

## LAB ASSIGNMENT 7.3

NAME: K.SANJANA

ROLL.NO:203A52139

BATCH:41

### TASK1: Fixing Syntax Errors

FIXING:

The screenshot shows a code editor interface with a dark theme. At the top, there are buttons for 'Generate code', 'Add Context...', 'GPT-5 mini', and 'Keep' / 'Undo'. The code area contains the following Python code:

```
1 def add(a, b):
2     return a + b
3
4
5
```

The cursor is positioned at the end of the first line, where a colon is expected after 'def'. A red squiggle underlines the colon, indicating a syntax error.

EXPLANATION:

The screenshot shows the same code editor after the error has been corrected. The code now includes a colon after the 'def' keyword:

```
# ...existing code...
def add(a, b):
    return a + b
# ...existing code...
```

Below the code, a message reads: "Save the file and run (e.g.) `python ass7.3.py`. Check for consistent indentation (no mixed tabs/spaces).". There are also icons for copy, paste, and refresh, along with the text "GPT-5 mini • 0x".

**JUSTIFICATION:** This task focuses on identifying and correcting syntax errors, which prevent a program from running at all. The missing colon in the function definition causes Python to raise a `SyntaxError` because the interpreter cannot understand the structure of the code. By using AI to detect and correct this issue, the task demonstrates how AI can quickly locate formatting mistakes and restore proper program execution. This improves code reliability and helps beginners understand Python's syntax rules.

### TASK2: Debugging Logic Errors in Loops

FIXING:

The screenshot shows a code editor with a debugger interface. The code is a simple 'count down' script:

```
ss7.3.py > count_down
def count_down(n):
    while n>=0:
        print(n)
        n+=1
    ↴ -
```

A red arrow points to the line `n+=1`, indicating a potential error. The debugger interface shows a stack trace or variable state below the code.

EXPLANATION:

Comment 1 of 1

**Code Review**

The function is named 'count\_down' but increments n instead of decrementing it. This creates an infinite loop since n will always be  $\geq 0$ . Change 'n+=1' to 'n-=1' to correctly count down.

Suggested change:

```
-     n+=1
+     n-=1
```

**Apply** | **Discard** | **▼**

**JUSTIFICATION:** The purpose of this task is to handle logic errors that do not crash the program but cause incorrect behavior, such as infinite loops. The loop failed because the loop variable was not updated, so the termination condition was never reached. Using AI to analyze and fix the logic ensures the loop progresses correctly and terminates as expected. This task highlights the importance of controlling loop conditions and shows how AI assists in preventing performance issues caused by endless execution.

### TASK3: Handling Runtime Errors (Division by Zero)

#### FIXING:

**GPT-5 mini**

```
def divide(a,b):
    return a/b
print(divide(10,0))
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return None

result = divide(10, 0)
if result is None:
    print("Error: division by zero")
else:
    print(result)
```

**Keep** | **Undo** | **↻**

#### EXPLANATION:

Comment 1 of 3

**Code Review**

Division by zero is not handled. This function will raise a ZeroDivisionError when b is 0. Add validation to check if b is zero before performing division, or handle the exception appropriately.

Suggested change:

```
+     if b == 0:
+         raise ZeroDivisionError("Cannot divide by zero")
```

**Apply and Go to Next** | **Discard and Go to Next** | **▼**

Comment 1 of 2

**Code Review**

Missing spaces after commas in function parameters. Should be 'def divide(a, b):' according to PEP 8 style guidelines.

Suggested change:

```
- def divide(a,b):
-     return a/b
- print(divide(10,0))
+ def divide(a, b):
+     return a/b
+ print(divide(10, 0))
```

---

Comment 1 of 1

**Code Review**

Function lacks a docstring explaining its purpose, parameters, return value, and potential exceptions (like ZeroDivisionError). Add a docstring to improve code maintainability.

Suggested change:

```
+     """Divide one number by another.
+
+     Args:
+         a: The dividend.
+         b: The divisor.
+
+     Returns:
+         The result of dividing `a` by `b`.
+
+     Raises:
+
```

---

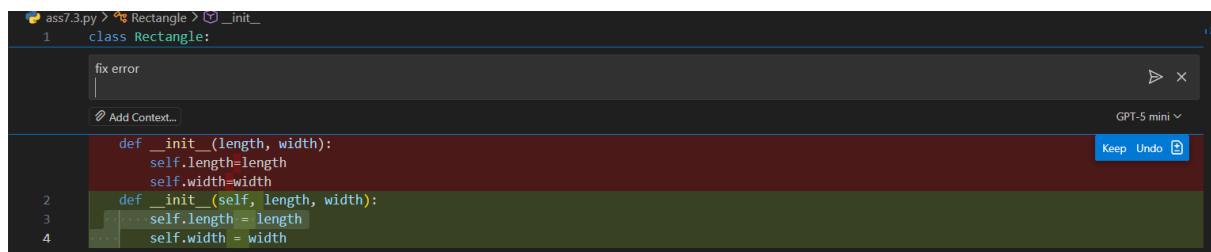
**Apply** | **Discard** | **⌄**

## JUSTIFICATION:

This task addresses runtime errors that occur during execution, specifically division by zero. Such errors crash the program if not handled properly. By applying AI-generated error handling using try-except, the program becomes more robust and user-friendly. This task justifies the use of exception handling to maintain program stability and demonstrates how AI can help programmers build safer, fault-tolerant applications.

## TASK4: Debugging Class Definition Errors

### FIXING:



The screenshot shows a code editor interface with a dark theme. A tooltip window is open over a piece of Python code. The code defines a class `Rectangle` with two `__init__` methods. The first method takes `length` and `width` as arguments and initializes `self.length` and `self.width`. The second method also takes `length` and `width` and initializes `self.length` and `self.width`. A tooltip window titled "fix error" contains the text "def \_\_init\_\_(self, length, width): self.length = length self.width = width". Below the code, there are buttons for "Keep" and "Undo".

### EXPLANATION:

Workspace

- > Used 1 reference
- ✓ Finalizing message structure

Add `self` as the first parameter of `__init__`. Fix and save:

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

GPT-5 mini • 0x

### JUSTIFICATION:

The goal of this task is to correct mistakes in object-oriented programming concepts, especially in class constructors. Without the `self` parameter, Python cannot bind variables to the object instance. AI helps identify this conceptual error and fixes it, ensuring proper object creation and data storage. This task reinforces understanding of how classes work in Python and shows how AI supports debugging structural and design-level errors.

### TASK5: Resolving Index Errors in Lists

#### FIXING:

```
fix the code
∅ Add Context...
GPT-5 mini ▾
Keep Undo ⌛
numbers=[1,2,3]
print(numbers[5])
1 numbers = [1, 2, 3]
2 index = 5
3
4 try:
5     print(numbers[index])
6 except IndexError:
7     print(f"Index {index} out of range (list length {len(numbers)}).")
```

#### EXPLANATION:

Comment 1 of 1

Code Review

Attempting to access index 5 in a list with only 3 elements (indices 0-2) will raise an `IndexError`. The valid indices for this list are 0, 1, and 2. If you intended to access the last element, use `numbers[2]` or `numbers[-1]`.

Suggested change:

```
- print(numbers[5])
+ print(numbers[-1])
```

Apply Discard ↴

### JUSTIFICATION:

This task focuses on preventing index-related runtime errors when accessing lists. Attempting to use an invalid index causes the program to crash. By using AI to apply bounds

checking or exception handling, the program safely accesses list elements. This task justifies defensive programming practices and demonstrates how AI can help avoid crashes, improve reliability, and ensure programs handle unexpected input gracefully