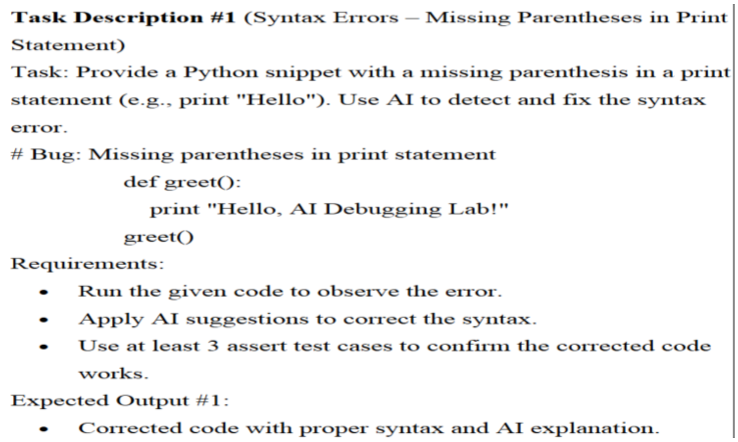
AI Assisted Coding

Name:D.udayan

Hall.no:2303A54017

Assignment: 7.1

Task1:



Task Description #1 (Syntax Errors – Missing Parentheses in Print Statement)

Task: Provide a Python snippet with a missing parenthesis in a print statement (e.g., print "Hello"). Use AI to detect and fix the syntax error.

```
# Bug: Missing parentheses in print statement
        def greet():
            print "Hello, AI Debugging Lab!"
        greet()
```

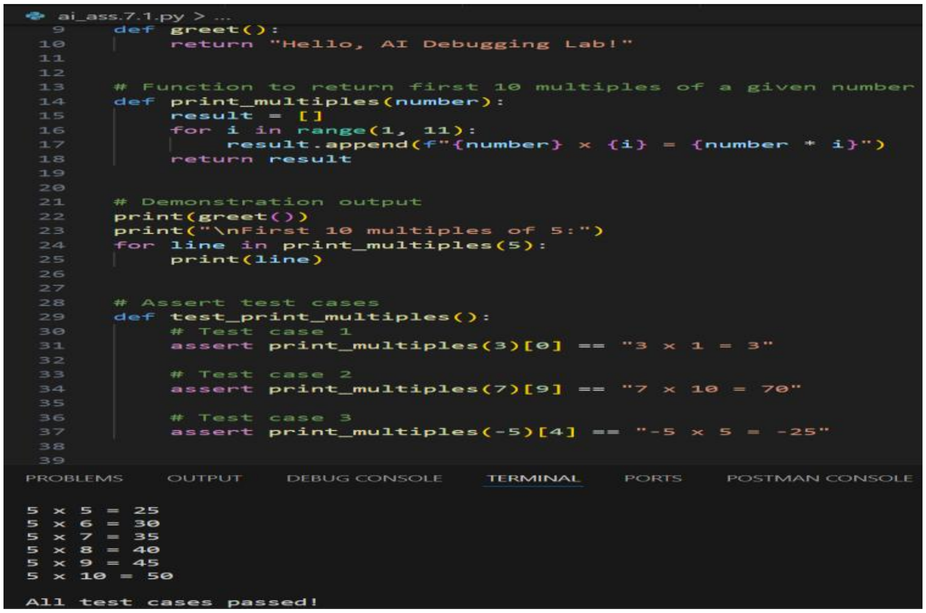
Requirements:

- Run the given code to observe the error.
- Apply AI suggestions to correct the syntax.
- Use at least 3 assert test cases to confirm the corrected code works.

Expected Output #1:

- Corrected code with proper syntax and AI explanation.

Ai generated code:



