

## AI ASSIGNMENT-7.1

Name: Mohammed Faisal qureshi

Ht no:2303A53015

Batch no:46

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.

The screenshot shows a Jupyter Notebook interface with two code cells. The top cell contains the following code:

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

greet_debug()
#File "Untitled0.ipynb", line 2
print "Hello, AI Debugging Lab"

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

The bottom cell contains the corrected code and test cases:

```
# Corrected code with proper syntax
def greet_clean():
    return "Hello, AI Debugging Lab!"

# Test cases to confirm the corrected code works
# In fact, the assert keyword we used to capture output
# is also a common pattern for testing print statements.

import io
import sys

# Test Case 1: Check the return value (if we modify to return instead of print)
def test_greet_clean_returned():
    assert greet_clean() == "Hello, AI Debugging Lab", "Test Case 1 Failed: Incorrect return value"
    print("Test Case 1 Passed: Correct return value")

# Test Case 2: Capture output to verify print worked
def test_greet_clean_printed():
    # Redirect stdout to a string buffer
    old_stdout = sys.stdout
    redirected_output = io.StringIO()
    sys.stdout = redirected_output

    # Call the original 'greet' function (concept: it's here to print)
    # For simplicity, it's just print the expected string for testing.
    print("Hello, AI Debugging Lab!")

    # Restore stdout
    sys.stdout = old_stdout

    assert redirected_output.getvalue() == "Hello, AI Debugging Lab", "Test Case 2 Failed: incorrect print output"
    print("Test Case 2 Passed: correct print output")
```

The screenshot shows a Google Colab interface with three tabs: 'Lab Assignment T.T.', 'Lab 7 AI Debugging', and 'Untitled0.ipynb - Colab'. The main code cell contains Python code for testing a 'greet' function. The output pane shows AI-generated test cases and their results:

```
# Test Case 1: Check if greet() returns a string
def greet():
    return "Hello, AI Debugging Lab!"

assert isinstance(greet(), str)
print("Test Case 1 Passed: Correct return type!")

# Test Case 2: Check if greet() prints the correct string
def greet():
    print("Hello, AI Debugging Lab!")

assert greet() == "Hello, AI Debugging Lab!"
print("Test Case 2 Passed: Correct print output!")

# Test Case 3: Check if greet() returns the correct string length
def greet():
    return "Hello, AI Debugging Lab!"

assert len(greet()) == 16
print("Test Case 3 Passed: Correct string length!")

print("All assertion tests passed for the corrected code!")
```

The status bar at the bottom indicates the date and time: 12:01 PM, Python 3, ENG IN, 30-01-2026.

## TASK-02: (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!"
```

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.

```

# Bug: Using assignment (=) instead of comparison (==)
def check_number_buggy(n):
    if n == 10:
        return "True"
    else:
        return "Not True"

# Attaching to cell1, the buggy function will raise a SyntaxError in Python 3
# check_number_buggy(10)

# File: /content/python_code/104Assignment.ipynb, line 3
if n = 10:
    print("Valid")
else:
    print("Invalid")

```

**RuntimeError: Invalid syntax. Maybe you meant "==" or "!=" instead of "="**

Next step: Explain error

The above code with: `if n = 10:` would actually cause a `SyntaxError` in modern Python versions because assignment expressions (`=`) are not allowed directly in a boolean context without explicit parenthesization, as they don't produce a boolean value in the same way comparisons do. If this were in a language where assignment does return a value (often the assigned value itself), or in older Python versions where `If var = value` might implicitly evaluate to `True` if `value` is truthy, it would lead to a logical bug where the condition always evaluates to `True` (or `False` if `value` is falsy) because it's an assignment, not a comparison. The intention is to compare `n` to `10`, not assign `10` to `n`.

Now, let's correct this by using the comparison operator `==` and add test cases.

Variables Terminal ✓ 12:09PM Python 3 ENG IN 30-01-2026

```

# Corrected code using comparison (==)
def check_number_fixed():
    if n == 10:
        return "True"
    else:
        return "Not True"

# A few cases to verify the corrected code works
# Test Case 1: n is 10 should be "True"
assert check_number_fixed() == "True", "Test Case 1 Failed. Expected 'True' got [check_number_fixed()]"
print("Test Case 1 Passed: Correctly Identified 10!")

