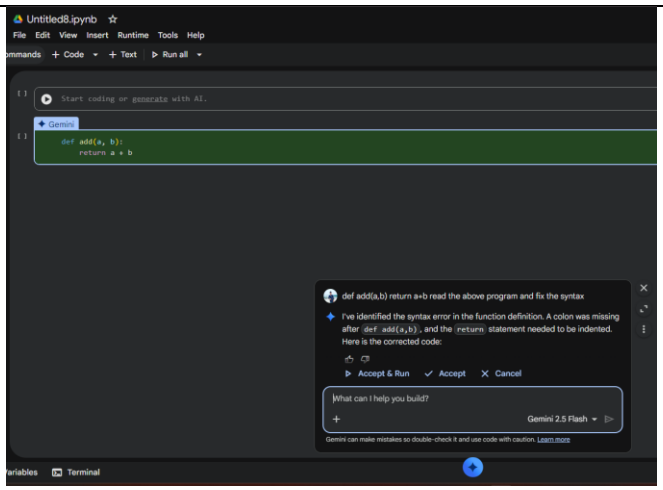
CH.VASU
2303A51618
BATCH 25

SCHOOL OF COMPUTER SC IENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
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:7.3(Present assignment number)/**24**(Total number of assignments)			

	Question	Expected Time to complete
	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	

1	**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
		Week4 - Wednesday



Prompt :

def add(a,b) return a+b read the above program and fix the syntax

I've identified the syntax error in the function definition. A colon was missing after `def add(a,b)`, and the `return` statement needed to be indented. Here is the corrected code:

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
def count_down(n): while(n>=0): print(n) n+=1 apply the correct loop increment logic and give Ai logic detection

I've identified and fixed the syntax and logical issues in your `count_down` function. The original code had a missing colon after the `while` condition and incorrect indentation. More critically, for a countdown function, the variable n needs to be decremented (`n -= 1`) within the loop, not incremented (`n += 1`), otherwise, it would result in an infinite loop if n starts as a non-negative number.

Task 3: Handling Runtime Errors (Division by Zero)

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

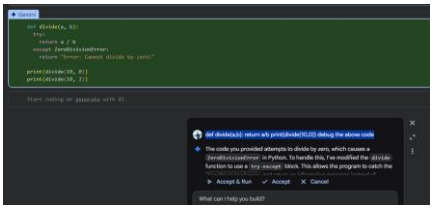
```
# Debug the following code
def divide(a, b):
    return a / b

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: def divide(a,b): return a/b print(divide(10,0)) debug the above code



The code you provided attempts to divide by zero, which causes a `ZeroDivisionError` in Python. To handle this, I've modified the `divide` function to use a `try-except` block. This allows the program to catch the `ZeroDivisionError` and return an informative message instead of crashing.
The code in cell 31be9d13 has executed successfully. As intended, the `divide` function correctly handled the division by zero, printing "Error: Cannot divide by zero!", and then successfully calculated 10 divided by 2,

which resulted in `5.0`.

Task 4: Debugging Class Definition Errors

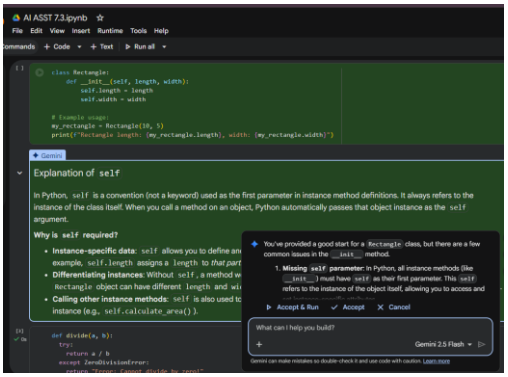
Scenario
You are given a faulty Python class where the constructor is incorrectly defined.

```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

python class Rectangle: def _init_(length, width): self.length = length self.width = width Requirements · Provide a class definition with missing self-parameter



In Python, `self` is a convention (not a keyword) used as the first parameter in instance method definitions. It always refers to the instance of the class itself. When you call a method on an object, Python automatically passes that object instance as the `self` argument.

Why is `self` required?

- **Instance-specific data**: `self` allows you to define and access attributes (data) that belong to a specific instance of the class. For example, `self.length` assigns a `length` to *that particular* `Rectangle` object.
- **Differentiating instances**: Without `self`, a method wouldn't know which object's attributes it should be operating on. Each `Rectangle` object can have different `length` and `width` values, and `self` ensures you're working with the correct object's data.
- **Calling other instance methods**: `self` is also used to call other methods defined within the same class on the current object instance (e.g., `self.calculate_area()`).

Task 5: Resolving Index Errors in Lists

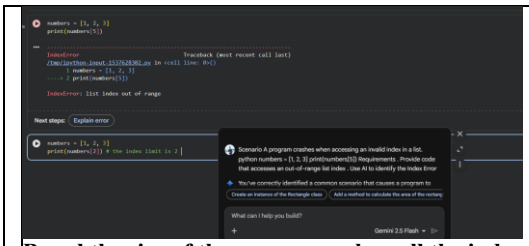
Scenario
A program crashes when accessing an invalid index in a list.

```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

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



Based the size of the array we cabn call the index to print

We must call it on its indexs ranges .