WEEK-ASS-7.3

2303A51494

Batch-27

Task-1:

Task 1: Fixing Syntax Errors
Scenario
You are reviewing a Python program where a basic function definition contains a syntax error.
Requirements
• Provide a Python function add(a, b) with a missing colon
• Use an AI tool to detect the syntax error
• Allow AI to correct the function definition
• Observe how AI explains the syntax issue
Expected Output
• Corrected function with proper syntax
• Syntax error resolved successfully
• AI-generated explanation of the fix

Prompt:

"The following Python function has a syntax error because a colon is missing. Please identify the syntax error, correct the function, and briefly explain what was wrong."

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

Code:

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

File "/tmp/ipython-input-3843817783.py", line 1
    def add(a,b)
                ^
SyntaxError: expected ':'
```

Use an AI tool to detect the syntax error:

```python
[2]  ▶  def add(a,b):
✓ 0s         return a+b
```

Allow AI to correct the function definition:

Fixed Version (correct syntax)

python   ⧉ Copy code

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

Explanation:

The error occurs because the function definition is missing a colon (:) at the end of the def line.
In Python, a colon is required to indicate the start of the function body.
Without the colon, Python raises a Syntax Error and cannot execute the code.

Task-2:

Task 2: Debugging Logic Errors in Loops
Scenario
You are debugging a loop that runs infinitely due to a logical mistake.

```python
python

def count_down(n):
    while n >= 0:
        print(n)
        n += 1   # Should be n -= 1
```

Requirements
• **Provide a loop with an increment or decrement error**
• **Use AI to identify the cause of infinite iteration**
• **Let AI fix the loop logic**
• **Analyse the corrected loop behaviour**
Expected Output
• **Infinite loop issue resolved**
• **Correct increment/decrement logic applied**
• **AI explanation of the logic error**

Prompt:

"The following Python loop runs infinitely due to a logical error. Please identify the cause of the infinite loop, fix the loop logic, and explain what was wrong."

Code:

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


n = 5
count_down(n)
```

```
5
4
3
2
1
0
```

Logical error:

explain the logical error

The logical error in the original `count_down` function was within the
`while` loop. The line `n+=1` was causing `n` to increase in each iteration.
Since the loop condition was `while n>=0`, if `n` started as a non-negative
number, it would always remain non-negative (or become positive), leading
to an infinite loop. For a countdown, `n` should be decreased with `n-=1`.

Explanation:

**The loop was intended to count from 1 to 5, but it contained a logical mistake
in the increment/decrement statement.**
**The variable i was decremented (i -= 1) instead of incremented, causing its
value to move further away from the loop's stopping condition.**

Because i kept decreasing, the condition i <= 5 always remained true, resulting in an infinite loop.

The issue was fixed by changing the decrement to an increment (i += 1), allowing the loop variable to progress toward the termination condition. After the fix, the loop executes correctly and stops after printing numbers from 1 to 5.

Task-3:

Task 3: Handling Runtime Errors (Division by Zero) Scenario

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

```python
# Debug the following code
def divide(a, b):
    return a / b


print(divide(10, 0))
```

Requirements
• **Provide a function that performs division without validation**
• **Use AI to identify the runtime error**
• **Let AI add try-except blocks for safe execution**
• **Review AI's error-handling approach**
Expected Output
• **Function executes safely without crashing**
• **Division by zero handled using try-except**
• **Clear AI-generated explanation of runtime error handling**

Prompt:

'The following Python function crashes at runtime due to a division by zero error.
Please identify the runtime error, modify the function using `try-except` to handle it safely, and explain how the error handling works."

Code and Output:

```python
def divide(a, b):
    return a / b

divide(10, 0)
```

```python
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."

divide(10, 0)
```

...   'Error: Division by zer

The code in cell `URBMDOaST3HZ` executed successfully! Your `divide` function, with its `try-except` block, correctly handled the attempt to divide by zero. Instead of crashing, it returned the message: `'Error: Division by zero is not allowed.'` as expected.

