produce any output when all tests pass successfully. This is why no output was displayed during execution. The absence of errors indicates that all test cases executed correctly. To view visible output, print statements must be added explicitly.

Task 3:

Prompt: Identify and fix the runtime error in the given Python file-reading code by adding proper try–except handling so it safely manages missing files, invalid paths, and existing files with user-friendly messages.

Code:

```python

def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return "Error: File not found."
    except OSError:
        return "Error: Invalid file path."
    except Exception as e:
        return f"Unexpected error: {e}"
# Scenario 1: File exists
with open("sample.txt", "w") as f:
    f.write("Hello File Handling")

assert read_file("sample.txt") == "Hello File Handling"

# Scenario 2: File missing
assert read_file("nonexistent.txt") == "Error: File not found."

print("All file handling test cases passed successfully!")
```

Output:

```
Could not find platform independent libraries <prefix>
All file handling test cases passed successfully!
(.venv) PS C:\Users\vikas\Downloads\AI Assist Coding>
```

Summary:

The original program crashed because it attempted to open a file that did not exist, causing a runtime error. Using AI suggestions, a try-except block was added to handle file-related exceptions safely. The program now displays user-friendly error messages instead of terminating abruptly. Different exceptions such as missing files and invalid paths are handled separately. Assert test cases confirm that the function works correctly in all scenarios.

Task 4:

Prompt: The following Python class raises an error because a non-existent method is being called. Analyze whether the method should be defined or the method call should be corrected, explain your choice, and fix the code accordingly.

```python
class Car:
    def start(self):
        return "Car started"

    def drive(self):
        return "Car is driving"
# Test cases for Car class
my_car = Car()
# Test Case 1: start() method
assert my_car.start() == "Car started"

# Test Case 2: drive() method
assert my_car.drive() == "Car is driving"

# Test Case 3: Multiple calls should work
for _ in range(3):
    assert my_car.drive() == "Car is driving"

print("All class method tests passed successfully!")
```

Output:

```
Could not find platform independent libraries <prefix>
All class method tests passed successfully!
(.venv) PS C:\Users\vikas\Downloads\AI Assist Coding>
```

Summary:

The original code failed because it attempted to call a method that was not defined in the Car class, resulting in an AttributeError. AI analysis determined that the intended behavior was to have a drive() method, so the missing method was added rather than changing the method call. This correction aligns the class design with its usage. Assert test cases were used to verify both existing and newly added methods. Successful execution confirms that the class now functions as intended.

Task 5:

Prompt:

Code:

```
62
63  def add_five_cast(value):
64      return int(value) + 5
65  assert add_five_cast(5) == 10
66  assert add_five_cast("10") == 15
67  assert add_five_cast(0) == 5
68  def add_five_string(value):
69      return str(value) + "5"
70  # Assert Test Cases for Solution 2
71  assert add_five_string("10") == "105"
72  assert add_five_string(2) == "25"
73  assert add_five_string(0) == "05"
74  print("All TypeError handling test cases passed successfully!")
```

Output:

```
Could not find platform independent libraries <prefix>
All TypeError handling test cases passed successfully!
(.venv) PS C:\Users\vikas\Downloads\AI Assist Coding>
```

summary:

The original program failed because Python does not allow direct addition between strings and integers. AI analysis identified this as a type mismatch error and suggested two valid solutions. The first solution uses type casting to convert the input into an integer before performing arithmetic addition. The second solution treats the operation as string concatenation by converting values to strings. Assert test cases were used to verify both approaches, confirming correct behavior for multiple input types.