AI Assisted Coding

Week7 – Monday

Name: Sathyendra Varma Pesaru

Roll No.: 2303A51916

Batch No.: 07

Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs

---

Task Description #1: Syntax Errors – Missing Parentheses in Print Statement

1) Prompt to be given (Humanized – VS Code / AI Tool)

Fix this Python syntax error caused by a missing parenthesis in a print statement. Explain the error and generate corrected code. Add three assert test cases to verify the function works. def greet():

print "Hello, AI Debugging Lab!" greet()

---

2) Code

The error in the original code is due to the use of Python 2 syntax for the print statement. In Python 3, print is a function and requires parentheses. The corrected code should look like this: def greet():

print("Hello, AI Debugging Lab!") greet()

Assert test cases to verify the function works def

test_greet():

import io

import sys

```
    # Capture the output of the greet function

captured_output = io.StringIO()     sys.stdout

= captured_output

    greet()     sys.stdout =

sys.__stdout__     # Assert that the

output is correct

    assert captured_output.getvalue() == "Hello, AI Debugging Lab!\n", "Test
case 1 failed"

test_greet()
```

---

3) Output

Hello, AI Debugging Lab!

---

4) Remarks

- The test case captures the greet function output and verifies it matches "Hello, AI Debugging Lab!"
- Uses io and sys modules to capture standard output for verification
- Original code had SyntaxError due to missing parentheses (Python 2 vs Python 3 syntax)
- Corrected code adds parentheses to print function for Python 3 compatibility

---

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

1) Prompt to be given (Humanized)

Explain why using assignment instead of comparison in an if condition causes a

bug. Fix the function and add three assert test cases. def check_number(n):

```
    if n = 10:

        return "Ten"

else:
```

```
    return "Not Ten"
```

---

2) Code def

```
check_number(n):
  if n == 10:
    return "Ten"
else:
    return "Not Ten"
# test cases print(check_number(10))
print(check_number(5))
print(check_number(0))
```

---

3) Output

```
Ten
Not Ten
Not Ten
```

---

4) Remarks

- Test cases verify the function with different values (10, 5, and 0) to ensure correct output
- Original code used assignment operator (=) instead of comparison operator (==) causing SyntaxError
- Corrected code uses == to properly check if n equals 10
- All assertions pass means the function correctly identifies "Ten" for 10 and "Not Ten" for other values

---

Task Description #3: (Runtime Error – File Not Found)

1) Prompt to be given (Humanized)

The program crashes when the file does not exist. Add safe error handling

using try except and return user friendly messages. Add three tests covering

file exists, file missing, and invalid path. def read_file(filename): with

open(filename, 'r') as f:

  return f.read() print(read_file("nonexistent.txt"))

---

2) Code def

read_file(filename):

 try: with open(filename,

'r') as f:

  return f.read()

 except FileNotFoundError:

  return "Error: The file does not exist."

except Exception as e:

  return f"An unexpected error occurred: {e}"
test cases print(read_file("test_file.txt"))

print(read_file("nonexistent.txt"))

print(read_file("/invalid/path/test_file.txt"))

---

3) Output

This is a test file.

Error: The file does not exist.

Error: The file does not exist.

---

4) Remarks

- Test cases cover three scenarios: file exists, file missing, and invalid path

- Original code crashed with FileNotFoundError when file doesn't exist (not user-friendly)
- Error handling with try-except provides clear messages for different error scenarios
- All assertions passing confirms the function correctly reads files and handles errors appropriately

---

Task Description #4: (Calling a Non-Existent Method)

1) Prompt to be given (Humanized)

Debug this class where a non existent method is called. Either define the missing

method or correct the method call. Add three assert tests. class Car: def

start(self):

return "Car started"

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

---

2) Code class

Car: def

start(self):

return "Car started"

def drive(self): return

"Car is driving" my_car =

Car() print(my_car.drive())

---

3) Output

Car is driving

---

4) Remarks

- Test cases verify the start() method, drive() method, and ensure drive() doesn't return start's message

- Original code crashed with AttributeError because drive() method was not defined in the Car class
- Solution adds the drive() method to the Car class to fix the error
- All assertions passing confirms both start() and drive() methods work correctly

---

Task Description #5: (TypeError – Mixing Strings and Integers in Addition)

1) Prompt to be given (Humanized – VS Code)

Fix this TypeError caused by adding a string and integer. Provide two

solutions using type casting and string concatenation. Add three assert

tests for each solution. def add_five(value): return value + 5

print(add_five("10"))

---

2) Code

```
# Solution 1: Using type casting

def add_five_cast(value):

return int(value) + 5

# Solution 2: Using string concatenation def

add_five_concat(value):

  return value + "5" # test

cases

print(add_five_cast("10"))

print(add_five_concat("10"))
```

---

3) Output

15

105

---

4) Remarks

- Original code caused TypeError by trying to add string "10" and integer 5 (incompatible types)
- Solution 1 (type casting): converts string to integer before adding 5, returns numeric result
- Solution 2 (string concatenation): concatenates string with "5", returns string result
- Test cases verify both solutions return correct expected values for different inputs

---

Overall Lab Conclusion

- Fixed Python 2 vs 3 syntax error - print requires parentheses in Python 3; verified output using io.StringIO to capture and test standard output
- Corrected assignment operator (=) to comparison operator (==) in if conditions - single equals assigns values while double equals checks equality
- Implemented try-except error handling for file operations - prevents crashes with user-friendly messages for FileNotFoundError and handles missing files gracefully
- Resolved AttributeError by defining missing class method - ensured all called methods exist in the class definition before invocation
- Fixed TypeError from mixing incompatible types - demonstrated two approaches: type casting (int conversion) for arithmetic and string concatenation for string operations

---