AI ASSISTED CODING ASS-7.3

2303A51406

B-27

Task 1: Fixing Syntax Errors

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

Python function with Syntax Error:

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

```
/HARSHAVARDHAN/Desktop/AI Assitant Coding/addition.py"
  File "c:\Users\HARSHAVARDHAN\Desktop\AI Assitant Coding\addition.py", line 1
    def add(a, b)
                 ^
SyntaxError: expected ':'
```

Corrected Python function:

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

Explanation of the fix:

Error: the function header was missing the required ":" which causes a SyntaxError because Python's parser expects a colon to start the function suite.

Fix: add ":" after def and ensure the body is indented.

Task 2: Debugging Logic Errors in Loops

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

Given Code:

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

AI Corrected Code:

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

Explanation:

Error: The loop uses n += 1 which increments n, causing an infinite loop instead of counting down. The condition while n >= 0 will always be true.

Fix: Change n += 1 to n -= 1 to decrement n each iteration.

The count_down function should decrease n by 1 each iteration until it reaches below 0. Using += does the opposite, making the loop infinite since n keeps growing and will always be >= 0.

Task 3: Handling Runtime Errors (Division by Zero)

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

Given Code:

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

print(divide(10, 0))  # This will raise a ZeroDivisionError
```

```
  File "c:\Users\HARSHAVARDHAN\Desktop\AI Assitant Coding\main.py", line 12, in <module>
    print(divide(10, 0))  # This will raise a ZeroDivisionError
          ~~~~~~^^^^^^^
  File "c:\Users\HARSHAVARDHAN\Desktop\AI Assitant Coding\main.py", line 10, in divide
    return a / b
           ~~^~~
ZeroDivisionError: division by zero
```

AI Corrected Code:

```
def divide(a, b):
    if b == 0:
        return "Error: Cannot divide by zero"
    return a / b

print(divide(10, 0))  # Returns error message instead of crashing
```

Explanation:

Error: Division by zero causes a ZeroDivisionError when b = 0. The code attempts to divide 10 by 0, which is mathematically undefined.

Fix: Add error handling using a try-except block to catch the exception.

Before performing division, check if b is 0. If it is, return an error message instead of attempting the division. This prevents the ZeroDivisionError from crashing your program. Alternatively, you can use try-except for more complex error handling.

Task 4: Debugging Class Definition Errors

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

Given Code:

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

AI Corrected Code:

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

Explanation:

Error: The __init__ method is missing self as the first parameter. This causes a TypeError when creating a Rectangle instance.

Fix: Add self as the first parameter.

All instance methods in Python classes must have self as the first parameter. self refers to the instance being created and allows you to set instance attributes like self.length and self.width. Without it, the method doesn't know which object's attributes to modify.

Task 5: Resolving Index Errors in Lists

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

Given Code:

```
numbers = [1, 2, 3]
print(numbers[5])  # This will raise an IndexError
```

```
  File "c:\Users\HARSHAVARDHAN\Desktop\AI Assitant Coding\main.py", line 24, in <module>
    print(numbers[5])  # This will raise an IndexError
          ~~~~~~~^^^
IndexError: list index out of range
```

AI Corrected Code:

```
numbers = [1, 2, 3]
#Access valid index
print(numbers[2])  # Prints 3
```

Explanation:

Error: IndexError - attempting to access index 5 in a list that only has 3 elements (indices 0, 1, 2). Array indices are out of bounds.

Fix: Check the list length or use valid index values.

Lists in Python are zero-indexed, so a list with 3 elements has valid indices 0, 1, and 2. Accessing index 5 doesn't exist, causing an IndexError. Always verify the index is within bounds before accessing, or use error handling with try-except.