# ASSIGNMENT-7

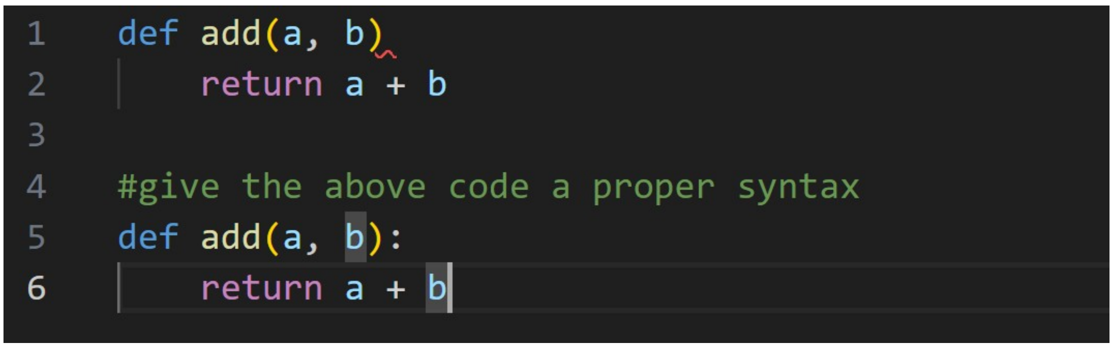
**Roll no:2303A52074**
**Batch-37**

## 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 synatx

**Code**



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

#give the above code a proper syntax
def add(a, b):
    return a + b
```

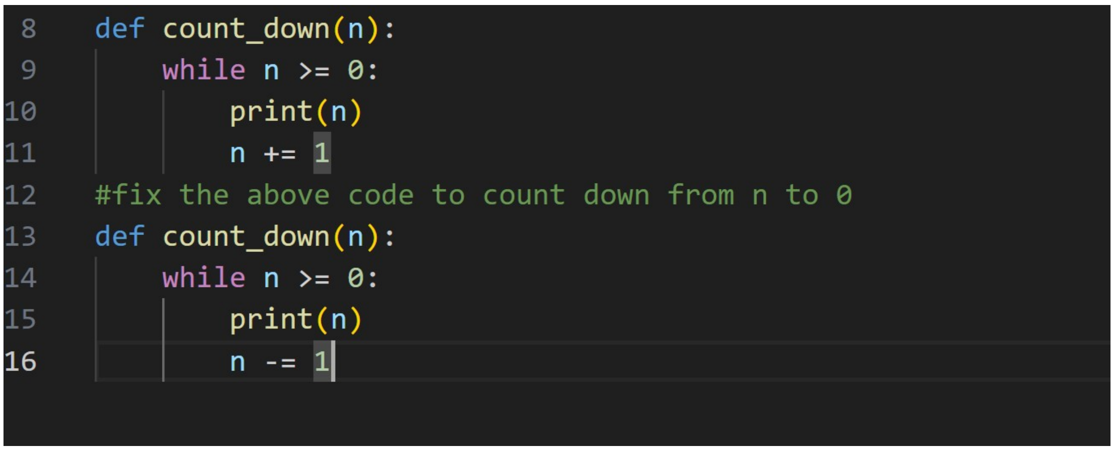
**Explanation**
The error occurs because Python requires a colon (:) at the end of a function definition to indicate the start of the function body, and adding this colon fixes the syntax and allows the code to execute correctly.

## 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**



```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1
#fix the above code to count down from n to 0
def count_down(n):
    while n >= 0:
        print(n)
        n -= 1
```

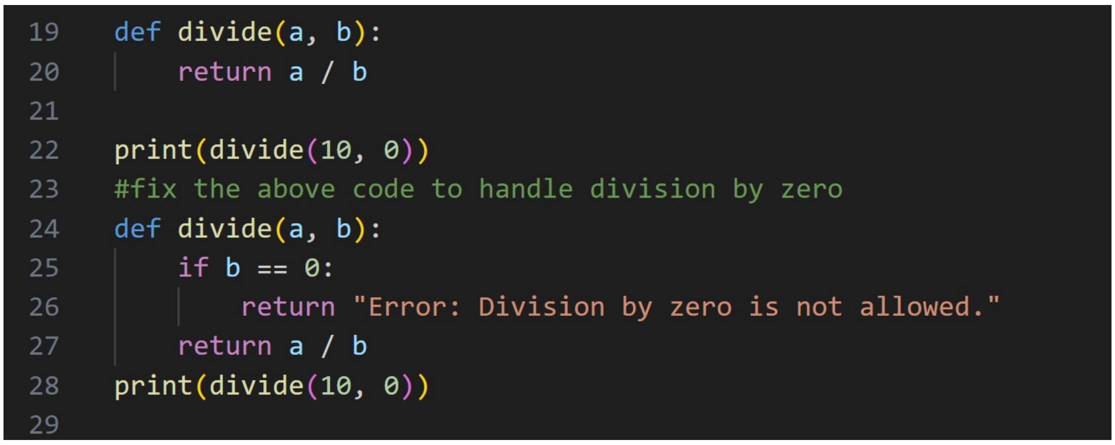
**Explanation**
The original loop increased the value of n, so the loop never stopped.
By changing it to decrease n, the loop now reaches zero and ends correctly.

## 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**

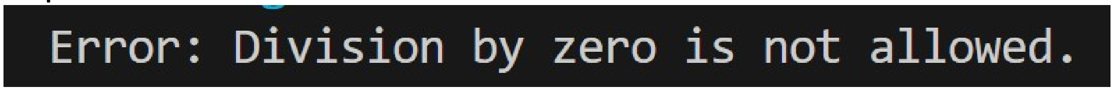


