

# **Lab 7.1: Error Debugging with AI – Systematic Approaches to Finding and Fixing Bugs**

**Name:B.TejasKumar**

**HallTicketNo:2303A51489**

## **Lab Outcomes**

- Detect and correct syntax, logic, and runtime errors.
- Understand AI explanations for bugs.
- Apply structured debugging strategies.
- Refactor buggy code safely and correctly.

## **Task1:SyntaxError–MissingParenthesesinPrintStatement**

BuggyCode:

```
defgreet():
    print"Hello,AIDebuggingLab!"
```

ObservedError:

SyntaxError:Missingparenthesesincallto'print'

AIExplanation:

Python3 requires parentheses for the print() function.

CorrectedCode:

```
defgreet():
    return"Hello,AIDebuggingLab!"

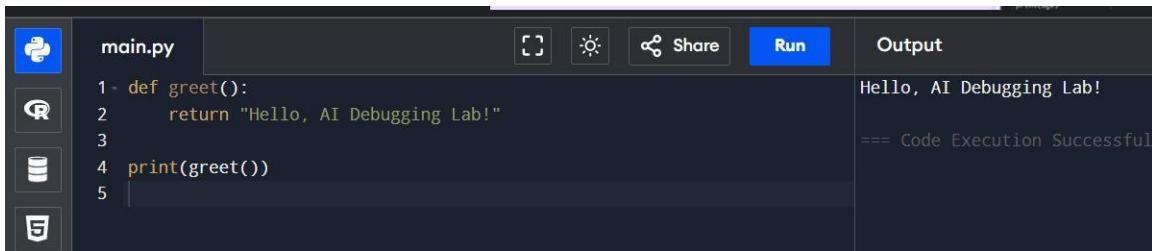
print(greet())
```

Assert Test Cases:

```
assert greet() == "Hello, AI Debugging Lab!"  
assert isinstance(greet(), str)  
assert greet().startswith("Hello")
```

Output:

Hello, AI Debugging Lab!



The screenshot shows a Jupyter Notebook interface with a dark theme. On the left, there is a sidebar with icons for file operations (New, Open, Save, etc.) and a search bar. The main area has a tab labeled "main.py". The code cell contains the following Python code:

```
1- def greet():  
2     return "Hello, AI Debugging Lab!"  
3  
4 print(greet())  
5
```

To the right of the code cell is the "Run" button, which is highlighted in blue. Below the code cell, the output pane displays the results of the execution:

```
Hello, AI Debugging Lab!  
== Code Execution Successful
```

## Task 2: Logic Error – Incorrect Condition in If Statement

Buggy Code:

```
def check_number(n):
    if n = 10:
        return "Ten"
```

AI Explanation:

= is assignment, == is comparison.

Corrected Code:

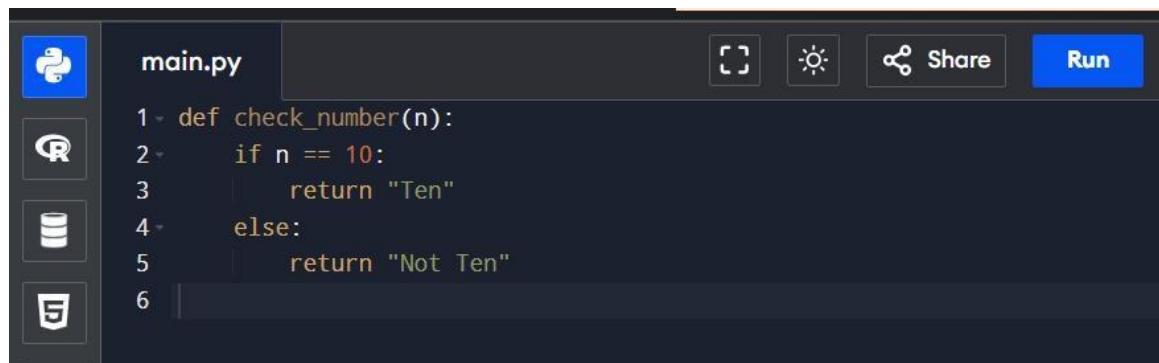
```
def check_number(n):
    if n == 10:
        return "Ten"
    else:
        return "Not Ten"
```