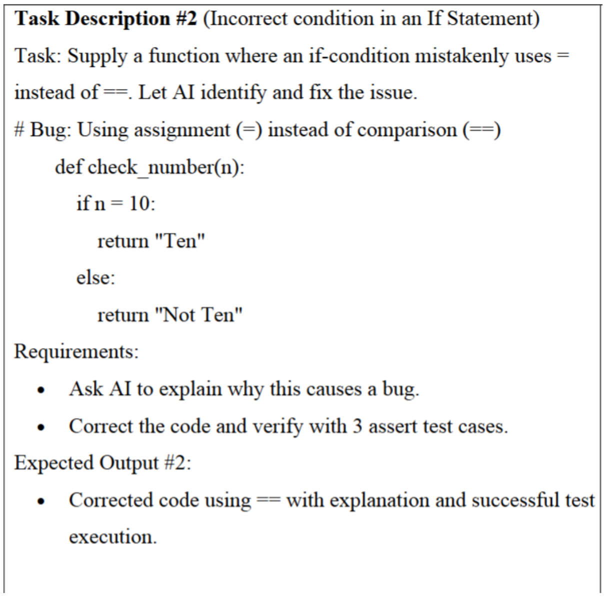
```python
def greet():
    return "Hello, AI Debugging Lab!"


# Function to return first 10 multiples of a given number
def print_multiples(number):
    result = []
    for i in range(1, 11):
        result.append(f"{number} x {i} = {number * i}")
    return result


# Demonstration output
print(greet())
print("\nFirst 10 multiples of 5:")
for line in print_multiples(5):
    print(line)


# Assert test cases
def test_print_multiples():
    # Test case 1
    assert print_multiples(3)[0] == "3 x 1 = 3"

    # Test case 2
    assert print_multiples(7)[9] == "7 x 10 = 70"

    # Test case 3
    assert print_multiples(-5)[4] == "-5 x 5 = -25"

```

```
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

All test cases passed!
```

Prompt:

```
#Create a simple Python program that contains a syntax error caused by missing
parentheses in a print statement. Run the code to observe the error, then use AI
assistance to identify and fix the syntax issue. After correcting the code, add
at least three assert test cases to verify that the function works correctly.
Keep the explanation clear and beginner-friendly, and ensure the final program
runs without errors.
```

Task2:



Task Description #2 (Incorrect condition in an If Statement)

Task: Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify and fix the issue.

```
# Bug: Using assignment (=) instead of comparison (==)
    def check_number(n):
        if n = 10:
            return "Ten"
        else:
            return "Not Ten"
```

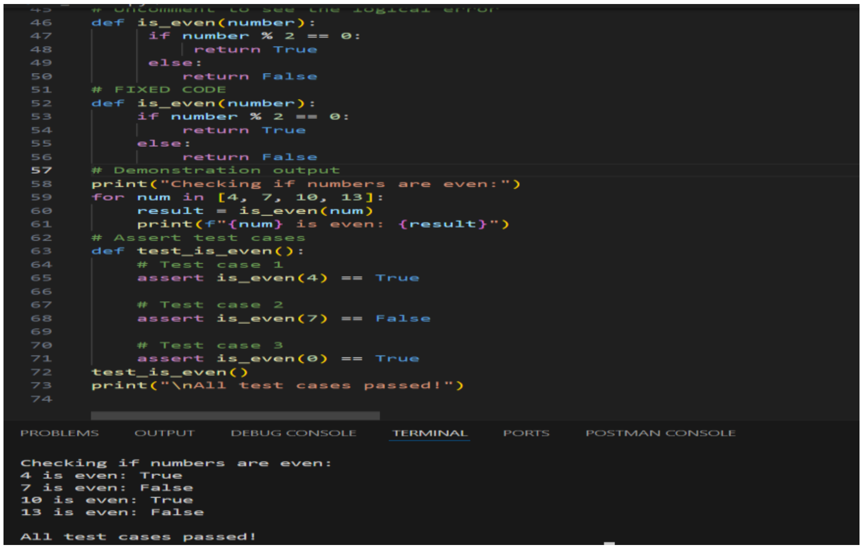
Requirements:

- Ask AI to explain why this causes a bug.
- Correct the code and verify with 3 assert test cases.

Expected Output #2:

- Corrected code using == with explanation and successful test execution.

Ai generated code:



