

AI Assisted Coding

Assignment - 7.1

V.Pranavith || 2303A51488|| Batch:- 08

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

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

```
[1] 0 print("Running the original buggy code to observe the error...")
# Original Bug: Missing parentheses in print statement (Python 2 syntax in a Python 3 environment)
# def greet_buggy():
#     print "Hello, AI Debugging Lab!"
# greet_buggy()

print("\n--- AI Debugging and Fix ---")

# Explanation of the error:
# In Python 3, "print" is a function and requires parentheses around its arguments.
# The original code print "Hello, AI Debugging Lab!" is Python 2 syntax and will raise a SyntaxError in Python 3.
# The fix is to enclose the string to be printed within parentheses: print("Hello, AI Debugging Lab!")

# Corrected code
def greet_corrected():
    return "Hello, AI Debugging Lab!"

# Calling the function to see the output
print("Corrected output:", greet_corrected())

print("\n--- Assert Test Cases ---")

# Assert test cases to confirm the corrected code works

# Test Case 1: Check if the function returns the expected string
assert greet_corrected() == "Hello, AI Debugging Lab!", "Test Case 1 Failed: Function did not return the expected string."
print("Test Case 1 Passed: Function returns the correct string.")

# Test Case 2: Check the type of the returned value
assert isinstance(greet_corrected(), str), "Test Case 2 Failed: Function did not return a string."
print("Test Case 2 Passed: Function returns a string.")

# Test Case 3: Check a partial string match (e.g., if it contains 'Hello')
assert "Hello" in greet_corrected(), "Test Case 3 Failed: Returned string does not contain 'Hello'."
print("Test Case 3 Passed: Returned string contains 'Hello'.")

print("\nAll assert tests passed. The corrected code is working as expected.")

x -- Running the original buggy code to observe the error...
--- AI Debugging and Fix ---
Corrected output: Hello, AI Debugging Lab!

--- Assert Test Cases ---
Test Case 1 Passed: Function returns the correct string.
Test Case 2 Passed: Function returns a string.
Test Case 3 Passed: Returned string contains 'Hello'.

All assert tests passed. The corrected code is working as expected.
```

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.

```
[2] 0 print("Running the original buggy code to observe the error...")
.0s # Original Bug: Using assignment (=) instead of comparison (==) in an if-statement
# def check_number_buggy(n):
#     if n = 10:
#         return "Ten"
#     else:
#         return "Not Ten"
# try:
#     print(check_number_buggy(10))
# except SyntaxError as e:
#     print(f"Caught expected errors: {e}")
print("\n--- AI Debugging and Fix ---")

# Explanation of the error:
# In Python, the single equals sign '=' is used for assignment, meaning it assigns a value to a variable.
# The double equals sign '==' is used for comparison, checking if two values are equal.
# When 'n = 10' is used inside an 'if' statement, Python interprets it as an attempt to assign the value 10 to 'n'.
# This is syntactically invalid within an 'if' statement in Python 3 because assignment expressions are not allowed in the test condition unless explicitly wrapped (e.g., using the walrus operator := in Python 3.8+).
# Even if it were allowed, it would result in an assignment rather than a comparison, leading to incorrect logic or unexpected behavior.
# The fix is to change 'n = 10' to 'n == 10' to perform a comparison.

# Corrected code
def check_number_corrected(n):
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"

# Calling the function to see the output
print("Corrected output (n=10):", check_number_corrected(10))
print("Corrected output (n=5):", check_number_corrected(5))

print("\n--- Assert Test Cases ---")

# Assert test cases to confirm the corrected code works

# Test Case 1: Check with the value that should return "Ten"
assert check_number_corrected(10) == "Ten", "Test Case 1 Failed: Expected 'Ten' for input 10."
print("Test Case 1 Passed: Correctly returns 'Ten' for 10.")

# Test Case 2: Check with a value that should return "Not Ten"
assert check_number_corrected(5) == "Not Ten", "Test Case 2 Failed: Expected 'Not Ten' for input 5."
print("Test Case 2 Passed: Correctly returns 'Not Ten' for 5.")

# Test Case 3: Check with another value that should return "Not Ten"
assert check_number_corrected(15) == "Not Ten", "Test Case 3 Failed: Expected 'Not Ten' for input 15."
print("Test Case 3 Passed: Correctly returns 'Not Ten' for 15.")

print("\nAll assert tests passed. The corrected code is working as expected.")

```
- Running the original buggy code to observe the error...
 --- AI Debugging and Fix ---
 Corrected output (n=10): Ten
 Corrected output (n=5): Not Ten

 --- Assert Test Cases ---
 Test Case 1 Passed: Correctly returns 'Ten' for 10.
 Test Case 2 Passed: Correctly returns 'Not Ten' for 5.
 Test Case 3 Passed: Correctly returns 'Not Ten' for 15.

 All assert tests passed. The corrected code is working as expected.

```

## 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.