Assert Test Cases:

```
assert check_number(10) == "Ten"
assert check_number(5) == "Not Ten"
assert check_number(0) == "Not Ten"
```

Output:

All test cases passed



```
1 def check_number(n):
2     if n == 10:
3         return "Ten"
4     else:
5         return "Not Ten"
6
```

## Task 3: Runtime Error – File Not Found

Corrected Code:

```
def read_file(filename):
    try:
        with open(filename, 'r') as f:
```

```

        return f.read()
    except FileNotFoundError:
        return "Error: File not found"
    except OSError:
        return "Error: Invalid file path"

```

Output:

Error: File not found



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code and its output. The code defines a function `read_file` that reads a file. It handles `FileNotFoundError` by returning "Error: File not found" and `OSError` by returning "Error: Invalid file path". The code then calls `read_file` with the argument "nonexistent.txt". The output panel shows the error message "ERROR! Error: File not found" followed by "== Code Execution Successful".

```

main.py
1 def read_file(filename):
2     try:
3         with open(filename, 'r') as f:
4             return f.read()
5     except FileNotFoundError:
6         return "Error: File not found"
7     except OSError:
8         return "Error: Invalid file path"
9
10 |
11 print(read_file("nonexistent.txt"))
12

```

Output

ERROR!  
Error: File not found  
== Code Execution Successful

#### Task 4: Calling a Non-Existent Method

Corrected Code:

```

class Car:
    def start(self):
        return "Car started"
    def drive(self):
        return "Car is driving"

```

Output:

Car is driving



The screenshot shows a Jupyter Notebook interface with a code cell containing Python code and its output. The code defines a class `Car` with methods `start` and `drive`. It then creates an instance `my_car` and prints the result of calling `drive` on it. The output panel shows "Car is driving" followed by "== Code Execution Successful".

```

main.py
1 class Car:
2     def start(self):
3         return "Car started"
4
5     def drive(self):
6         return "Car is driving"
7
8
9 my_car = Car()
10 print(my_car.drive())
11

```

Output

Car is driving  
== Code Execution Successful

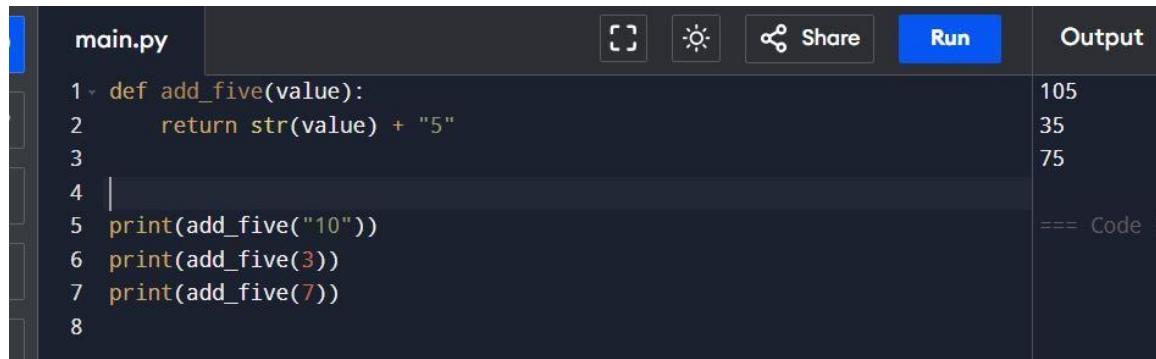
## Task 5: TypeError – Mixing Strings and Integers

Solution 1:

```
def add_five(value):
    return int(value) + 5
```

Solution 2:

```
def add_five(value):
    return str(value) + "5"
```



The screenshot shows a code editor interface with a dark theme. On the left is the code editor pane containing `main.py`. The code defines a function `add_five` that adds 5 to its input. It also contains three print statements. On the right is the output pane, which displays the results of running the code. The output is as follows:

Output
105
35
75

Below the output, there is a message: `==== Code`.

## Conclusion

This lab demonstrated how AI-assisted debugging helps identify errors, explain bugs clearly, and suggest safe fixes.