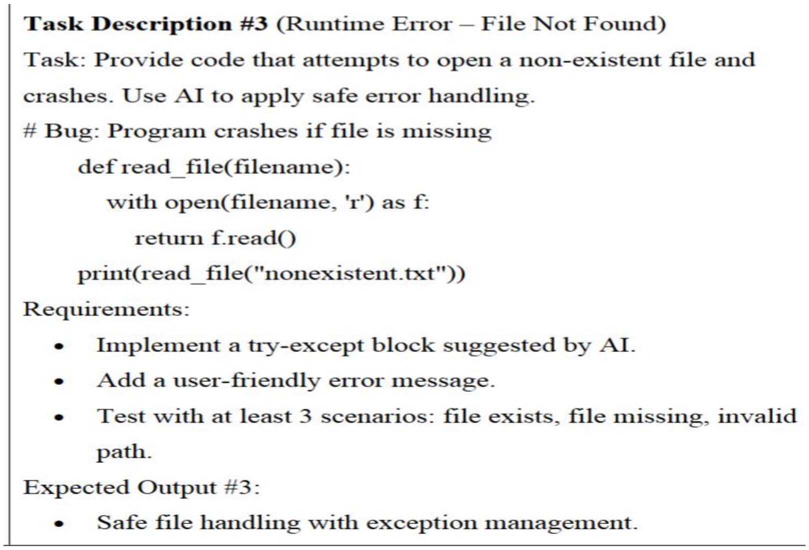
```
# Uncomment to see the logical error
def is_even(number):
     if number % 2 == 0:
         return True
     else:
        return False
# FIXED CODE
def is_even(number):
    if number % 2 == 0:
        return True
    else:
        return False
# Demonstration output
print("Checking if numbers are even:")
for num in [4, 7, 10, 13]:
    result = is_even(num)
    print(f"{num} is even: {result}")
# Assert test cases
def test_is_even():
    # Test case 1
    assert is_even(4) == True

    # Test case 2
    assert is_even(7) == False

    # Test case 3
    assert is_even(0) == True
test_is_even()
print("\nAll test cases passed!")
```

```
Checking if numbers are even:
4 is even: True
7 is even: False
10 is even: True
13 is even: False

All test cases passed!
```

Prompt:

```
#Create a simple Python function that contains a bug where an if condition
incorrectly uses the assignment operator (=) instead of the comparison operator
(==). Run the code to observe the error, then use AI assistance to identify and
explain why this causes a bug. Correct the condition using == and verify the fix
by adding at least three assert test cases. Keep the code and explanation clear,
beginner-friendly, and ensure the final program runs without errors
```

Task3:



Task Description #3 (Runtime Error – File Not Found)

Task: Provide code that attempts to open a non-existent file and crashes. Use AI to apply safe error handling.

```
# Bug: Program crashes if file is missing
    def read_file(filename):
        with open(filename, 'r') as f:
            return f.read()
    print(read_file("nonexistent.txt"))
```

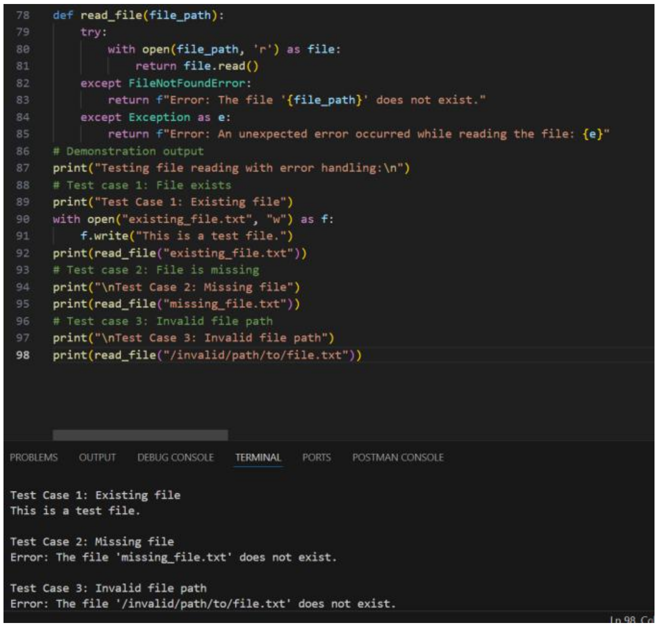
Requirements:

- Implement a try-except block suggested by AI.
- Add a user-friendly error message.
- Test with at least 3 scenarios: file exists, file missing, invalid path.

Expected Output #3:

- Safe file handling with exception management.

Ai generated code:



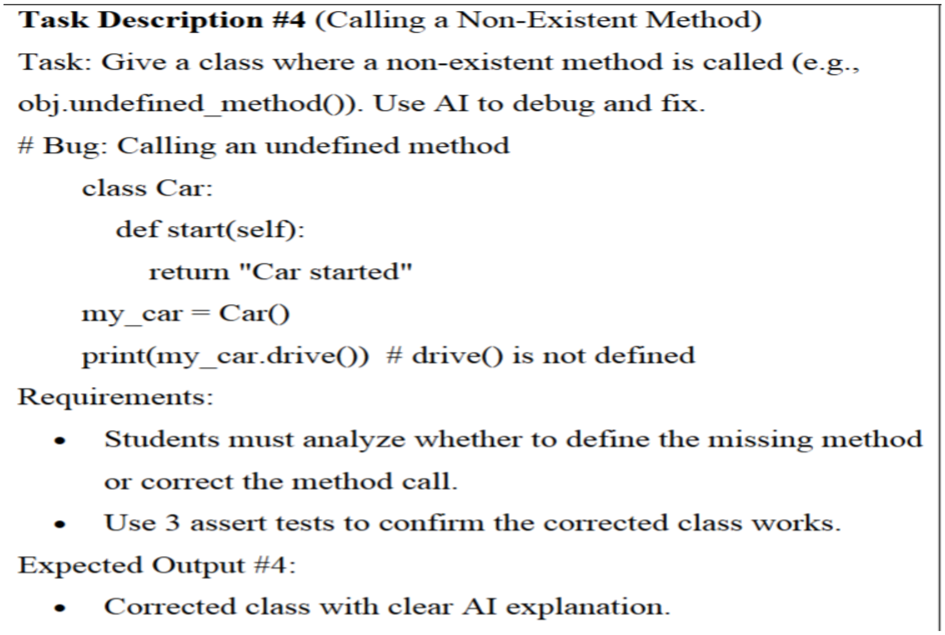
```python
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            return file.read()
    except FileNotFoundError:
        return f"Error: The file '{file_path}' does not exist."
    except Exception as e:
        return f"Error: An unexpected error occurred while reading the file: {e}"
# Demonstration output
print("Testing file reading with error handling:\n")
# Test case 1: File exists
print("Test Case 1: Existing file")
with open("existing_file.txt", "w") as f:
    f.write("This is a test file.")
print(read_file("existing_file.txt"))
# Test case 2: File is missing
print("\nTest Case 2: Missing file")
print(read_file("missing_file.txt"))
# Test case 3: Invalid file path
print("\nTest Case 3: Invalid file path")
print(read_file("/invalid/path/to/file.txt"))
```

```
Test Case 1: Existing file
This is a test file.

Test Case 2: Missing file
Error: The file 'missing_file.txt' does not exist.

Test Case 3: Invalid file path
Error: The file '/invalid/path/to/file.txt' does not exist.
```

Prompt:

```
#Create a Python program that attempts to open a file that does not exist,
causing a runtime error. Run the program to observe the crash, then use AI
assistance to apply safe error handling using a try-except block. Modify the code
to display a clear, user-friendly error message instead of crashing. Finally,
test the corrected program with at least three scenarios: when the file exists,
when the file is missing, and when an invalid file path is provided. Ensure the
final code runs safely without errors
```

Task4:



Task Description #4 (Calling a Non-Existent Method)

Task: Give a class where a non-existent method is called (e.g., obj.undefined_method()). Use AI to debug and fix.

```
# Bug: Calling an undefined method
    class Car:
        def start(self):
            return "Car started"
    my_car = Car()
    print(my_car.drive())  # drive() is not defined
```

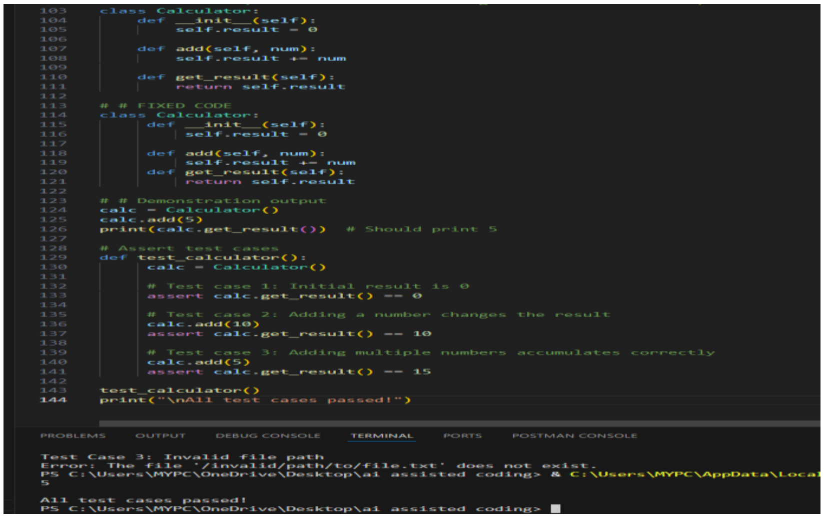
Requirements:

- Students must analyze whether to define the missing method or correct the method call.
- Use 3 assert tests to confirm the corrected class works.

Expected Output #4:

- Corrected class with clear AI explanation.

