

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

Name: B. Sarayu

Hall Ticket No: 2303A51842

Week: 4 (Monday)

Lab Objectives

- To identify and correct syntax, logic, and runtime errors in Python programs using AI tools.
- To understand common programming bugs and AI-assisted debugging suggestions.
- To evaluate how AI explains, detects, and fixes different types of coding errors.
- To build confidence in using AI for systematic debugging.

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.

Task 1: Syntax Error – Missing Parentheses in Print Statement

Buggy Code:

```
def greet():  
    print "Hello, AI Debugging Lab!"
```

Observed Error:

SyntaxError: Missing parentheses in call to 'print'

AI Explanation:

Python 3 requires parentheses for the print() function.

Corrected Code:

```
def greet():  
    return "Hello, AI Debugging Lab!"
```

```
print(greet())
```

Assert Test Cases:

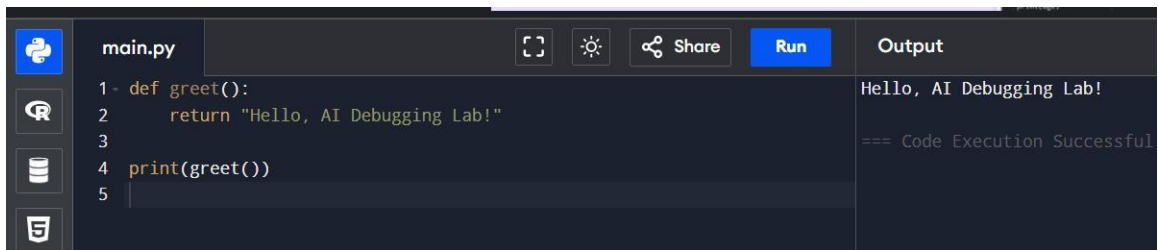
```
assert greet() == "Hello, AI Debugging Lab!"
```

```
assert isinstance(greet(), str)
```

```
assert greet().startswith("Hello")
```

Output:

Hello, AI Debugging Lab!



The screenshot shows a code editor interface with a dark theme. On the left, there is a sidebar with icons for Python, a debugger, a database, and a file explorer. The main editor area displays a file named 'main.py' containing the following Python code:

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

At the top of the editor, there are icons for a full-screen view, settings, and a 'Share' button, followed by a blue 'Run' button. To the right of the code editor is an 'Output' panel. It displays the output of the program: 'Hello, AI Debugging Lab!' followed by '=== 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:

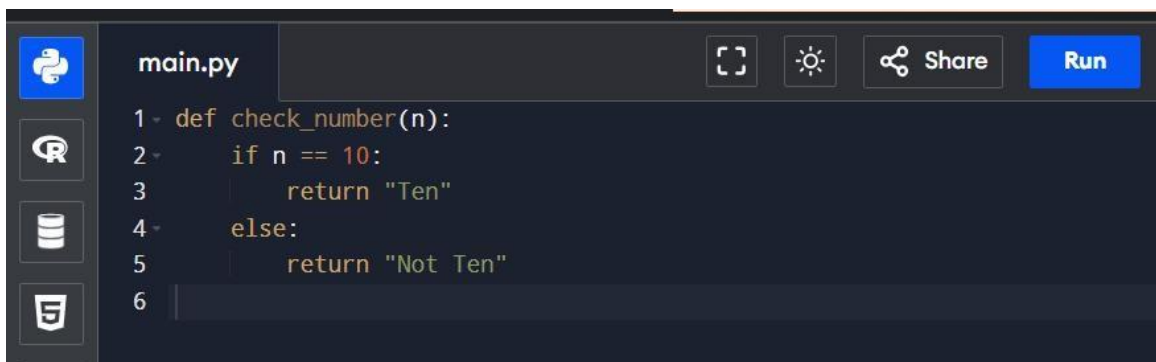
```
def check_number(n):  
    if n == 10:  
        return "Ten"  
    else:  
        return "Not Ten"
```

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

A screenshot of a code editor interface. The editor has a dark theme. On the left, there is a sidebar with icons for Python, a debugger, a database, and a file explorer. The main area shows a file named 'main.py' with the following code:

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

At the top right of the editor, there are icons for a full-screen view, settings, and a 'Share' button, followed by a blue 'Run' button.

