

Assignment-7.1

Ch.Sanhith reddy

2303A52251

Batch-43

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!"
```

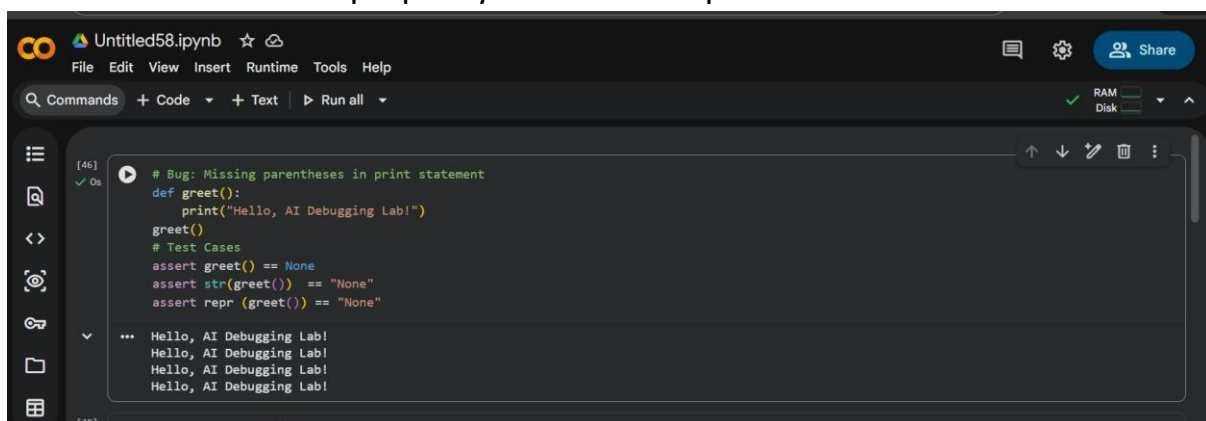
```
greet()
```

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.



The screenshot shows a Jupyter Notebook titled 'Untitled58.ipynb'. The code cell contains the following Python code:

```
# Bug: Missing parentheses in print statement
def greet():
    print("Hello, AI Debugging Lab!")
greet()
# Test Cases
assert greet() == None
assert str(greet()) == "None"
assert repr(greet()) == "None"
```

The output cell shows the result of running the code:

```
... Hello, AI Debugging Lab!
Hello, AI Debugging Lab!
Hello, AI Debugging Lab!
Hello, AI Debugging Lab!
```

Task Description #2 (Incorrect condition in an If Statement)

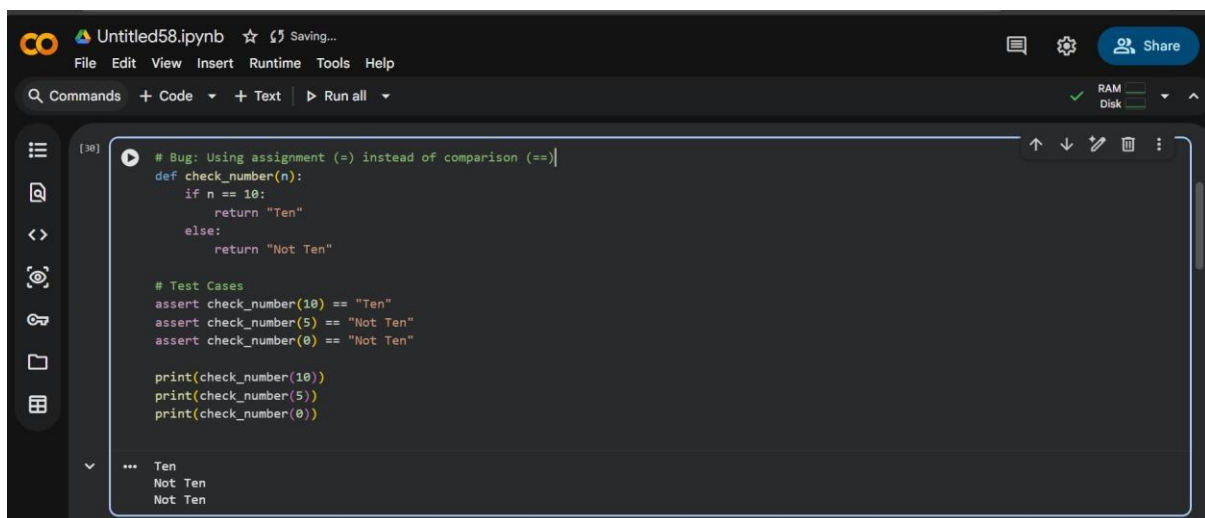
Task: Supply a function where an if-condition mistakenly uses = instead of ==. Let AI identify and fix the issue. # Bug: Using assignment (=) instead of comparison (==) def check_number(n): if n = 10: return "Ten" else: return "Not Ten"

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.



The screenshot shows a Jupyter Notebook window titled 'Untitled58.ipynb'. The code in the cell is as follows:

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

# Test Cases
assert check_number(10) == "Ten"
assert check_number(5) == "Not Ten"
assert check_number(0) == "Not Ten"

print(check_number(10))
print(check_number(5))
print(check_number(0))
```