Ai generated code:



```
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result += num

    def get_result(self):
        return self.result

# # FIXED CODE
class Calculator:
     def __init__(self):
         self.result = 0

     def add(self, num):
         self.result += num
     def get_result(self):
         return self.result

# # Demonstration output
calc = Calculator()
calc.add(5)
print(calc.get_result())  # Should print 5

# Assert test cases
def test_calculator():
     calc = Calculator()

     # Test case 1: Initial result is 0
     assert calc.get_result() == 0

     # Test case 2: Adding a number changes the result
     calc.add(10)
     assert calc.get_result() == 10

     # Test case 3: Adding multiple numbers accumulates correctly
     calc.add(5)
     assert calc.get_result() == 15

test_calculator()
print("\nAll test cases passed!")
```

```
Test Case 3: Invalid file path
Error: The file '/invalid/path/to/file.txt' does not exist.
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding> & C:\Users\MYPC\AppData\Loca
5

All test cases passed!
PS C:\Users\MYPC\OneDrive\Desktop\ai assisted coding> █
```

Prompt:

```
#Create a simple Python program to calculate the sum of the first n natural
numbers. First, write a function named sum_to_n(n) that uses a for loop to
compute the sum. Then, implement an alternative solution using a while loop or a
mathematical formula. Display the result for sample inputs and keep the code
clear, beginner-friendly, and easy to understand. Ensure the program runs without
errors and clearly shows the output
```

Tack5:

Task Description #5 (TypeError – Mixing Strings and Integers in Addition)

Task: Provide code that adds an integer and string ("5" + 2) causing a TypeError. Use AI to resolve the bug.

```
# Bug: TypeError due to mixing string and integer
    def add_five(value):
      return value + 5
    print(add_five("10"))
```

Requirements:

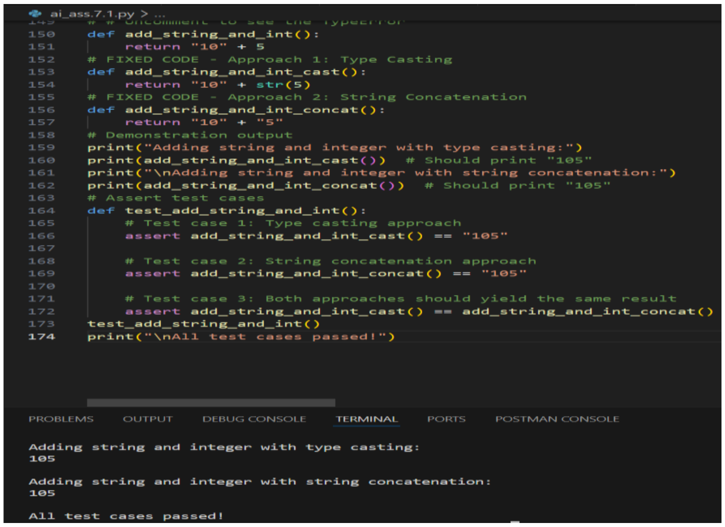
- Ask AI for two solutions: type casting and string concatenation.
- Validate with 3 assert test cases.

Expected Output #5:

- Corrected code that runs successfully for multiple inputs.

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

Ai generated code:



```
def add_string_and_int():
    return "10" + 5
# FIXED CODE - Approach 1: Type Casting
def add_string_and_int_cast():
    return "10" + str(5)
# FIXED CODE - Approach 2: String Concatenation
def add_string_and_int_concat():
    return "10" + "5"
# Demonstration output
print("Adding string and integer with type casting:")
print(add_string_and_int_cast())  # Should print "105"
print("\nAdding string and integer with string concatenation:")
print(add_string_and_int_concat())  # Should print "105"
# Assert test cases
def test_add_string_and_int():
    # Test case 1: Type casting approach
    assert add_string_and_int_cast() == "105"

    # Test case 2: String concatenation approach
    assert add_string_and_int_concat() == "105"

    # Test case 3: Both approaches should yield the same result
    assert add_string_and_int_cast() == add_string_and_int_concat()
test_add_string_and_int()
print("\nAll test cases passed!")
```

```
Adding string and integer with type casting:
105

Adding string and integer with string concatenation:
105

All test cases passed!
```

Prompt:

```
#Create a simple Python program that causes a TypeError by trying to add a string
and an integer (for example, "10" + 5). Run the code to observe the error, then
use AI assistance to identify and explain why the error occurs. Fix the problem
using two different approaches: first by applying type casting, and second by
using string concatenation. Verify both solutions by adding at least three assert
test cases. Keep the code clear, beginner-friendly, and ensure the final program
runs successfully for multiple inputs.
```