

## ASSIGNMENT – 7.3

V.Vishnu

2303A52035

Batch – 38

### Task 1: Fixing Syntax Errors

#### Scenario

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

```
1  def add(a, b)
2  |    return a + b
```

#### Requirements

- Provide a Python function add(a, b) with a missing colon
- Use an AI tool to detect the syntax error
- Allow AI to correct the function definition
- Observe how AI explains the syntax issue

#### Expected Output

- Corrected function with proper syntax
- Syntax error resolved successfully
- AI-generated explanation of the fix

#### OUTPUT:

```
PS C:\Users\vishn> & C:/Users/vishr
File "c:/Users/vishn\Design a sim
def add(a, b)
^
SyntaxError: expected ':'
```

You are correct. In Python, function definitions require a colon `:` at the end of the function signature. Here's the corrected code:

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

#### Explanation:

- The colon `:` tells Python that the following indented lines are part of the function body.
- Without the colon, you'll get a `SyntaxError`.

#### Tip:

Always check for colons after function, class, and control statement headers (`if`, `for`, `while`, etc.).

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

## Task 2: Debugging Logic Errors in Loops

### Scenario

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

C: > Users > vishn > ✎ Design a simple ca

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

### Requirements

- Provide a loop with an increment or decrement error
- Use AI to identify the cause of infinite iteration
- Let AI fix the loop logic
- Analyze the corrected loop behavior

### Expected Output

- Infinite loop issue resolved
- Correct increment/decrement logic applied
- AI explanation of the logic error

### OUTPUT:

```
1 def count_down(n):  
2     while n >= 0:  
3         print(n)  
4         n += 1
```

Comment 1 of 1

Code Review

The statement `n += 1` causes an infinite loop or incorrect behavior for a countdown. It should be `n -= 1` to decrement the counter.

No change found to suggest.

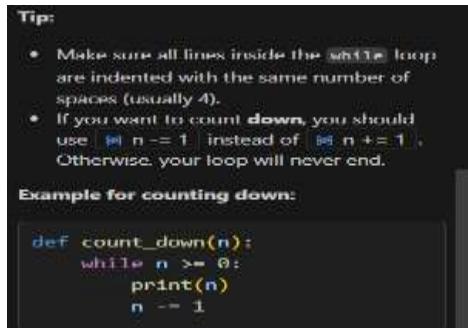
```
1 def count_down(n):  
2     while n >= 0:  
3         print(n)  
4         n -= 1
```

Comment 1 of 1

Code Review

The statement `n += 1` increments `n`, causing an infinite loop when `n` is non-negative. For a countdown, you should decrement `n` with `n -= 1`.

No change found to suggest.



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

### Scenario

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

```
C: > Users > vishn > Design a simple calculator.py > ...
1  def divide(a, b):
2      if b == 0:
3          return "Error: Cannot divide by zero"
4      return a / b
5
6  print(divide(10, 0))
```

### Requirements

- Provide a function that performs division without validation
- Use AI to identify the runtime error
- Let AI add try-except blocks for safe execution
- Review AI's error-handling approach

### Expected Output

- Function executes safely without crashing
- Division by zero handled using try-except
- Clear AI-generated explanation of runtime error handling

```
1  def divide(a, b):
2      return a / b
3  print(divide(10, 0))

Comment 1 of 2
Code Review
No error handling for division by zero. Calling divide(10, 0) will raise a ZeroDivisionError. Consider adding a check for b > 0 and handling it appropriately (e.g., raising a custom exception or returning a specific value).
No change found to suggest.

Apply and Go to Next ⌘D Discard and Go to Next ⌘Z
```

```
3  print(divide(10, 0))
```

A screenshot of a code editor showing a Python script. The code defines a function `divide(a, b)` that returns `a / b`. A call to `divide(10, 0)` is made, which will result in a `ZeroDivisionError`. The code editor has a status bar at the bottom.

```
1 def divide(a, b):
2     return a / b
3 print(divide(10, 0))
```

### Explanation:

The runtime error occurs when dividing by zero.

Using a try-except block catches the `ZeroDivisionError` and prevents the program from crashing.

## Task 4: Debugging Class Definition Errors

### Scenario

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

### Requirements

A screenshot of a code editor showing a Python class definition for `Rectangle`. The class has a constructor `__init__` that takes `length` and `width` parameters but does not use the `self` parameter to initialize instance variables.

```
1 class Rectangle:
2     def __init__(length, width):
3         self.length = length
4         self.width = width
```

- Provide a class definition with missing `self`-parameter
- Use AI to identify the issue in the `__init__()` method
- Allow AI to correct the class definition
- Understand why `self` is required

### Expected Output

- Corrected `__init__()` method
- Proper use of `self` in class definition
- AI explanation of object-oriented error

A screenshot of a code editor showing the corrected Python class definition for `Rectangle`. The constructor now uses the `self` parameter to initialize the `length` and `width` instance variables.

```
1 class Rectangle:
2     def __init__(self, length, width):
3         self.length = length
4         self.width = width
```

### EXPLANATION:

The constructor `__init__` was missing the `self` parameter.

`self` refers to the current object and is required to store and access instance variables inside a class.

## Task 5: Resolving Index Errors in Lists

### Scenario

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

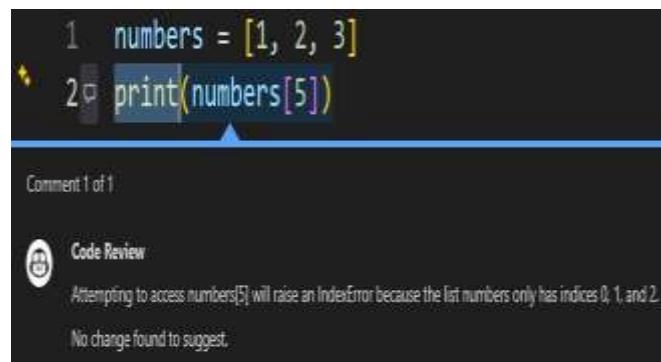
### Requirements

```
C: > Users > vishn > Design a sir  
1   Numbers = [1, 2, 3]  
2   print(Numbers[5])
```

- Provide code that accesses an out-of-range list index
- Use AI to identify the Index Error
- Let AI suggest safe access methods
- Apply bounds checking or exception handling

### Expected Output

- Index error resolved
- Safe list access logic implemented
- AI suggestion using length checks or exception handling



```
1 # Resolve the index error in list from the below code
2 numbers = [1, 2, 3]
3 try:
4     print(numbers[5])
5 except IndexError:
6     print("Index is out of bounds")
```

### Explanation:

The error occurs because index 5 does not exist in the list.

Using try-except catches the Index Error and prevents the program from crashing.