The output of the code is displayed below the cell:

```
*** Ten
Not Ten
Not Ten
```

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()

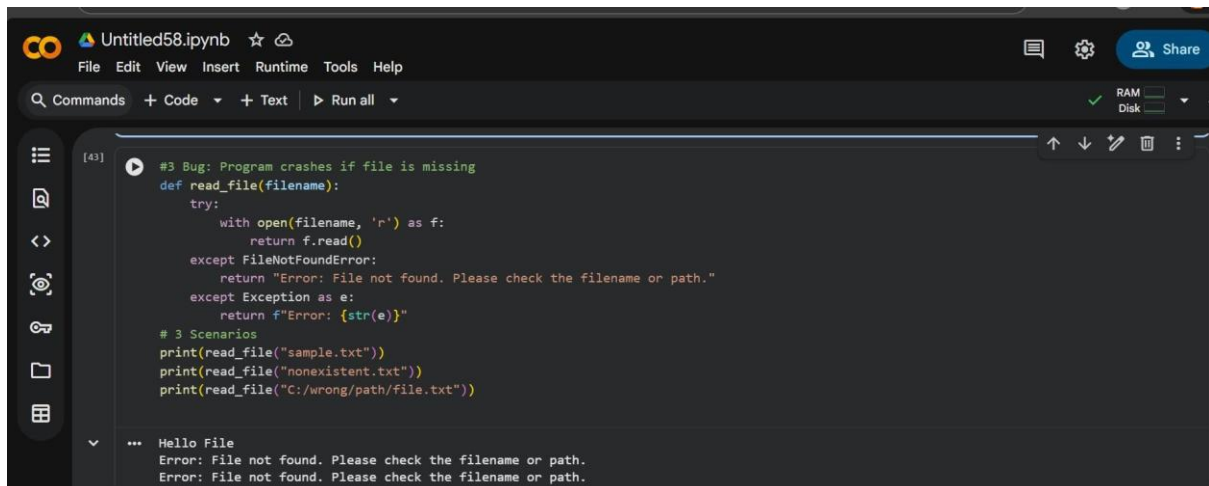
print(read_file("nonexistent.txt"))
```

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.



The screenshot shows a Jupyter Notebook titled 'Untitled58.ipynb'. The code cell contains a function `read_file(filename)` that uses a try-except block to handle `FileNotFoundError` and other exceptions. It includes three test scenarios: a valid file, a non-existent file, and an invalid path. The output shows the function successfully reading a file, and then printing error messages for the missing file and invalid path.

```
[43]: #3 Bug: Program crashes if file is missing
def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return "Error: File not found. Please check the filename or path."
    except Exception as e:
        return f"Error: {str(e)}"

# 3 Scenarios
print(read_file("sample.txt"))
print(read_file("nonexistent.txt"))
print(read_file("C:/wrong/path/file.txt"))

... Hello File
Error: File not found. Please check the filename or path.
Error: File not found. Please check the filename or path.
```

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()
```

