

Name:- NIPUN I

H.T.NO:- 2303A52208

Assignment :- 7.1

Task:01

(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.

Code:

```
def greet():
    print("hello world")
greet()
assert greet() is None          # Test 1: return value should be None
assert str(greet()) == "None"    # Test 2: string of None is "None"
assert repr(greet()) == "None"   # Test 3: repr of None is also "None"
print("All test cases passed!")
```



OUTPUT:-

```
● hello world
hello world
hello world
hello world
All test cases passed!
```

Observation1:

1. Missing parentheses in print causes a syntax error in modern Python.
 2. Adding parentheses corrects the syntax and allows the program to run.
 3. The corrected function behaves properly and can be verified using assert tests.
-

Task2:

ask: 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:

```
#task 2
def check_number(n):
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"
print(check_number(10))
print(check_number(5))
# Test case 1: when input is 10
assert check_number(10) == "Ten"

# Test case 2: when input is not 10
assert check_number(5) == "Not Ten"

# Test case 3: check return type is always string
assert isinstance(check_number(10), str)

print("All test cases passed!")
#task 3:
```

Output :

```
All test cases passed.
Ten
Not Ten
All test cases passed!
AI Debugging Lab
```

Observation2:

Using = in an if condition causes a syntax error in Python.

Replacing it with == correctly compares values and fixes the bug.

The corrected function works as expected and passes all assert tests.

```
=====
```

Task3:

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:

```
#task 3:
def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()

    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found."

    except OSError:
        return f"Error: '{filename}' is an invalid file path."
# Scenario 1 - file exists
with open("sample.txt", "w") as f:
    f.write("AI Debugging Lab")

print(read_file("sample.txt"))

# Scenario 2 - file missing
print(read_file("missing.txt"))

# Scenario 3 - invalid path
print(read_file("C:\\\\//??"))

print("\nSafe file handling with exception management.")
```

Output:

```
not run
All test cases passed!
AT Debugging Lab
Error: The file 'missing.txt' was not found.
Error: 'C:\\\\//??' is an invalid file path.

Safe file handling with exception management.
```

Observation3:

1. Attempting to open a missing file without handling exceptions causes the program to crash.
2. Using a try-except block allows the program to handle errors safely with a user-friendly message.
3. Testing with different scenarios confirms that the file handling works reliably in all cases.

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. 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.

Code:

```
#task 4
class Car:
    def start(self):
        return "Car started"

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

my_car = Car()
print(my_car.drive())
my_car = Car()

# Test Case 1: Check start method
assert my_car.start() == "Car started"

# Test Case 2: Check drive method
assert my_car.drive() == "Car is driving"

# Test Case 3: Check that return type is string
assert isinstance(my_car.drive(), str)

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

Output:

```
Save file handling with exception management.
Car is driving
All test cases passed!
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>
```

Observation4:

1. Calling a method that is not defined in a class causes an `AttributeError`.
 2. The bug can be fixed either by adding the missing method or correcting the method call.
 3. After correction, the class works properly and is verified using assert tests.
-

Task 5:

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.

Code:

```
#task 5
def add_five_cast(value):
    return int(value) + 5
print(add_five_cast("10"))
def add_five_concat(value):
    return value + str(5)
print(add_five_concat("10")) # Output: "105"
# ---- Tests for Type Casting Solution ----
assert add_five_cast("10") == 15
assert isinstance(add_five_cast("10"), int)

# ---- Test for String Concatenation ----
assert add_five_concat("10") == "105"

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

Output:

```
15
105
All test cases passed!
PS C:\Users\gadda\OneDrive\Desktop\Ai Assitant coding>
```

Observations 5:

Python raises a `TypeError` when adding a string and an integer directly.

The error can be fixed either by type casting or string concatenation.

AI helps identify the bug and suggests correct and working solution.