AI ASSISTED CODING -7.3

2303A51878

B-28

Task 1: Fixing Syntax Errors

Scenario :- You are reviewing a Python program where a basic function definition contains a syntax error.

Original Code (with Syntax Error)

```
def add(a,b)
    return a + b
```

Problem:
Python requires a colon : at the end of a function definition line. Without it, the interpreter raises a SyntaxError.

AI Detects the Syntax Error

When this code is analyzed, the AI (or Python interpreter) detects:

```
PS C:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1> & C:/Users/kotes/AppData/Local/
rs/kotes/OneDrive/Desktop/AI-AC/Week-2.1/week.7.3/Task1py
  File "c:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1\week.7.3\Task1py", line 1
    def add(a,b)
                ^
SyntaxError: expected ':'
PS C:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1>
```

What AI notices:

- def add(a, b) is a function header
- Function headers must end with :
- The missing colon breaks Python's syntax rules

AI-Corrected Code (Fixed Version)

After correction, the function becomes:

```
def add(a, b):
    return a + b
```

AI-Generated Explanation of the Fix

The syntax error occurred because Python function definitions must end with a colon (:). The colon indicates the start of the function body. Adding the missing colon resolves the SyntaxError and allows the function to execute properly.

Task 2: Debugging Logic Errors in Loops

Scenario:- You are debugging a loop that runs infinitely due to a logical mistake.

Original Code (Logic Error → Infinite Loop)

```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1
```

What's wrong?

- The statement i - 1 does not change the value of i
- i always remains 1
- Condition i <= 5 is always true
- Result → infinite loop

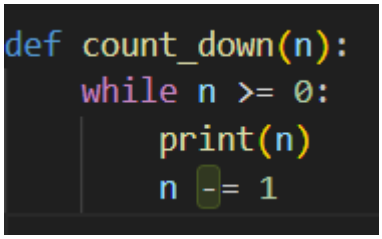
AI Identifies the Cause of Infinite Iteration

AI Analysis:

- The loop condition depends on i

- i is never updated inside the loop
- Expression i - 1 performs a calculation but does not assign the result back to i
- Therefore, the loop never progresses toward termination

AI-Fixed Code (Corrected Loop Logic)



What changed?

- i += 1 properly increments the loop variable
- The loop now moves toward the stopping condition

Analysis of Corrected Loop Behavior

How the fixed loop works:

- Iteration 1 → i = 1
- Iteration 2 → i = 2
- Iteration 3 → i = 3
- Iteration 4 → i = 4
- Iteration 5 → i = 5
- Loop exits when i = 6

AI Explanation of the Logic Error

The infinite loop occurred because the loop variable was never updated. The expression i - 1 does not modify the value of i. Replacing it with i += 1 correctly updates the variable, allowing the loop condition to eventually become false and the loop to terminate.

Task 3: Handling Runtime Errors (Division by Zero)

Scenario :- A Python function crashes during execution due to a division by zero error.

Original Code (No Validation → Runtime Error)

```
def divide(a, b):
    return a / b
print(divide(10, 0))  # This will raise a ZeroDivisionError
```

AI Identifies the Runtime Error

```
PS C:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1> & C:/Users/kotes/AppData/Local/Programs/Python/P
rs/kotes/OneDrive/Desktop/AI-AC/Week-2.1/week.7.3/Task1py
  File "c:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1\week.7.3\Task1py", line 5, in <module>
    print(divide(10, 0))  # This will raise a ZeroDivisionError
          ~~~~~~^^^^^^^
  File "c:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1\week.7.3\Task1py", line 4, in divide
    return a / b
           ~~^~~
ZeroDivisionError: division by zero
```

AI Diagnosis:

- The function performs division without checking the divisor
- When b equals 0, Python raises a ZeroDivisionError
- This is a runtime error, not a syntax or logic error, because the code is syntactically correct but fails during execution

AI-Fixed Code (Using try–except for Safety)

```
def divide(a, b):
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed"
# Example usage:
print(divide(10, 2))  # Output: 5.0
print(divide(10, 0))  # Output: Error: Division by zero is not allowed
```

AI-Generated Explanation of Runtime Error Handling

The runtime error occurred because division by zero is mathematically undefined and not allowed in Python.
Wrapping the division operation in a try-except block allows the program to catch the ZeroDivisionError.
This prevents the application from crashing and enables graceful error handling by returning a meaningful message.

Task 4: Debugging Class Definition Errors

Scenario :- You are given a faulty Python class where the constructor is incorrectly defined.

Original Code (Faulty Class Definition)

```
class Rectangle:
    def __init__(length, width):
        self.length = length
        self.width = width
```

Problem

- The constructor __init__() is **missing the self parameter**
- Python automatically passes the object reference as the first argument
- This leads to a runtime error when creating an object

AI Identifies the Error

AI Detection:

- Instance methods must have self as the first parameter
- Without self, Python cannot bind attributes to the object

AI-Corrected Code (Fixed Version)

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

AI Explanation of the Fix (Detailed)

The error occurred because the __init__() method in the Rectangle class did not include the self parameter as its first argument.

In Python, self represents the current instance of the class and is automatically passed when an object is created. Without self, Python cannot associate the parameters length and width with the object.

As a result, the constructor receives an unexpected number of arguments, leading to a TypeError.

By adding self as the first parameter and using self.length and self.width, the values are correctly stored as instance variables. This allows each Rectangle object to maintain its own data and ensures proper object-oriented behavior.

Task 5: Resolving Index Errors in Lists

Scenario:- A program crashes when accessing an invalid index in a list.

Original Code (Out-of-Range Index Error)

```
numbers = [1, 2, 3]

print(numbers[5])
```

Problem

- The list has only 3 elements
- Valid indexes are 0, 1, 2
- Accessing index 5 causes a runtime crash

AI Identifies the Index Error

AI Detection:

- The program tries to access a list index that does not exist
- Python raises an IndexError when an index is outside the valid range

```
PS C:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1> & C:/Users/kotes/AppData/Local/Programs/Pyt
rs/kotes/OneDrive/Desktop/AI-AC/Week-2.1/week.7.3/Task1py
Traceback (most recent call last):
  File "c:\Users\kotes\OneDrive\Desktop\AI-AC\Week-2.1\week.7.3\Task1py", line 5, in <module>
    print(numbers[5])
          ~~~~~~~^^^
IndexError: list index out of range
```

AI Suggests Safe Access Methods

The AI recommends two safe approaches:

1. **Bounds checking using len()**
2. **Exception handling using try-except**

AI-Corrected Code (Using Bounds Checking)

```
numbers = [10, 20, 30]
index = 5

if index < len(numbers):
    print(numbers[index])
else:
    print("Error: Index out of range")
```

Alternative AI Fix (Using try-except)

```
numbers = [10, 20, 30]

try:
    print(numbers[5])
except IndexError:
    print("Error: Index out of range")
```

AI Explanation of the Fix

The error occurred because the program attempted to access an index that is outside the valid range of the list.
Python lists are zero-indexed, meaning the last valid index is always one less than the list length.
By applying bounds checking or handling the IndexError with a try-except block, the program safely prevents crashes and ensures reliable list access.