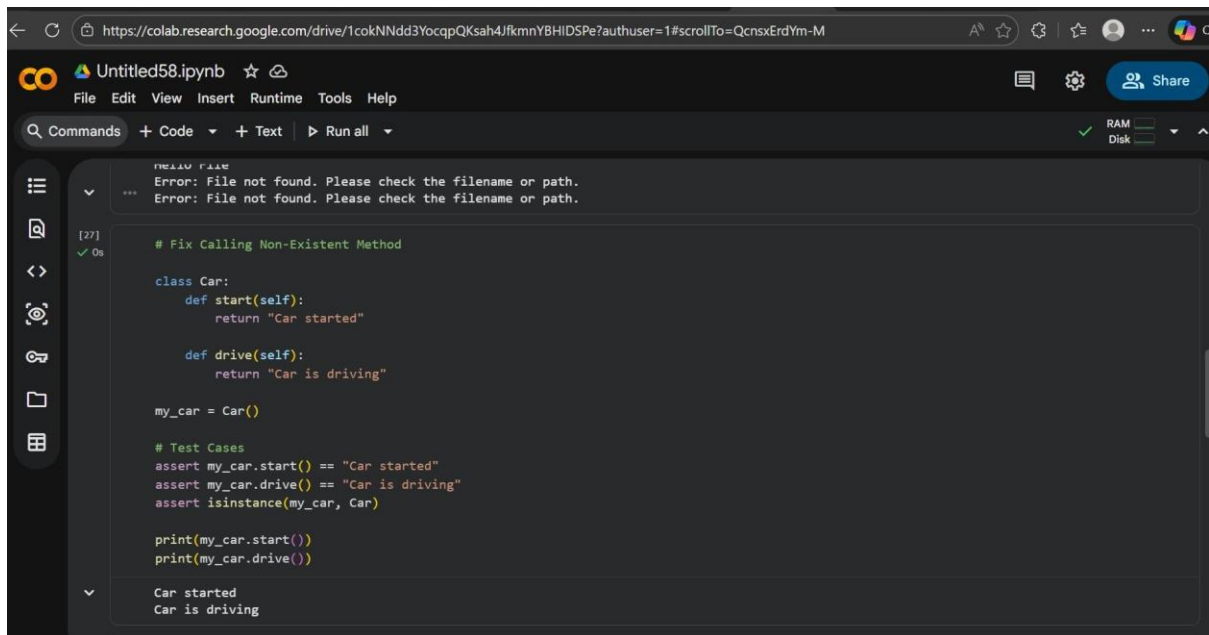
```
print(my_car.drive()) # drive() is not defined
```

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.



The screenshot shows a Google Colab notebook interface. The top bar includes the Colab logo, the file name 'Untitled58.ipynb', and various icons for file management, settings, and sharing. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The main area is a code editor with a dark background. It contains a Python class definition for 'Car' with two methods: 'start' and 'drive'. The 'start' method returns 'Car started' and the 'drive' method returns 'Car is driving'. Below the class definition, there are three assert tests: 'assert my_car.start() == "Car started"', 'assert my_car.drive() == "Car is driving"', and 'assert isinstance(my_car, Car)'. The code is followed by two print statements: 'print(my_car.start())' and 'print(my_car.drive())'. The output of the code is shown at the bottom: 'Car started' and 'Car is driving'.

```
FILE NOT FOUND
Error: File not found. Please check the filename or path.
***
Error: File not found. Please check the filename or path.

[27]
✓ 0s

# Fix Calling Non-Existent Method

class Car:
    def start(self):
        return "Car started"

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

my_car = Car()

# Test Cases
assert my_car.start() == "Car started"
assert my_car.drive() == "Car is driving"
assert isinstance(my_car, Car)

print(my_car.start())
print(my_car.drive())

Car started
Car is driving
```