👍 👎

Explanation:

- **The function attempts to divide two numbers without checking the divisor.**

- **When the function is called with b = 0, Python raises a ZeroDivisionError.**

- **This error occurs at runtime, not during function definition.**

- **The program crashes because division by zero is mathematically invalid.**

- **A try-except block is used to handle this error safely.**

- **The risky division operation is placed inside the try block.**

- **The except ZeroDivisionError block catches the error.**

- **Instead of crashing, the function returns a meaningful error message.**

- **This ensures the program continues running without interruption.**

Task-4:

Resolving Index Errors in Lists

Scenario

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

Requirements

• **Provide code that accesses an out-of-range list index**

• **Use AI to identify the Index Error**

• **Let AI suggest safe access methods**

• **Apply bounds checking or exception handling**

Expected Output

• **Index error resolved**

• **Safe list access logic implement**

```python
numbers = [1, 2, 3]
print(numbers[5])
```

Requirements

• **Provide code that accesses an out-of-range list index**

• **Use AI to identify the Index Error**

• **Let AI suggest safe access methods**

• **Apply bounds checking or exception handling**

Expected Output

• **Index error resolved**

• **Safe list access logic implementedPrompt:**

'The following Python code causes a runtime error.
Please identify the error, explain why it occurs, and provide a corrected version that safely handles the issue."

Code and Output:

```
numbers=[1,2,3]
print(numbers[5])
```

```
---------------------------------------------------------------
IndexError                        Traceback (most recent call last)
/tmp/ipython-input-3809996345.py in <cell line: 0>()
      1 numbers=[1,2,3]
----> 2 print(numbers[5])

IndexError: list index out of range
```
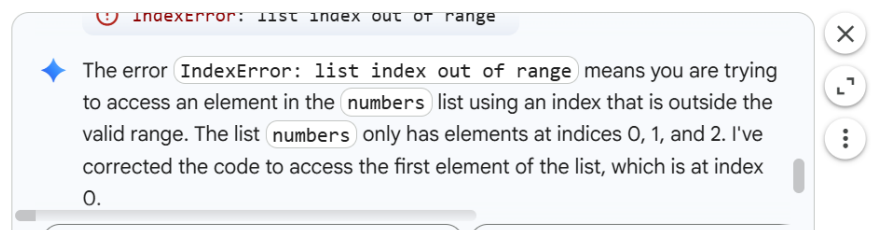
Next steps: **Explain error**

Correct code:

```
numbers=[1,2,3]
print(numbers[0])
```

```
1
```

⚠ IndexError: list index out of range

✦ The error `IndexError: list index out of range` means you are trying to access an element in the `numbers` list using an index that is outside the valid range. The list `numbers` only has elements at indices 0, 1, and 2. I've corrected the code to access the first element of the list, which is at index 0.

Explanation:

- **The list numbers contains three elements.**

- **Python uses zero-based indexing.**

- **The valid indexes for this list are 0, 1, and 2.**

- **Index 5 does not exist in the list.**

- **Attempting to access an invalid index causes an Index Error.**

- **This error occurs during program execution (runtime).**

- **The program crashes because the index is outside the list's range.**

- **Proper index validation or error handling can prevent this issue.**

Task-5:

Debugging Class Definition Errors
Scenario
You are given a faulty Python class where the constructor is incorrectly defined.

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

Requirements
- **Provide a class definition with missing self-parameter**
- **Use AI to identify the issue in the __init__() method**
- **Allow AI to correct the class definition**
- **Understand why self is required**

Expected Output
- **Corrected __init__() method**
- **Proper use of self in class definition**
- **AI explanation of object-oriented errorPrompt:**

Prompt:

'The following Python class has an error in its constructor.
Please identify the issue in the `__init__()` method, correct the class definition, and explain why the `self` parameter is required."

Code and Output:

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

```
    File "/tmp/ipython-input-781339241.py", line 2
        def __init__(length,width):
                                   ^
    SyntaxError: invalid syntax
```

Next steps: (Explain error)

Correct code:

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

You're getting a `SyntaxError` because there's no space between `def` and `__init__`. Also, the `__init__` method needs `self` as its first parameter to refer to the instance of the class. I've corrected these for you.

The code in cell `vYpQa8QpY-ka` executed successfully! The `Rectangle` class has been defined with an `__init__` method that takes `self`,

(Explain the `IndexError` again with a different index) (Show me how to access other elements)

Explanation:

- **The __init__() method is missing the self parameter.**

- **In Python, self represents the current object instance.**

- **Python automatically passes the object to instance methods as self.**

- **Without self, Python cannot bind variables to the object.**

- **Statements like name = name only create local variables.**

- **Instance variables must be assigned using self.variable_name.**

- **Missing self causes the class to behave incorrectly or raise errors.**

- **Adding self allows proper initialization of object attributes.**