

AI Assisted Coding

Assignment – 7.1

M.Sathwik || 2303A51483 || Batch:- 08

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.

```
lab7.py > ...  
1  # Find the error in the following code and provide a corrected version and explanation.  
2  # def greet():  
3  # print "Hello, AI Debugging Lab!"  
4  # greet()  
5  # The error in the above code is that the print statement is not using parentheses, which is required in Python 3.  
6  # The corrected version of the code is as follows:  
7  def greet():  
8      print("Hello, AI Debugging Lab!")  
9  greet()  
10 # Explanation: In Python 3, the print statement was changed to a print function,  
11 # which requires parentheses around the string to be printed.
```

PROBLEMS 26 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Sathwik\Desktop\AI-Lab> & C:/Users/Sathwik/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Sathwik/Desktop/AI-Lab/lab7.py
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.

```
Click to add a breakpoint
13 #fix the error in the condition for the below provided code and explain the error and the fix.
14 # def check_number(n):
15 # if n = 10:
16 # return "Ten"
17 # else:
18 # return "Not Ten"
19 # The error in the above code is that the assignment operator (=) is used instead of the equality operator (==) in the if condition.
20 # The corrected version of the code is as follows:
21 def check_number(n):
22     if n == 10:
23         return "Ten"
24     else:
25         return "Not Ten"
26 # Explanation: In the original code, the if condition is trying to assign the value 10 to n instead of checking if n is equal to 10.
27 # This will result in a syntax error. In the corrected version, we use the equality operator (==) to check if n is equal to 10,
28 # which is the intended behavior.
```

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.

```

3 #Handle the error in the below code by try and except block and explain the error and add the user friendly error message.
4 # def read_file(filename):
5 #     with open(filename, 'r') as f:
6 #         return f.read()
7 #     print(read_file("nonexistent.txt"))
8 # The error in the above code is that it tries to read a file that does not exist, which will raise a FileNotFoundError.
9 # The corrected version of the code with error handling is as follows:
10 def read_file(filename):
11     try:
12         with open(filename, 'r') as f:
13             return f.read()
14     except FileNotFoundError:
15         return f"Error: The file '{filename}' was not found. Please check the filename and try again."
16 print(read_file("nonexistent.txt"))
17 # Explanation: In the original code, if the file "nonexistent.txt" does not exist,
18 # it will raise a FileNotFoundError and the program will crash.
19 # In the corrected version, we wrap the file reading operation in a try-except block to catch the FileNotFoundError.
20 # If the file is not found, we return a user-friendly error message that informs the user about the issue and suggests checking the filename.
21 # This way, the program does not crash and provides helpful feedback to the user.

PROBLEMS 26 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\sathwik\OneDrive\Desktop\AI-Lab> & C:/Users/Sathwik/AppData/Local/Programs/Python/Python313/python.exe c:/Users/sathwik/OneDrive/Desktop/AI-Lab/l.py
Error: The file 'nonexistent.txt' was not found. Please check the filename and try again.
PS C:\Users\sathwik\OneDrive\Desktop\AI-Lab>

```

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.

```

3 #In the below code fix the bug related to the calling an undefined method and explain the error and the fix.
4 # class Car:
5 #     def start(self):
6 #         return "Car started"
7 # my_car = Car()
8 # print(my_car.drive()) #drive() is not defined
9 # The error in the above code is that the method `drive()` is called on the `my_car` object, but it is not defined in the `Car` class.
10 # The corrected version of the code is as follows:
11 class Car:
12     def start(self):
13         return "Car started"
14
15     def drive(self):
16         return "Car is driving"
17 my_car = Car()
18 print(my_car.drive()) #Now drive() is defined
19 # Explanation: In the original code, the `drive` method is called on the `my_car` object,
20 # but since it is not defined in the `Car` class, it will raise an AttributeError.
21 # In the corrected version, we define the `drive()` method within the `Car` class,
22 # which allows us to call it on the `my_car` object without any errors.

PROBLEMS 26 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\sathwik\OneDrive\Desktop\AI-Lab> & C:/Users/Sathwik/AppData/Local/Programs/Python/Python313/python.exe c:/Users/sathwik/OneDrive/Desktop/AI-Lab/l.py
Car is driving
PS C:\Users\sathwik\OneDrive\Desktop\AI-Lab>

```

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.

```
3  #In the below code there is a TypeError due to mixing string and integer. You Fix the error and add that numbers.
4  # def add_five(value):
5  #     return value + 5
6  # print(add_five("10"))
7  # The error in the above code is that it tries to add an integer (5) to a string ("10"), which will raise a TypeError.
8  # The corrected version of the code is as follows:
9  def add_five(value):
10     try:
11         return int(value) + 5
12     except ValueError:
13         return f"Error: The value '{value}' is not a valid number. Please provide a numeric value."
14 print(add_five("10")) #This will now return 15
15 # Explanation: In the original code, the function `add_five` attempts to add an integer (5) to a string ("10"),
16 # which is not allowed in Python and results in a TypeError.
17 # In the corrected version, we convert the input `value` to an integer using `int(value)`.
```

PROBLEMS 26 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab> & C:/Users/Sathwik/AppData/Local/Programs/Python/Python313/python.exe c:/Users/Sathwik/OneDr
py
15
PS C:\Users\Sathwik\OneDrive\Desktop\AI-Lab>