```python
def divide(a, b):
    return a / b

print(divide(10, 0))
#fix the above code to handle division by zero
def divide(a, b):
    if b == 0:
        return "Error: Division by zero is not allowed."
    return a / b
print(divide(10, 0))

```

**Output**



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

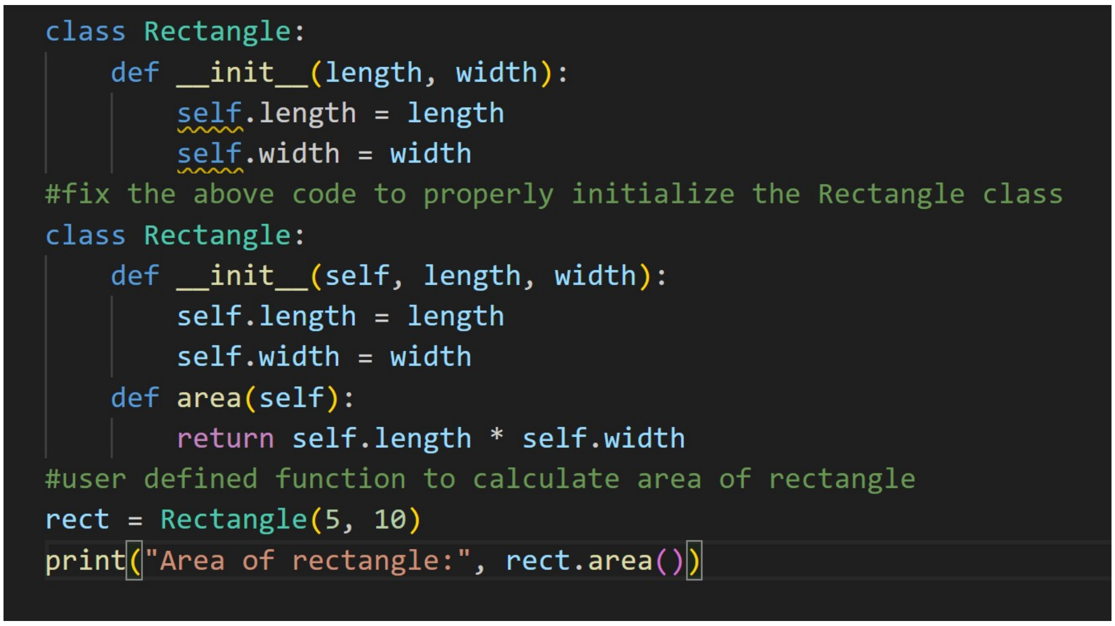
**Explanation**
The code checks for division by zero before performing the operation, preventing the program from crashing.

## 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.

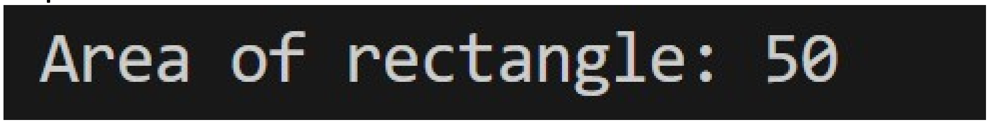
**Prompt**
Fix the above code to properly initialize the Rectangle class

**Code**



```python
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.

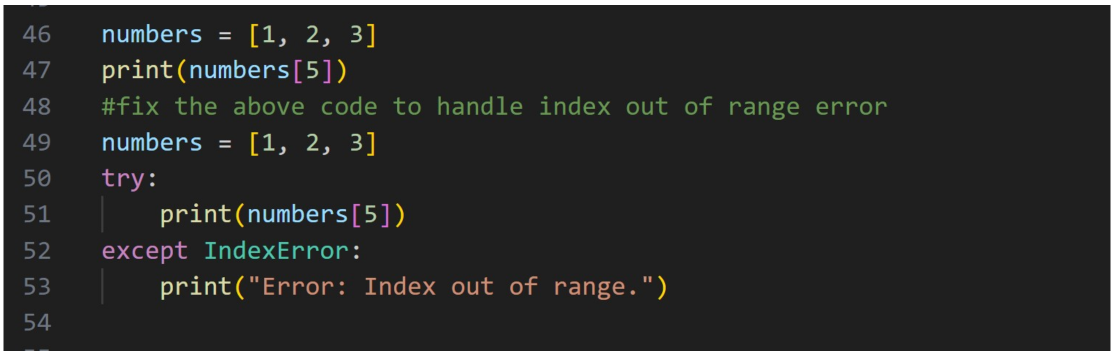
## Task 5: Resolving Index Errors in Lists

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

**Prompt**

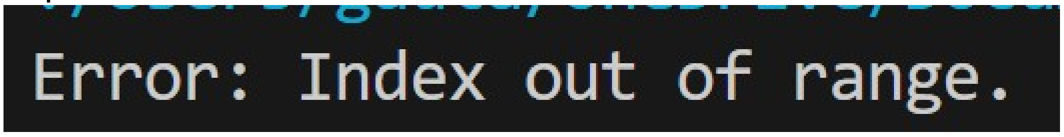
Fix the above code to handle index out of range error

**Code**



```python
numbers = [1, 2, 3]
print(numbers[5])
#fix the above code to handle index out of range error
numbers = [1, 2, 3]
try:
    print(numbers[5])
except IndexError:
    print("Error: Index out of range.")
```

**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.