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
```

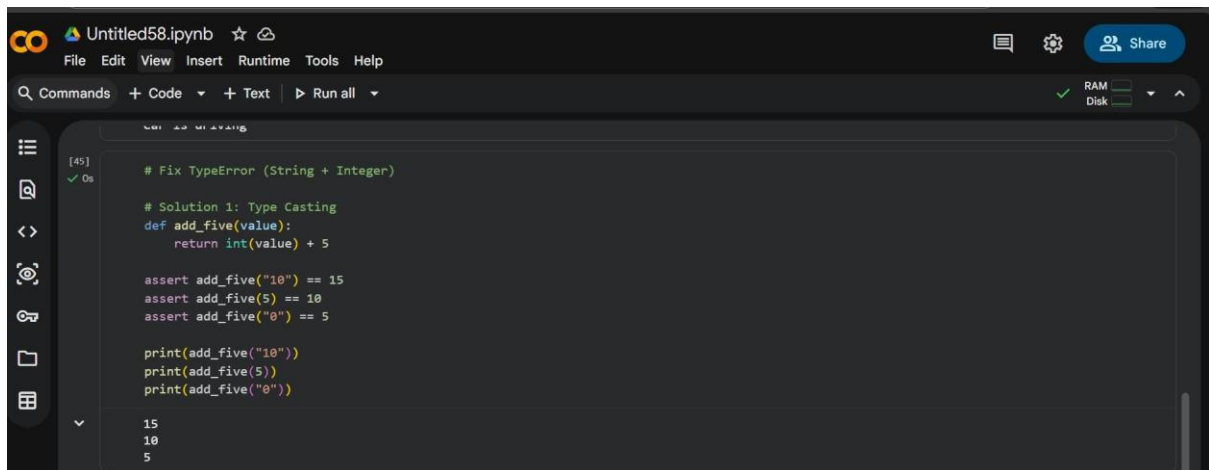
```
print(add_five("10"))
```

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.



The screenshot shows a Jupyter Notebook interface with the title 'Untitled58.ipynb'. The code cell [45] contains the following Python code:

```
# Fix TypeError (String + Integer)

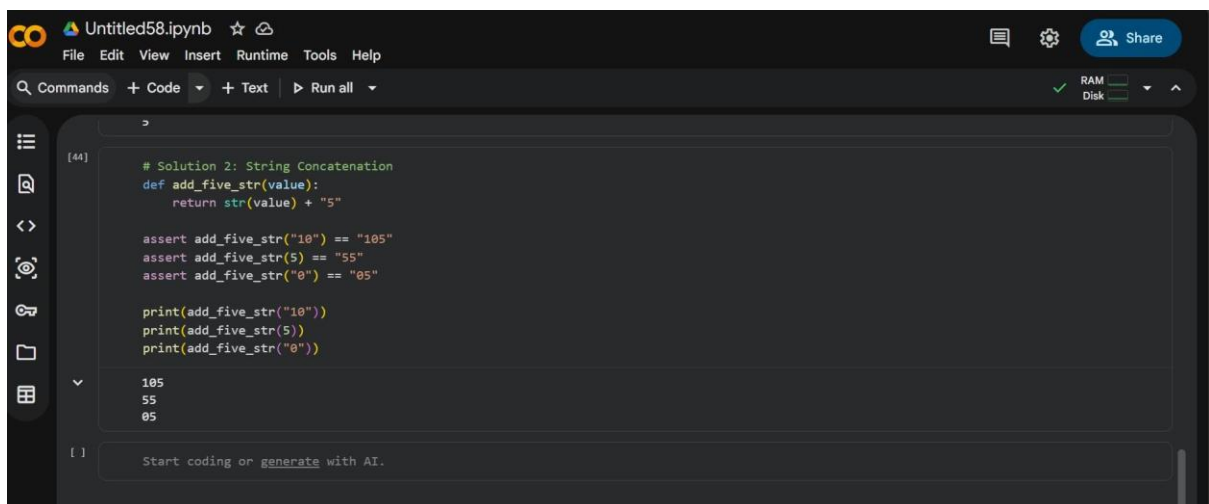
# Solution 1: Type Casting
def add_five(value):
    return int(value) + 5

assert add_five("10") == 15
assert add_five(5) == 10
assert add_five("0") == 5

print(add_five("10"))
print(add_five(5))
print(add_five("0"))
```

The output of the code cell is:

```
15
10
5
```



The screenshot shows a Jupyter Notebook interface with the title 'Untitled58.ipynb'. The code cell [44] contains the following Python code:

```
# Solution 2: String Concatenation
def add_five_str(value):
    return str(value) + "5"

assert add_five_str("10") == "105"
assert add_five_str(5) == "55"
assert add_five_str("0") == "05"

print(add_five_str("10"))
print(add_five_str(5))
print(add_five_str("0"))
```

The output of the code cell is:

```
105
55
05
```

Below the code cell, there is a prompt: "Start coding or generate with AI."