ASSIGNMENT :- 7.1

BATCH NO :- 13

DATE :- 15.09.2025

HALLTICKET NUMBER :- 2403A51317

TASK 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.

CODE :-

```python
import io
import sys
import unittest.mock

# Corrected function
def greet():
    print("Hello, AI Debugging Lab!")

def test_greet_output():
    # Test case 1: Ensure the output is correct
    with unittest.mock.patch('sys.stdout', new=io.StringIO()) as fake_out:
        greet()
        assert fake_out.getvalue().strip() == "Hello, AI Debugging Lab!", "Test 1 F

    # Test case 2: Ensure the function doesn't return anything
    assert greet() is None, "Test 2 Failed: Function should not return a value"

    # Test case 3: Ensure no extra characters are printed
    with unittest.mock.patch('sys.stdout', new=io.StringIO()) as fake_out:
        greet()
        # The .strip() is used to remove leading/trailing whitespace, including new
        assert len(fake_out.getvalue().strip()) == len("Hello, AI Debugging Lab!")

# Run the tests
test_greet_output()
print("\nAll tests passed successfully!")
```

OUTPUT :-

Actual Output:

```css
All tests passed successfully!
```

CODE ANALYSIS:-

Test 1 (Correct Output): We use unittest.mock.patch to capture what print sends to the console. The output is stored in fake_out, and we then assert that the captured string is exactly "Hello, AI Debugging Lab!". The .strip() method is used to remove any newline characters (\n) that print adds by default.

Test 2 (Function Return Value): We call greet() and assert that its return value is None. This is important because the print function itself doesn't return a value; it only has a side effect of displaying text.

Test 3 (Output Length): This test is a more robust check to ensure that the output is not only the correct string but also has the correct length, confirming there aren't any hidden characters or extra spaces.

TASK 2 :-

(Logic Error – 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.

CODE:-

```
# Add assert test cases to confirm the corrected code works.
assert check_number(10) == "Ten", "Test Case 1 Failed: Should return 'Ten' for 10"
assert check_number(5) == "Not Ten", "Test Case 2 Failed: Should return 'Not Ten' for 5"
assert check_number(15) == "Not Ten", "Test Case 3 Failed: Should return 'Not Ten' for 15"

print("All test cases passed!")
```

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

OUTPUT:-

Output of the Corrected Code

The corrected code runs without any errors because all the `assert` statements evaluate to `True`.

Output:

```
All test cases passed!
```

CODE ANLYSIS:-

Here is a Python function with the specified logic error.

You are getting a SyntaxError because in Python, the single equals sign (=) is used for assignment, while the double equals sign (==) is used for comparison. In an if statement, you need to compare the value of n to 10, so you should use ==. I will correct the code for you.

I have corrected the code by changing = to ==. Now, I will add three assert test cases to verify that the check_number function works correctly.

I have successfully explained the bug, corrected the code, and verified it with three assert test cases. The task is now complete.

TASK 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.

CODE :-

```
import os

def read_file_safe(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found. Please check the file
    except Exception as e:
        return f"An unexpected error occurred: {e}"

# --- Testing Scenarios ---

# Scenario 1: File exists
# Create a dummy file for testing
with open("testfile.txt", "w") as f:
    f.write("Hello, this is a test.")

print("--- Scenario 1: File exists ---")
output1 = read_file_safe("testfile.txt")
print(output1)

# Scenario 2: File is missing
print("\n--- Scenario 2: File is missing ---")
output2 = read_file_safe("nonexistent.txt")
print(output2)
```

```
# Scenario 3: Invalid path
# An invalid path can be an empty string or a path to a directory
print("\n--- Scenario 3: Invalid path ---")
output3 = read_file_safe("")
print(output3)

# Clean up the dummy file
os.remove("testfile.txt")
```

OUTPUT:-

```
--- Scenario 1: File exists ---
Hello, this is a test.

--- Scenario 2: File is missing ---
Error: The file 'nonexistent.txt' was not found. Please check the file name and pat

--- Scenario 3: Invalid path ---
Error: The file '' was not found. Please check the file name and path.
```

CODE ANALYSIS

The try-except block is a crucial tool for **robust programming**. It allows you to gracefully handle errors without the program terminating.

- **try block**: Attempts to open and read the file. If successful, it returns the file's content.
- **except FileNotFoundError**: If the file is not found, the program jumps to this block instead of crashing. It returns a user-friendly error message, guiding the user on how to resolve the issue.
- **except Exception as e**: This is a broader exception handler that catches any other potential errors (like an empty filename or a permission issue) and provides a more general error message.

TASK 4 :-

(AttributeError – 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.

CODE:-

```
# Fixed: Calling a defined method
class Car:
    def start(self):
        return "Car started"

my_car = Car()

# Assert tests to confirm the corrected class works
assert my_car.start() == "Car started", "Test 1 Failed: Expected 'Car started' fro
assert isinstance(my_car, Car), "Test 2 Failed: Expected my_car to be an instance
assert hasattr(my_car, 'start'), "Test 3 Failed: Expected my_car to have a 'start'

print(my_car.start())
print("\nAll tests passed successfully!")
```

OUTPUT:-

```
Car started

All tests passed successfully!
```

CODE ANALYSIS:-

1. **Test 1 (assert my_car.start() == "Car started"):** This is the primary test. It calls the corrected method start() and verifies that its return value is the expected string, "Car started". This confirms the method works as intended.
2. **Test 2 (assert isinstance(my_car, Car)):** This test checks that my_car is indeed an instance of the Car class. This is a good practice to ensure the object was instantiated correctly before testing its methods.
3. **Test 3 (assert hasattr(my_car, 'start')):** This test uses the built-in hasattr() function to check if the my_carobject has an attribute (in this case, a method) named start. This confirms that the method exists, which is the root of the original error.

TASK 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.

CODE:-

```python
def add_five_with_casting(value):
    # Cast the input 'value' from a string to an integer
    return int(value) + 5

# --- Test Cases ---
assert add_five_with_casting("10") == 15, "Test 1 Failed: Expected 15"
assert add_five_with_casting("0") == 5, "Test 2 Failed: Expected 5"
assert add_five_with_casting("-5") == 0, "Test 3 Failed: Expected 0"

print(add_five_with_casting("10"))
print("\nSolution 1: All tests passed successfully!")
```

EXPANATION:-

The int() function converts the string input "10" into the integer 10. The operation then becomes 10 + 5, which correctly returns 15. This is the most common and correct way to fix this type of error because it aligns with the function's name and probable purpose.

```python
def concatenate_five(value):
    # Convert the number 5 to a string and concatenate
    return value + str(5)

# --- Test Cases ---
assert concatenate_five("10") == "105", "Test 1 Failed: Expected '105'"
assert concatenate_five("abc") == "abc5", "Test 2 Failed: Expected 'abc5'"
assert concatenate_five("2") == "25", "Test 3 Failed: Expected '25'"

print(concatenate_five("10"))
print("\nSolution 2: All tests passed successfully!")
```

EXPLANATION:-

The str() function converts the integer 5 into the string "5". The operation then becomes "10" + "5", which concatenates the two strings to produce "105". This method changes the function's behavior to be a string operation, which might not be the user's initial intention, but it is a valid way to resolve the TypeError.

```
15

Solution 1: All tests passed successfully!
105

Solution 2: All tests passed successfully!
```