```
[3] 0 Import os
 # Create a dummy file for testing 'file exists' scenario
 with open('existent_file.txt', 'w') as f:
 f.write("This is a test file content.")

 print("Running the original buggy code to observe the error...")
 # Original Bug: Program crashes if file is missing
 # def read_file_buggy(filename):
 # with open(filename, 'r') as f:
 # return f.read()
 # try:
 # print(read_file_buggy("nonexistent.txt"))
 # except FileNotFoundError as e:
 # print(f"Caught expected error: {e}")

 print("\n--- AI Debugging and Fix ---")

 # Explanation of the error:
 # The original code attempts to open a file in read mode ('r'). If the specified file
 # does not exist, Python will raise a 'FileNotFoundError', causing the program to crash.
 # To prevent this, we use a 'try-except' block to catch the 'FileNotFoundError'
 # (and other potential I/O errors) and provide a graceful, user-friendly response.

 # Corrected code with safe error handling
 def read_file_safely(filename):
 try:
 with open(filename, 'r') as f:
 content = f.read()
 return f"Successfully read from '{filename}':\n{content}"
 except FileNotFoundError:
 return f"Error: The file '{filename}' was not found. Please check the file path and name."
 except IsADirectoryError:
 return f"Error: '{filename}' is a directory, not a file. Cannot read directory contents directly like a file."
 except PermissionError:
 return f"Error: Permission denied to access '{filename}'. Check file permissions."
 except IOError as e:
 return f"An unexpected I/O error occurred while reading '{filename}': {e}"
 except Exception as e:
 return f"An unexpected error occurred: {e}"

 print("\n--- Test Scenarios ---")

 # Scenario 1: File exists
 print("\nScenario 1: Testing with an existing file (existent_file.txt)")
 result_existing = read_file_safely("existent_file.txt")
 print(result_existing)
 # Assertions for Scenario 1
 assert "Successfully read from 'existent_file.txt'" in result_existing, "Test Case 1 Failed: Should successfully read existing file."
 print("Test Case 1 Passed: Successfully handled existing file.")

 # Scenario 2: File missing
 print("\nScenario 2: Testing with a missing file (nonexistent_file.txt)")
 result_missing = read_file_safely("nonexistent_file.txt")
 print(result_missing)
 # Assertions for Scenario 2
 assert "Error: The file 'nonexistent_file.txt' was not found" in result_missing, "Test Case 2 Failed: Should report file not found."
 print("Test Case 2 Passed: Successfully handled missing file.")

 # Scenario 3: Invalid path (attempt to read a directory as a file)
 # We'll use the current directory '.' as an example of an invalid path
 print("\nScenario 3: Testing with an invalid path (current directory '.')")
 result_invalid_path = read_file_safely(".")
 print(result_invalid_path)
 # Assertions for Scenario 3
 assert "'.' is a directory, not a file" in result_invalid_path, "Test Case 3 Failed: Should report a directory error."
 print("Test Case 3 Passed: Successfully handled invalid path (directory).")

 print("\nAll assert tests passed. Safe file handling implemented and working as expected.")

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

```
* - Running the original buggy code to observe the error...
--- AI Debugging and Fix ---
--- Test Scenarios ---

Scenario 1: Testing with an existing file (existent_file.txt)
Successfully read from 'existent_file.txt':
This is a test file content.
Test Case 1 Passed: Successfully handled existing file.

Scenario 2: Testing with a missing file (nonexistent_file.txt)
Error: The file 'nonexistent_file.txt' was not found. Please check the file path and name.
Test Case 2 Passed: Successfully handled missing file.

Scenario 3: Testing with an invalid path (current directory '.')
Error: '.' is a directory, not a file. Cannot read directory contents directly like a file.
Test Case 3 Passed: Successfully handled invalid path (directory).

All assert tests passed. Safe file handling implemented and working as expected.
```

#### 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.

```
(4) 0 print("Running the original buggy code to observe the error...")
Original Bug: Calling an undefined method
class CarBuggy:
def start(self):
return "Car started"
my_car_buggy = CarBuggy()
try:
print(my_car_buggy.drive()) # drive() is not defined
except AttributeError as e:
print(f"Caught expected error: {e}")

print("\n--- AI Debugging and Fix ---")

Explanation of the error:
The original code attempts to call 'my_car.drive()', but the 'Car' class does not have a method named 'drive'.
In Python, attempting to access an attribute (which includes methods) that does not exist on an object
will raise an 'AttributeError'. This indicates that the object does not possess the requested functionality.

Analysis for correction:
When encountering an 'AttributeError' for a missing method, there are two primary approaches:
1. Define the missing method: If the intended functionality is genuinely missing from the class design,
the correct solution is to add the method to the class.
2. Correct the method call: If a method with similar functionality already exists (e.g., 'move' instead of 'drive'),
or if the method call was simply a typo, then changing the call to the correct method name is the fix.

For this task, we will demonstrate the first approach: defining the missing method.

Corrected class with the 'drive' method defined
class CarCorrected:
 def start(self):
 return "Car started"

 def drive(self):
 return "Car is driving"

Creating an instance of the corrected class
my_car_corrected = CarCorrected()

Calling the methods to see the output
print("Corrected output (start method):", my_car_corrected.start())
print("Corrected output (drive method):", my_car_corrected.drive())

print("\n--- Assert Test Cases ---")

Assert test cases to confirm the corrected class works

Test Case 1: Check if the 'start' method returns the expected string
assert my_car_corrected.start() == "Car started", "Test Case 1 Failed: 'start' method did not return the expected string."
print("Test Case 1 Passed: 'start' method works correctly.")

Test Case 2: Check if the newly defined 'drive' method returns the expected string
assert my_car_corrected.drive() == "Car is driving", "Test Case 2 Failed: 'drive' method did not return the expected string."
print("Test Case 2 Passed: 'drive' method works correctly.")

Test Case 3: Check the type of the returned value from 'drive' method
assert isinstance(my_car_corrected.drive(), str), "Test Case 3 Failed: 'drive' method did not return a string."
print("Test Case 3 Passed: 'drive' method returns a string.")

print("\nAll assert tests passed. The corrected code is working as expected.")

-- Running the original buggy code to observe the error...

--- AI Debugging and Fix ---
Corrected output (start method): Car started
Corrected output (drive method): Car is driving

--- Assert Test Cases ---
Test Case 1 Passed: 'start' method works correctly.
Test Case 2 Passed: 'drive' method works correctly.
Test Case 3 Passed: 'drive' method returns a string.

All assert tests passed. The corrected code is working as expected.
```

