

AI Assisted Coding

Name: Saisuraj Telu

Hall.no: 2403A54L01

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:

```
ai_ass.7.1.py > ...
 9  def greet():
10     return "Hello, AI Debugging Lab!"
11
12
13 # Function to return first 10 multiples of a given number
14 def print_multiples(number):
15     result = []
16     for i in range(1, 11):
17         result.append(f"{number} x {i} = {number * i}")
18     return result
19
20
21 # Demonstration output
22 print(greet())
23 print("\nFirst 10 multiples of 5:")
24 for line in print_multiples(5):
25     print(line)
26
27
28 # Assert test cases
29 def test_print_multiples():
30     # Test case 1
31     assert print_multiples(3)[0] == "3 x 1 = 3"
32
33     # Test case 2
34     assert print_multiples(7)[9] == "7 x 10 = 70"
35
36     # Test case 3
37     assert print_multiples(-5)[4] == "-5 x 5 = -25"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
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

Prompt:

```
# UNCOMMENT TO SEE THE LOGICAL ERROR
46 def is_even(number):
47     if number % 2 == 0:
48         return True
49     else:
50         return False
51 # FIXED VERSION
52 def is_even(number):
53     if number % 2 == 0:
54         return True
55     else:
56         return False
57 # Demonstration output
58 print("Checking if numbers are even:")
59 for num in [4, 7, 10, 13]:
60     result = is_even(num)
61     print(f"{num} is even: {result}")
62 # Assert test cases
63 def test_is_even():
64     # Test case 1
65     assert is_even(4) == True
66
67     # Test case 2
68     assert is_even(7) == False
69
70     # Test case 3
71     assert is_even(10) == True
72 test_is_even()
73 print("\nAll test cases passed!")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
```

```
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!
```

```
#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:

Prompt:

The screenshot shows a code editor with a dark theme. A Python script is open, demonstrating file reading with error handling. The code includes three test cases: reading an existing file, reading a missing file, and reading an invalid file path. The terminal tab below the code shows the execution results for each case, displaying clear, user-friendly error messages.

```
78 def read_file(file_path):
79     try:
80         with open(file_path, 'r') as file:
81             return file.read()
82     except FileNotFoundError:
83         return f"Error: The file '{file_path}' does not exist."
84     except Exception as e:
85         return f"Error: An unexpected error occurred while reading the file: {e}"
86 # Demonstration output
87 print("Testing file reading with error handling:\n")
88 # Test case 1: File exists
89 print("Test Case 1: Existing file")
90 with open("existing_file.txt", "w") as f:
91     f.write("This is a test file.")
92 print(read_file("existing_file.txt"))
93 # Test case 2: File is missing
94 print("\nTest Case 2: Missing file")
95 print(read_file("missing_file.txt"))
96 # Test case 3: Invalid file path
97 print("\nTest Case 3: Invalid file path")
98 print(read_file("/invalid/path/to/file.txt"))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
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.
```

#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:

Ai generated code:

Prompt:

```
103     class Calculator:
104         def __init__(self):
105             self.result = 0
106
107         def add(self, num):
108             self.result += num
109
110         def get_result(self):
111             return self.result
112
113     # # FIXED CODE
114     class Calculator:
115         def __init__(self):
116             self.result = 0
117
118         def add(self, num):
119             self.result += num
120         def get_result(self):
121             return self.result
122
123     # # Demonstration output
124     calc = Calculator()
125     calc.add(5)
126     print(calc.get_result()) # Should print 5
127
128     # Assert test cases
129     def test_calculator():
130         calc = Calculator()
131
132         # Test case 1: Initial result is 0
133         assert calc.get_result() == 0
134
135         # Test case 2: Adding a number changes the result
136         calc.add(10)
137         assert calc.get_result() == 10
138
139         # Test case 3: Adding multiple numbers accumulates correctly
140         calc.add(5)
141         assert calc.get_result() == 15
142
143     test_calculator()
144     print("\nAll test cases passed!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

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\Local

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

.g.,

method

- Use 3 assert tests to confirm the corrected class works.

Expected Output #4:

- Corrected class with clear AI explanation.

```
#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:

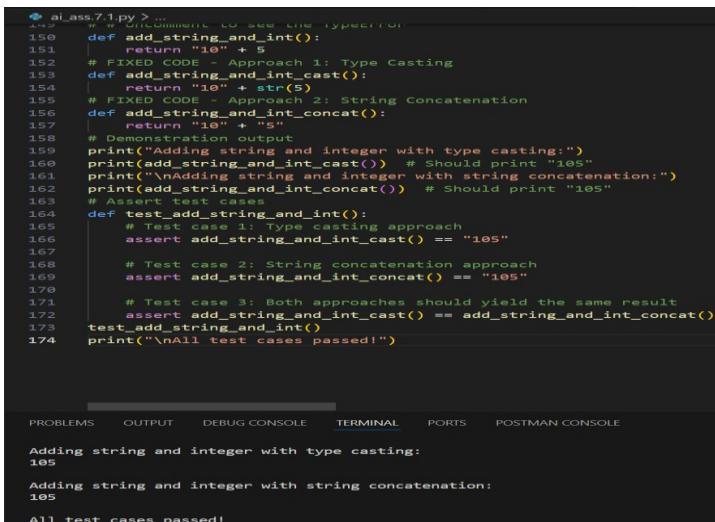
- Ask AI for two solutions: type casting and string concatenation.
- Validate with 3 assert test cases.

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

Prompt:



```
# ai_class_7_1.py >
# w/o comments to see the typeerror
150     def add_string_and_int():
151         return "10" + 5
152     # FIXED CODE - Approach 1: Type Casting
153     def add_string_and_int_cast():
154         return "10" + str(5)
155     # FIXED CODE - Approach 2: String Concatenation
156     def add_string_and_int_concat():
157         return "10" + "5"
158     # Demonstration output
159     print("Adding string and integer with type casting:")
160     print(add_string_and_int()) # Should print "105"
161     print("\nAdding string and integer with string concatenation:")
162     print(add_string_and_int_concat()) # Should print "105"
163     # Assert test cases
164     def test_add_string_and_int():
165         # Test case 1: Type casting approach
166         assert add_string_and_int() == "105"
167
168         # Test case 2: String concatenation approach
169         assert add_string_and_int_concat() == "105"
170
171         # Test case 3: Both approaches should yield the same result
172         assert add_string_and_int_cast() == add_string_and_int_concat()
173     test_add_string_and_int()
174     print("\nAll test cases passed!")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
Adding string and integer with type casting:
105
Adding string and integer with string concatenation:
105
All test cases passed!
```

Ai generated code

```
#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.
```