Task 3: Runtime Error – File Not Found

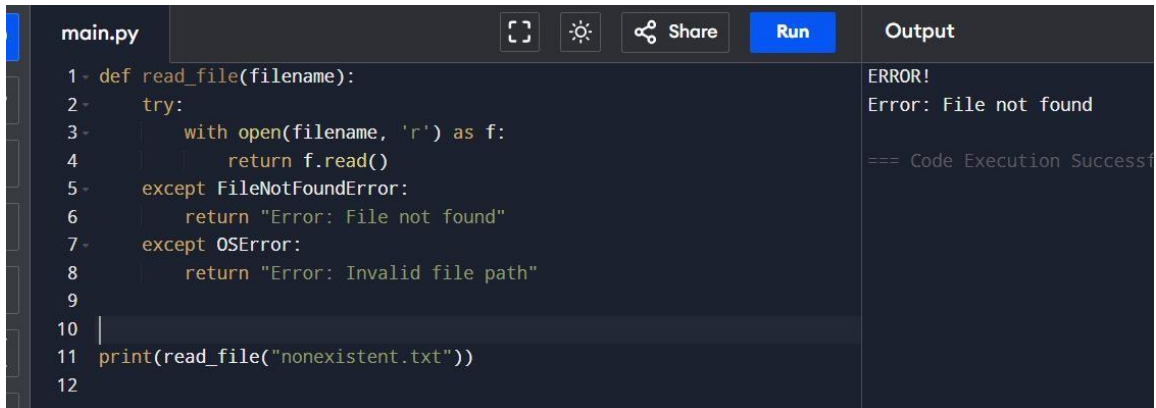
Corrected Code:

```
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 code editor with a file named 'main.py'. The code defines a function 'read_file(filename)' that attempts to open a file in read mode. If the file is not found, it returns 'Error: File not found'. If there is an OS error, it returns 'Error: Invalid file path'. The script then calls 'read_file("nonexistent.txt")' and prints the result. The output pane on the right shows 'ERROR!' followed by 'Error: File not found' and '=== 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 code editor with a file named 'main.py'. The code defines a 'Car' class with 'start' and 'drive' methods. It then creates an instance 'my_car' and calls 'my_car.drive()'. The output pane on the right shows 'Car is driving' and '=== 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

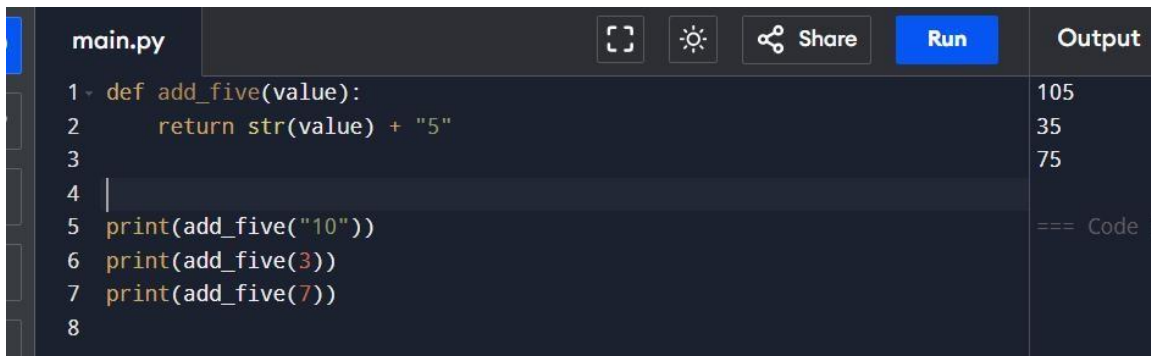
Task 5: TypeError – Mixing Strings and Integers

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 with a dark theme. The file name is 'main.py'. The code defines a function 'add_five' that takes a 'value' and returns 'str(value) + "5"'. Below the function definition, there are three print statements: 'print(add_five("10"))', 'print(add_five(3))', and 'print(add_five(7))'. The output panel on the right shows the results of these calls: '105', '35', and '75'. The editor also has icons for a code playground, a light/dark theme toggle, a share button, and a 'Run' button.

```
main.py  
1 def add_five(value):  
2     return str(value) + "5"  
3  
4  
5 print(add_five("10"))  
6 print(add_five(3))  
7 print(add_five(7))  
8
```

Output
105
35
75
=== Code

Conclusion

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