

Name: K.Likitha

Hall Ticket: 2303A52144

Batch: 41

Task 1: Fixing Syntax Errors

```
python

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

Prompt:

def add(a,b) return a+b

detect the syntax error in the above code, correct the function definition and explain the syntax issue and give correct output

Code:

```
def add(a, b):
    return a + b
print(add(2, 3))
```

Output:

```
pts/python.exe c:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/Untitled-2.py
5
o (.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding>
```

Explanation:

Python requires a **colon (:)** after the function parameter list.

The function body must be on the next line and **indented** (typically 4 spaces).

Task 2: Debugging Logic Errors in Loops

```
python

def count_down(n):
    while n >= 0:
        print(n)
        n += 1 # Should be n -= 1
```

Prompt:

fix the error in the code by Providing a loop with an increment or decrement error, identify the cause of infinite iteration and fix the loop logic

- Analyze the corrected loop behavior and explain the logic error

Code:

```
Untitled-2.2.py > count_down
1 def count_down(n):
2     while n>=0:
3         print(n)
4         n+=1
5         n-=1
```

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

# Call the function to see the output
count_down(5)
```

Output:

```
5
4
3
2
1
0
```

Explanation:

Problem:

- The loop condition `while n>=0` expects `n` to decrease toward 0.
- The body uses `n+=1`, which increases `n`.
- Since `n` only increases, it never becomes negative, so the loop never exits.

Task 3: Handling Runtime Errors (Division by Zero)

```
# Debug the following code
def divide(a, b):
    return a / b

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

Prompt:

Debug the following code.add try-except blocks to handle division by zero error.Provide a function that performs division without validation.give error handlin approach and identify the runtime error

Code:

```
# Function WITHOUT validation (original - will raise ZeroDivisionError)
def divide_no_validation(a, b):
    """Performs division without any error checking."""
    return a / b

# Function WITH error handling using try-except
def divide(a, b):
    """
    Performs division with error handling for division by zero.
    Returns None if division by zero occurs.
    """
    try:
        result = a / b
        return result
    except ZeroDivisionError:
        print(f"Error: Cannot divide {a} by zero!")
        return None

# Function WITH validation (proactive approach)
def divide_with_validation(a, b):
    """
    Performs division with validation before attempting division.
    Checks if divisor is zero before dividing.
    """
    if b == 0:
        print(f"Error: Cannot divide {a} by zero!")
        return None
    return a / b

# Test the functions
print("== Testing divide() with try-except ==")
result1 = divide(10, 2)
print(f"divide(10, 2) = {result1}")

result2 = divide(10, 0)
print(f"divide(10, 0) = {result2}")

print("\n== Testing divide_with_validation() ==")
result3 = divide_with_validation(10, 2)
print(f"divide_with_validation(10, 2) = {result3}")

result4 = divide_with_validation(10, 0)
print(f"divide_with_validation(10, 0) = {result4}")

print("\n== Testing divide_no_validation() (will raise error) ==")
```

```
try:  
    result5 = divide_no_validation(10, 2)  
    print(f"divide_no_validation(10, 2) = {result5}")  
  
    # This will raise ZeroDivisionError  
    result6 = divide_no_validation(10, 0)  
    print(f"divide_no_validation(10, 0) = {result6}")  
except ZeroDivisionError as e:  
    print(f"Runtime Error Caught: {type(e).__name__}: {e}")
```

Output:

```
== Testing divide() with try-except ==  
divide(10, 2) = 5.0  
Error: Cannot divide 10 by zero!  
divide(10, 0) = None  
  
== Testing divide_with_validation() ==  
divide_with_validation(10, 2) = 5.0  
Error: Cannot divide 10 by zero!  
divide_with_validation(10, 0) = None  
  
