

# ASSIGNMENT-7.2.

Roll no:2303A52410  
Batch-38

## Task 1: Fixing Syntax Errors

### Expected Output

The Python function is corrected by adding the missing colon after the function definition.

### Prompt

Give the above code a proper syntax

### Code

```
1  def add(a, b)
2      return a + b
3
4  #give the above code a proper syntax
5  def add(a, b):
6      return a + b
```

### Explanation

The error occurs because Python requires a colon (:) at the end of a function definition to indicate the start of the function body, and adding this colon fixes the syntax and allows the code to execute correctly.

## Task 2: Debugging Logic Errors in Loops

### Expected Output

The infinite loop is identified and corrected.

The loop logic is fixed by using the correct decrement operation.

### Prompt

Fix the above code to count down from n to 0

### Code

```
8  def count_down(n):
9      while n >= 0:
10         print(n)
11         n += 1
12  #fix the above code to count down from n to 0
13  def count_down(n):
14      while n >= 0:
15         print(n)
16         n -= 1
```

#### Explanation

The original loop increased the value of n, so the loop never stopped.

By changing it to decrease n, the loop now reaches zero and ends correctly.

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

#### Expected Output

The program handles division by zero without crashing.

An error message will be displayed safely.

#### Prompt

Fix the above code to handle division by zero

#### Code

```
19 def divide(a, b):
20     return a / b
21
22 print(divide(10, 0))
23 #fix the above code to handle division by zero
24 def divide(a, b):
25     if b == 0:
26         return "Error: Division by zero is not allowed."
27     return a / b
28 print(divide(10, 0))
29
```

#### Output

```
Error: Division by zero is not allowed.
```

#### Explanation

The code checks for division by zero before performing the operation, preventing the program from crashing.

## Task 4: Debugging Class Definition Errors

### Expected Output

The constructor is corrected by adding the missing self parameter.  
The class now properly assigns values to object attributes.

### Prompt

Fix the above code to properly initialize the Rectangle class

### Code

```
class Rectangle:
    def __init__(length, width):
        self.length = length
        self.width = width
#fix the above code to properly initialize the Rectangle class
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
    def area(self):
        return self.length * self.width
#user defined function to calculate area of rectangle
rect = Rectangle(5, 10)
print("Area of rectangle:", rect.area())
```

### Output

Area of rectangle: 50

### Explanation

The constructor was missing the self parameter, which caused the class to be incorrect. Adding self allows the class to store values inside the object. This helps understand which object the variables belong to.

## Task 5: Resolving Index Errors in Lists

### Expected Output

The IndexError is identified and resolved.  
The program safely accesses the list without crashing.

### Prompt

Fix the above code to handle index out of range error

#### Code

```
46 numbers = [1, 2, 3]
47 print(numbers[5])
48 #fix the above code to handle index out of range error
49 numbers = [1, 2, 3]
50 try:
51     print(numbers[5])
52 except IndexError:
53     print("Error: Index out of range.")
54
```

#### Output

```
Error: Index out of range.
```

#### Explanation

The error occurs because the program tries to access a list index that does not exist. By checking the list length or using exception handling, the program avoids the crash.