

K.Akshay  
2303A51817  
B-26

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju	
Course Code	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week4 – Wednesday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
AssignmentNumber: <b>7.3</b> (Present assignment number)/ <b>24</b> (Total number of assignments)			
Q.No.	Question		Expected Time to complete
1			Week4 -

	**Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs**  **Lab Objectives**   - To identify and correct syntax, logic, and runtime errors in Python programs using AI tools - To understand common programming bugs and AI-assisted debugging suggestions - To evaluate how AI explains, detects, and fixes different types of coding errors - To build confidence in using AI for structured debugging practices   **Lab Outcomes (LOs)**  After completing this lab, students will be able to:   - Use AI tools to detect and correct syntax, logic, and runtime errors - Interpret AI-suggested bug fixes and explanations - Apply systematic debugging strategies using AI-generated insights - Refactor buggy code using reliable programming patterns   **Task 1: Fixing Syntax Errors**  **Scenario** You are reviewing a Python program where a basic function definition contains a syntax error.   ``` python  def add(a, b)     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   **PROMPT:** Here is a Python function with a syntax error. Please find the error, fix it, and explain what was wrong. `def add(a, b) return a + b`  **EXPLANATION:** The error was caused by a missing colon (`:`) at the end of the function definition. In Python, a colon is required to start the function body. Adding it fixes the syntax and allows the code to run properly.  The screenshot shows a Jupyter Notebook cell with the following code:   ``` def add(a, b):     return a + b ```   An AI-generated note above the code states: "The original function definition `def add(a, b)` was missing a colon (`:`) at the end. In Python, a colon is crucial to mark the beginning of the function body. Without it, the interpreter doesn't know where the function's code starts, leading to a `SyntaxError`. The corrected code is:   ``` def add(a, b):     return a + b ```   The output cell shows the result of running the corrected code:   ``` The sum is: 8 ```

## Task 2: Debugging Logic Errors in Loops

### Scenario

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

```
python

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

**PROMPT:** Here is a loop that runs infinitely. Please identify why it does not stop, fix the logic error, and explain the correction.

def count down(n):

```
    while n >= 0:
        print(n)
        n = 1
```

**EXPLANATION:** The loop was infinite because the counter variable was never updated correctly inside the loop.

By adding the proper increment/decrement, the condition eventually becomes false.

This allows the loop to stop after the expected number of iterations.

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text | Run all

The count_down function you provided does not run infinitely. It correctly counts down and terminates. The 'logic error' was in the assumption that this loop would run infinitely.

Here's why it stops:
• The while n >= 0: loop condition checks if n is greater than or equal to zero.
• Inside the loop, n -= 1 decrements the value of n by 1 in each iteration.
• This ensures that n will eventually become a negative number (e.g., if n starts at 3, it will become 2, 1, 0, and then -1).
• When n becomes negative, the condition n >= 0 becomes False, and the loop naturally exits, preventing an infinite loop.

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

print("Counting down from 5:")
count_down(5)
print("Loop finished.\n")

print("Counting down from 0:")
count_down(0)
print("Loop finished.\n")

print("Trying with a negative number (-2):")
count_down(-2)
print("Loop finished.")

*** Counting down from 5:
5
4
3
2
1
0
Loop finished.

Counting down from 0:
0
Loop finished.

Trying with a negative number (-2):
Loop finished.
```

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

### Scenario

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

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

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

**PROMPT:** Here is a Python function that performs division but crashes at runtime. Please identify the error, fix it using try-except, and explain how the error is handled."

**EXPLANATION:** The program crashes when division by zero occurs, which raises a runtime error.

Using a try-except block prevents the program from stopping suddenly. Instead, it safely catches the error and shows a user-friendly message.