# Test Case 2: n is not 10 should be "Not True"
assert check_number_fixed() == "Not True", "Test Case 2 Failed. Expected 'Not True' got [check_number_fixed()]"
print("Test Case 2 Passed: Correctly Identified 5 as not 10")

# Test Case 3: Another value not 10 should be "Not True"
assert check_number_fixed() == "Not True", "Test Case 3 Failed. Expected 'Not True' got [check_number_fixed()]"
print("Test Case 3 Passed: Correctly Identified 15 as not 10")

print("All assertion tests passed for the corrected code!")

```

First, let's run the buggy code with the missing parenthesis in the `print` statement. This will result in a `SyntaxError`.

Variables Terminal ✓ 12:09PM Python 3 ENG IN 30-01-2026

### TASK-03:

(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.

```
[14] ① 0 # 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"))

***
FileNotFoundError Traceback (most recent call last)
/tmp/ipython-input-3019688361.py in <cell line: 0>()
      4     return f.read()
      5
----> 6 print(read_file("nonexistent.txt"))

/tmp/ipython-input-3019688361.py in read_file(filename)
      1 # Bug: Program crashes if file is missing
      2 def read_file(filename):
----> 3     with open(filename, 'r') as f:
      4         return f.read()
      5

FileNotFoundError: [Errno 2] No such file or directory: 'nonexistent.txt'

Next steps: Explain error
```

The code above will raise a `FileNotFoundException` because `nonexistent.txt` does not exist. This is a common runtime error that can be handled gracefully using `try-except` blocks. The `open()` function will fail if the file is not found, or if there are permission issues or an invalid path.

```
[15] ✓ On ① Import os

# Corrected code with safe error handling
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 path."
    except IOError as e:
        return f"Error: An I/O error occurred while reading '{filename}': {e}"
    except Exception as e:
        return f"Error: An unexpected error occurred: {e}"

# ---- Test Scenarios ----

# Scenario 1: File does not exist
print("\n---- Scenario 1: File Missing ----")
result_missing = read_file_safe("nonexistent_file.txt")
print(result_missing)
assert result_missing == "Error: The file 'nonexistent_file.txt' was not found", "Test Case 1 Failed: File not found"

# Scenario 2: Create a temporary file and read it (File exists)
print("\n---- Scenario 2: File Exists ----")
test_filename = "test_file.txt"
with open(test_filename, 'w') as f:
    f.write("This is a test file content.")

result_exists = read_file_safe(test_filename)
print(result_exists)
assert result_exists == "This is a test file content.", "Test Case 2 Failed: File content not read correctly."
e Clean up the temporary file
```

```
assert result_exists == "This is a test file content.", "Test Case 2 Failed: File content not as expected."  
  
# Clean up the temporary file  
os.remove(test_filename)  
  
# Scenario 3: Invalid path (e.g., trying to open a directory as a file)  
print("\n--- Scenario 3: Invalid Path (Directory) ---")  
# Create a temporary directory  
test_dir = "test_directory"  
os.makedirs(test_dir, exist_ok=True)  
  
result_invalid_path = read_file_safe(test_dir)  
print(result_invalid_path)  
assert "Error: An I/O error occurred" in result_invalid_path, "Test Case 3 Failed: Invalid path error not handled."  
  
# Clean up the temporary directory  
os.rmdir(test_dir)  
  
print("\nAll scenarios tested. Safe file handling implemented successfully!")  
  
***  
--- Scenario 1: File Missing ---  
Error: The file 'nonexistent_file.txt' was not found. Please check the file path.  
  
--- Scenario 2: File Exists ---  
This is a test file content.  
  
--- Scenario 3: Invalid Path (Directory) ---  
Error: An I/O error occurred while reading 'test_directory': [Errno 21] Is a directory: 'test_directory'  
  
All scenarios tested. Safe file handling implemented successfully!
```

## TASK-04: (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

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

## Expected Output #4:

- Corrected class with clear AI explanation.

```
[10] On # Bug: Calling an undefined method
class Car:
    def start(self):
        return "Car started"

my_car = Car()
print(my_car.drive()) # drive() is not defined

AttributeError Traceback (most recent call last)
/tmp/ipython-input-2715618295.py in <cell line: 8>()
      5
      6 my_car = Car()
--> 7 print(my_car.drive()) # drive() is not defined

AttributeError: 'Car' object has no attribute 'drive'

Next steps: Explain error
```

The code above will raise an `AttributeError`: 'Car' object has no attribute 'drive'. This error occurs because you are trying to call a method named `drive()` on an instance of the `Car` class, but the `Car` class definition does not include a method with that name.

To resolve this, you have two main options:

1. Define the missing method: Add a `drive` method to the `Car` class.
2. Correct the method call: Change `my_car.drive()` to call an existing method, such as `my_car.start()`.

For this task, I will proceed by correcting the method call to `start()`.

```
[11] On # Corrected class and method call
class Car:
    def start(self):
        return "Car started"

    def accelerate(self):
        return "Car is accelerating"

    def stop(self):
        return "Car stopped"

my_car_corrected = Car()

# Assert test cases to confirm the corrected class works
# Test Case 1: Calling the existing 'start' method
assert my_car_corrected.start() == "Car started", "Test Case 1 Failed: start() method not working correctly."
print("Test Case 1 Passed: start() method works.")

# Test Case 2: Calling another existing method (e.g., accelerate) to confirm general functionality
assert my_car_corrected.accelerate() == "Car is accelerating", "Test Case 2 Failed: accelerate() method not working correctly."
print("Test Case 2 Passed: accelerate() method works.")

# Test Case 3: Verify that calling the previously undefined method still raises an AttributeError (if not defined)
try:
    my_car_corrected.drive()
    assert False, "Test Case 3 Failed: calling non-existent method did not raise AttributeError."
except AttributeError as e:
    assert "'Car' object has no attribute 'drive'" in str(e), "Test Case 3 Failed: AttributeError message is incorrect."
    print("Test Case 3 Passed: Calling non-existent drive() correctly raises AttributeError.")

print("\nAll test cases passed for the corrected class!")

... Test Case 1 Passed: start() method works.
Test Case 2 Passed: accelerate() method works.
Test Case 3 Passed: Calling non-existent drive() correctly raises AttributeError.

All test cases passed for the corrected class!
```

Variables Terminal

## TASK-05: (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

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

print(add_five("10"))

<ipython-input-1235398455.py:1>
  1 # Bug: TypeError due to mixing string and integer
  2 def add_five(value):
  3     return value + 5
  4
  5 print(add_five("10"))

[11]: # Bug: TypeError due to mixing string and integer
      def add_five(value):
----> 3     return value + 5
      4
      5 print(add_five("10"))

TypeError: can only concatenate str (not "int") to str

Next steps: Explain error
```

The code above will raise a `TypeError: can only concatenate str (not "int") to str`. This error occurs because you are attempting to perform an addition operation (`+`) between a string ("10") and an integer (5). Python's `+` operator has different behaviors based on the data types: for numbers, it performs arithmetic addition; for strings, it performs concatenation. When types are mixed this way, Python doesn't know how to proceed, leading to a `TypeError`.

↳ Solution 1: Type Casting (Converting to Integer)

This solution involves converting the input string `value` to an integer using `int()` before performing the addition. This ensures both operands are integers, allowing for arithmetic addition.

```
[12]: # Corrected code using type casting
def add_five_type_cast(value):
    return int(value) + 5

# Assert test cases for type casting solution
# Test Case 1: Valid string that can be cast to int
assert add_five_type_cast("10") == 15, "Test Case 1 Failed: Expected 15 for '10'"
print("Test Case 1 Passed: '10' + 5 = 15")

# Test Case 2: Another valid string representation of an integer
assert add_five_type_cast("0") == 5, "Test Case 2 Failed: Expected 5 for '0'"
print("Test Case 2 Passed: '0' + 5 = 5")

# Test Case 3: Negative integer string
assert add_five_type_cast("-3") == -2, "Test Case 3 Failed: Expected -2 for '-3'"
print("Test Case 3 Passed: '-3' + 5 = 2")

print("All test cases passed for Type Casting solution!")

Test Case 1 Passed: '10' + 5 = 15
Test Case 2 Passed: '0' + 5 = 5
Test Case 3 Passed: '-3' + 5 = 2

All test cases passed for type casting solution!
```

## Solution 2: String Concatenation (Converting to String)

This solution assumes the intent was to combine the string representation of the number with the string representation of 5. It converts the integer 5 to a string using `str()` before performing the + operation, resulting in string concatenation.

```
[1]: 0% # Connected code using string concatenation
def add_five_string_concat(value):
    return value + str(5)

# Assert test cases for string concatenation solution
# Test Case 1: Valid string input
assert add_five_string_concat("10") == "105", "Test Case 1 Failed: Expected '105' for '10' + '5'"
print("Test Case 1 Passed: '10' + '5' = '105'")

# Test Case 2: Another string input
assert add_five_string_concat("abc") == "abc5", "Test Case 2 Failed: Expected 'abc5' for 'abc' + '5'"
print("Test Case 2 Passed: 'abc' + '5' = 'abc5'")

# Test Case 3: Empty string input
assert add_five_string_concat("") == "5", "Test Case 3 Failed: Expected '5' for empty string"
print("Test Case 3 Passed: '' + '5' = '5'")

print("\nAll test cases passed for String Concatenation solution!")

*** Test Case 1 Passed: '10' + '5' = '105'
Test Case 2 Passed: 'abc' + '5' = 'abc5'
Test Case 3 Passed: '' + '5' = '5'

All test cases passed for String Concatenation solution!
```