AI Assisted coding

ASSIGNMENT 7.1

Name: [E. SAMPATH]

batch no: 12

hall ticket no:2403A51270

QUESTION:

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 to perform structured debugging practices.

Task 1 – Syntax Error: Missing Parentheses in

Print Statement

Buggy Code:

```
# Bug: Missing parentheses in print statement
def greet():
    print "Hello, AI Debugging Lab!"

greet()
```

Explanation:

AI detects a SyntaxError due to the incorrect use of the print statement without parentheses. In

Python 3, print is a function and must be used with parentheses.

Corrected Code:

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

# Test Cases
assert greet() == "Hello, AI Debugging Lab!"
assert isinstance(greet(), str)
assert "AI Debugging" in greet()
```

Output

Hello, AI Debugging Lab!

Task 2 – Logic Error: Using = Instead of ==

Buggy Code:

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

Explanation:

The code uses = (assignment operator) instead of == (comparison operator). In if conditions, ==

is required to check equality. Using = causes a SyntaxError

CORRECT CODE:

```
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"
```

Output:

Ten

Not Ten

Not Ten

Task 3 – Runtime Error: File Not Found

Buggy Code:

```
# 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"))
```

Explanation:

Attempting to open a non-existent file raises a FileNotFoundError. AI suggests using a try-except block to handle this gracefully.

Corrected Code:

```
def read_file(filename):
    try:
        with open(filename, 'r') as f:
            return f.read()
    except FileNotFoundError:
        return "Error: File not found."
    except Exception as e:
        return f"Error: {str(e)}"

# Test Scenarios
print(read_file("existing_file.txt"))  # Assume this file exists
print(read_file("nonexistent.txt"))
print(read_file("/invalid/path/to/file.txt"))
```

Output Examples

<file content if file exists>

Error: File not found.

Error: [Errno 2] No such file or directory: '/invalid/path/to/file.txt'

Task 4 – AttributeError: Calling Undefined

Method

Buggy Code:

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

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

Explanation:

Calling a method drive() which is not defined in the Car class causes an AttributeError. AI

suggests either defining the drive() method or correcting the method call if it was a typo.

Corrected Code (Option 1: Define the method)

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

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

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

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

Output:

Car is driving

Task 5 – TypeError: Mixing Strings and

Integers

Buggy Code:

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

print(add_five("10"))  # TypeError
```

Explanation:

This causes a TypeError because a string ("10") and an integer (5) cannot be added directly. AI

suggests two fixes:

1. Convert string to integer before adding.

2. Convert integer to string for concatenation.

Solution 1: Type Casting

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

# Test Cases
assert add_five("10") == 15
assert add_five(5) == 10
assert add_five("0") == 5

print(add_five("10"))  # Output: 15
print(add_five(5))     # Output: 10
print(add_five("0"))   # Output: 5
```

def add_five(value):

return int(value) + 5

Output

15

10

5

Solution 2: String Concatenation

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

# Test Cases
assert add_five(10) == "105"
```

```
assert add_five("10") == "105"
assert add_five(0) == "05"

print(add_five(10))    # Output: "105"
print(add_five("10"))  # Output: "105"
print(add_five(0))     # Output: "05"
```

Output

105

105

05