== Testing divide_no_validation() (will raise error) ==  
divide_no_validation(10, 2) = 5.0  
Runtime Error Caught: ZeroDivisionError: division by zero  
o (.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> □
```

Explanation:

Runtime error identified

Error type: ZeroDivisionError

Cause: Dividing by zero (e.g., 10 / 0)

The runtime error occurs because dividing any number by zero is not allowed in Python. This causes a ZeroDivisionError when divide(10, 0) is executed. The try block runs the division code that may cause an error. The except ZeroDivisionError block catches the error instead of crashing the program. This approach handles the error gracefully by displaying a clear error message.

Task 4: Debugging Class Definition Errors

```
python
```

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

Prompt:

Provide a class definition with missing self-parameter. identify the issue in the `__init__()` method. correct the class definition. Understand why `self` is required and explain it

Code:

```
Untitled-2.2.py > ... ^ 2 of 2 | Undo File Keep File Ctrl+D  
class rectangle:  
    def __init__(length, width):  
1 class Rectangle:  
2     def __init__(self, length, width):  
3         self.length = length  
4         self.width = width  
5  
6  
7 # Example usage (so you can see output when you run the file)  
8 rect = Rectangle(10, 5)  
9 print(rect.length, rect.width) ^ 2 of 2 | Undo Ctrl+N Keep Ctrl+Shift+Y
```

```
class Rectangle:  
    def __init__(self, length, width):  
        self.length = length  
        self.width = width  
  
# Example usage (so you can see output when you run the file)  
rect = Rectangle(10, 5)  
print(rect.length, rect.width)
```

Output:

```
OneDrive\Desktop\AI-Assisted-Coding\Untitled-2.2.py  
10 5  
o (.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> []  
Ctrl+K to generate command
```

Explanation:

What was wrong in your `__init__()` (missing `self`)

Your code:

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

- **Issue 1 (missing `self`):** Instance methods must take the instance as the **first parameter** (conventionally named `self`).
- **What happens at runtime:** When you call `rectangle(10, 5)`, Python *implicitly* passes the new object as the first argument. So your parameters behave like:
 - `length` actually receives the instance (the object)
 - `width` receives `10`
 - and then `self` is **not defined** inside the method → error.

The issue was that the `__init__()` method was missing the `self` parameter. Without `self`, the class cannot store values in instance variables. Adding `self` allows `length` and `width` to be saved as object attributes. The corrected class properly initializes each Rectangle object. `self` is required because it refers to the current object and lets methods access its data.

Task 5: Resolving Index Errors in Lists

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

Prompt:

Provide code that accesses an out-of-range list index. Identify the Index Error. Suggest safe access methods. Apply bounds checking or exception handling. Explain the changes made.

Code:

```
1 numbers = [1, 2, 3]  
2 print(numbers[5])  
3  
4 # Out-of-range access (this causes IndexError: list index out of range)  
5 # print(numbers[index])  
6  
7 # Safe access method 1: bounds checking  
8 if 0 <= index < len(numbers):  
9     print(numbers[index])  
10 else:  
11     print(f"Index {index} is out of range for list of length {len(numbers)}")  
12  
13 # Safe access method 2: exception handling  
14 try:  
15     print(numbers[index])  
16 except IndexError:  
17     print(f"Caught IndexError: index {index} is out of range")
```

Ctrl+N Ctrl+Shift+Y

Output:

```
OneDrive/Desktop/AI-Assisted-Coding/Untitled-2.2.py
Index 5 is out of range for list of length 3
Caught IndexError: index 5 is out of range
○ (.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> 
```

Explanation:

Out-of-range access + the `IndexError`

Your original line `numbers[5]` raises `IndexError: list index out of range` because `numbers = [1, 2, 3]` only has valid indices `0..2`.

What changed and why

- **Bounds checking:** prevents the error by ensuring `index` is within `0` and `len(numbers)-1`.
- **`try/except IndexError`:** allows the access attempt, but **handles the runtime error gracefully** if it happens.

The list only has three items, but the code tries to access a sixth one. Python gets confused because that position doesn't exist and raises an `IndexError`. The `try` block is used to attempt the risky operation. When the error happens, the `except` block steps in instead of crashing the program. This way, the program stays safe and shows a helpful message instead of stopping.