## 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

```
[4] 0 print("Running the original buggy code to observe the error...")
>0s 0 Original Bug: Calling an undefined method
class CarBuggy:
def start(self):
return "Car started"
my_car_buggy = CarBuggy()
try:
print(my_car_buggy.drive()) # drive() is not defined
except AttributeError as e:
print(f"Caught expected error: {e}")
print("\n--- AI Debugging and Fix ---")

Explanation of the error:
The original code attempts to call 'my_car.drive()', but the 'Car' class does not have a method named 'drive'.
In Python, attempting to access an attribute (which includes methods) that does not exist on an object
will raise an 'AttributeError'. This indicates that the object does not possess the requested functionality.

Analysis for correction:
When encountering an 'AttributeError' for a missing method, there are two primary approaches:
1. Define the missing method: If the intended functionality is genuinely missing from the class design,
the common approach is to add the method to the class.
2. Correct the method call: If a method with similar functionality already exists (e.g., 'move' instead of 'drive'),
or if the method call was simply a typo, then changing the call to the correct method name is the fix.

For this task, we will demonstrate the first approach: defining the missing method.

Corrected class with the 'drive' method defined
class CarCorrected:
 def start(self):
 return "Car started"

 def drive(self):
 return "Car is driving"

Creating an instance of the corrected class
my_car_corrected = CarCorrected()

Calling the methods to see the output
print("Corrected output [start method]:", my_car_corrected.start())
print("Corrected output [drive method]:", my_car_corrected.drive())
print("\n--- Assert Test Cases ---")

Assert test cases to confirm the corrected class works

Test Case 1: Check if the 'start' method returns the expected string
assert my_car_corrected.start() == "Car started", "Test Case 1 Failed: 'start' method did not return the expected string."
print("Test Case 1 Passed: 'start' method works correctly.")

Test Case 2: Check if the newly defined 'drive' method returns the expected string
assert my_car_corrected.drive() == "Car is driving", "Test Case 2 Failed: 'drive' method did not return the expected string."
print("Test Case 2 Passed: 'drive' method works correctly.")

Test Case 3: Check the type of the returned value from 'drive' method
assert isinstance(my_car_corrected.drive(), str), "Test Case 3 Failed: 'drive' method did not return a string."
print("Test Case 3 Passed: 'drive' method returns a string.")

print("\nAll assert tests passed. The corrected code is working as expected.")

> - Running the original buggy code to observe the error...
--- AI Debugging and Fix ---
Corrected output [start method]: Car started
Corrected output [drive method]: Car is driving

--- Assert Test Cases ---
Test Case 1 Passed: 'start' method works correctly.
Test Case 2 Passed: 'drive' method works correctly.
Test Case 3 Passed: 'drive' method returns a string.

All assert tests passed. The corrected code is working as expected.
```

```
y ~ Running the original buggy code to observe the error...
--- AI Debugging and Fix ---

--- Solution 1: Type Casting (Convert string to integer for mathematical addition) ---
Corrected output (type casting with '10'): 15
Corrected output (type casting with '20'): 25

--- Assert Test Cases for Type Casting Solution ---
Test Case 1 Passed: Correctly adds integer after type casting.
Test Case 2 Passed: Handles '0' correctly.
Test Case 3 Passed: Handles negative numbers correctly.

All assert tests for Type Casting passed. The corrected code is working as expected.

--- Solution 2: String Concatenation (Convert integer to string for string joining) ---
Corrected output (string concatenation with '10'): 105
Corrected output (string concatenation with 'A'): A5

--- Assert Test Cases for String Concatenation Solution ---
Test Case 1 Passed: Correctly concatenates '5' as a string.
Test Case 2 Passed: Handles alphabetical strings correctly.
Test Case 3 Passed: Handles empty strings correctly.

All assert tests for String Concatenation passed. The corrected code is working as expected.
```