The screenshot shows a Jupyter Notebook cell with the following content:

```
# Debug the following code
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
        return None # Or any other appropriate value/handler

print("Result of 10 / 2:", divide(10, 2))
print("Result of 10 / 0:", divide(10, 0))
print("Result of 5 / 2.5:", divide(5, 2.5))
```

Below the code cell, the output is displayed:

```
Result of 10 / 2: 5.0
Error: Cannot divide by zero!
Result of 10 / 0: None
Result of 5 / 2.5: 2.0
```

### Task 4: Debugging Class Definition Errors

#### Scenario

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

```
python

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

#### Requirements

- 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

**PROMPT:** Here is a Python class with an incorrect constructor. Please find the error in the `__init__` method, fix it, and explain why `self` is required.

**EXPLANATION:** The constructor was missing `self` as the first parameter.

`self` is required to store values inside the object.

After adding it, the class works correctly.

The `__init__` method in your `Rectangle` class is correctly defined. The parameter `self` is crucial in Python classes for the following reasons:

- **Instance Reference:** `self` is a conventional name (though not a Python keyword) for the first parameter of all instance methods. It represents the instance of the class itself. When you create an object (e.g., `my_rectangle = Rectangle(10, 5)`), `self` inside the `__init__` method refers to this newly created `my_rectangle` object.
- **Accessing Instance Attributes:** It allows you to access and set attributes that belong specifically to *that particular instance* of the class. For example, `self.length = length` means that the `length` value passed to the constructor is stored as an attribute `length` of the `self` object.
- **Distinction from Local Variables:** Without `self`, `length = length` within `__init__` would assign the `length` parameter to a local variable named `length`, not to an attribute of the instance, meaning the object wouldn't retain these values.

In essence, `self` ensures that the method operates on the correct object instance and allows each object to have its own unique set of data.

```

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

    def get_area(self):
        return self.length * self.width

# Demonstrating the correct class and its usage:
my_rectangle = Rectangle(10, 5)
print("The rectangle has a length of " + str(my_rectangle.length) + " and a width of " + str(my_rectangle.width))
print("The area of the rectangle is: " + str(my_rectangle.get_area()))

other_rectangle = Rectangle(7, 3)
print("The other rectangle has a length of " + str(other_rectangle.length) + " and a width of " + str(other_rectangle.width))
print("The area of the other rectangle is: " + str(other_rectangle.get_area()))

```

... The rectangle has a length of 10 and a width of 5.  
The area of the rectangle is: 50  
The other rectangle has a length of 7 and a width of 3.  
The area of the other rectangle is: 21

#### Task 5: Resolving Index Errors in Lists

##### Scenario

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

python

```

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

```

##### Requirements

- 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.

**PROMPT:** Here is a Python program that crashes due to an invalid list index. Please identify the IndexError, fix it using safe list access (length check or try-except), and explain the solution.

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

**EXPLANATION:** The error occurs because the program tries to access an index that does not exist in the list.

By checking the list length or using try-except, we prevent the crash.  
This makes the program run safely even when the index is invalid.

The screenshot shows a Jupyter Notebook interface with a single cell containing Python code. The code demonstrates two methods for handling list index errors: try-except and length checking. It includes explanatory comments and output from running the code.

```
numbers = [1, 2, 3]

# --- Using try-except for safe access ---
index_to_access = 5
try:
    print("Attempting to access index {index_to_access}: {numbers[index_to_access]}")
except IndexError:
    print("Error: Index {index_to_access} is out of bounds for the list.")

index_to_access = 1
try:
    print("Attempting to access index {index_to_access}: {numbers[index_to_access]}")
except IndexError:
    print("Error: Index {index_to_access} is out of bounds for the list.")

# --- Alternative: Using a length check ---
print("\n--- Using length check for safe access ---")

def safe_list_access(my_list, index):
    if 0 <= index < len(my_list):
        return my_list[index]
    else:
        print("Error: Index {index} is out of bounds for the list.")
        return None

print("Accessing index 5: {safe_list_access(numbers, 5)}")
print("Accessing index 0: {safe_list_access(numbers, 0)}")
```

Output from the cell:

```
--> Error: Index 5 is out of bounds for the list.
Attempting to access index 1: 2
--- Using length check for safe access ---
Error: Index 5 is out of bounds for the list.
Accessing index 5: None
Accessing index 0: 1
```

**Note:** Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots