

## LAB ASSIGNMENT-7.3

NAME: SINDHU KATAKAM

ENROLLEMNT NUMBER: 2303A52481

BATCH: 41

### Task 1: Fixing Syntax Errors

FIX:

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

EXPLAIN:

@workspace /explain Expected ":"

Workspace

> Used 1 reference

✓ Fixing syntax errors

The error is coming from this line:

- def add(a, b) → in Python, a function header **must** end with :.

JUSTIFICATION:

The syntax error happened because the function definition line did not end with a colon. In Python, a colon tells the interpreter that an indented block of code will follow. Adding the colon fixes the structure and allows the function to run correctly.

## Task 2: Debugging Logic Errors in Loops

**FIX:**

```
def count_down(n):
    while n>=0:
        print(n)
        n+=1
        n -= 1
print(count_down(5))
```

Keep Undo 

**REVIEW:**

### Code Review

The function increments n instead of decrementing it, causing an infinite loop. For a countdown, change n+=1 to n-=1.

Suggested change:

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

**JUSTIFICATION:**

The infinite loop happened because the loop variable was not incremented or decremented properly. Updating the variable correctly ensures the loop condition will eventually become false, preventing infinite execution.

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

**FIX:**

```
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)
print("Cannot divide by zero" if result is None else result)
```

## REVIEW:

Comment 1 of 3

 **Code Review**  

Division by zero is not handled. This will raise a ZeroDivisionError when b is 0. Add validation to check if b is zero before performing division, or wrap the operation in a try-except block.

Suggested change:

```
-     return a/b
+     if b == 0:
+         raise ValueError("Division by zero is not allowed.")
+     return a / b
```

 **Code Review**  

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

Suggested change:

```
- def divide(a,b):
+ def divide(a, b):
```

 **Code Review**  

```
+     """
+     Divide one number by another.
+
+     Parameters:
+         a (float or int): The dividend.
+         b (float or int): The divisor.
+
+     Returns:
+         float: The result of dividing `a` by `b`.
+
+     Raises:
+         ZeroDivisionError: If `b` is zero.
+     """
```

## JUSTIFICATION:

In this task, a function caused an error because a number was divided by zero. This error happens while the program is running, so it is called a runtime error. To fix this problem, a try-except block was used to handle the error safely. The program now shows an error message instead of crashing. This makes the program more reliable and user-friendly.

## Task 4: Debugging Class Definition Errors

### MODIFY:

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

  

## EXPLAIN:

SYNTAX ERROR: EXPECTED COLON ISSUE

Workspace

Used 1 reference

Explaining coding issues with `_init_`

Your `_init_` signature is missing the `self` parameter. In instance methods, Python passes the instance as the **first** argument; by convention it's named `self`. Since your method is defined as `def _init_(length, width):`, the name `self` inside the body is indeed undefined.

Fix it like this:

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

Gotcha: with your original code, calling `Rectangle(10, 5)` would bind `length=10`, `width=5`, and there's no variable named `self` to assign attributes onto.

## REVIEW:

Code Review

The `__init__` method is missing the 'self' parameter as its first argument. It should be 'def `__init__(self, length, width)`'. Without this, the method will fail when instantiating a Rectangle object.

Suggested change:

```
-     def __init__(length,width):
+     def __init__(self, length, width):
```

## JUSTIFICATION:

In this task, the class constructor was written incorrectly because the `self` parameter was missing. The `self` keyword is important because it allows the object to store and access its own data. Without `self`, the class cannot work properly. After adding `self`, the class works correctly and can create objects. This helps in understanding how object-oriented programming works in Python.

## Task 5: Resolving Index Errors in Lists

### MODIFY:

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

# Safely access an index that may be out of range
index = 5
if 0 <= index < len(numbers):
    print(numbers[index])
else:
    print(f"Index {index} is out of range for list of length {len(numbers)}")
```

## REVIEW:

 **Code Review** thumb up thumb down

Index out of range error: The list 'numbers' has only 3 elements (indices 0-2), but the code attempts to access index 5. This will raise an IndexError at runtime. Consider using a valid index (0, 1, or 2) or adding bounds checking before accessing the list.

---

Suggested change:

```
- print(numbers[5])
-
+ if len(numbers) > 5:
+     print(numbers[5])
+ else:
+     print("Index 5 is out of range for list 'numbers'")
```

## JUSTIFICATION:

In this task, an index error occurred when trying to access a list element that does not exist. This happens when the index number is outside the range of the list. The issue was fixed by using exception handling or checking the list length before accessing the index. This prevents the program from crashing. It helps in writing safer and more stable code.