SCHOOL OF COMPUTER SCIENCE 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		
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week1 – Wednesday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number:1.3(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*

1	Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI **Lab Objectives:** ❖ To explore and evaluate the functionality of Google Gemini for AI-assisted coding within Google Colab. ❖ To understand and use Cursor AI for code generation, explanation, and refactoring. ❖ To compare outputs and usability between Gemini, GitHub Copilot, and Cursor AI. ❖ To perform code optimization and documentation using AI tools. **Lab Outcomes (LOs):** After completing this lab, students will be able to: ❖ Generate Python code using Google Gemini in Google Colab. ❖ Analyze the effectiveness of code explanations and suggestions by Gemini. ❖ Set up and use Cursor AI for AI-powered coding assistance. ❖ Evaluate and refactor code using Cursor AI features. ❖ Compare AI tool behavior and code quality across different platforms. **Task 1: Refactoring Odd/Even Logic (List Version)** ❖ **Scenario:** You are improving legacy code. ❖ **Task:** Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI. ❖ **Expected Output:** ❖ Original and improved code **Task 2: Area Calculation Explanation** ❖ **Scenario:** You are onboarding a junior developer. ❖ **Task:** Ask Gemini to explain a function that calculates the area of different shapes. ❖ **Expected Output:** ➢ Code ➢ Explanation **Task 3: Prompt Sensitivity Experiment** ❖ **Scenario:** You are testing how AI responds to different prompts. ❖ **Task:** Use Cursor AI with different prompts for the same problem and observe code changes. ❖ **Expected Output:** ➢ **Prompt list**	Week1 - Monday

Lab 2:

Task 1: Refactoring Odd/Even Logic (List Version):

Given Prompt:

Write a Refactoring Odd/Even Logic (List Version) according to the given sceniro You are improving legacy code the task is to Write a program to calculate the sum of odd and even numbers in a list, then refactor it using AI. The expected output should be Original and improved code

Orginal Version of code:

```
# Original Code - Verbose and repetitive
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

odd_sum = 0
even_sum = 0

for num in numbers:
    if num % 2 == 0:
        even_sum += num
    else:
        odd_sum += num

print(f"Odd Sum: {odd_sum}")
print(f"Even Sum: {even_sum}")
```

Sample Input given in the code:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Sample output:

```
Odd Sum: 25
Even Sum: 30
```

Refactored Code - Clean and Python:

```
# Refactored Code - Clean and Python
def calculate_sums(numbers):
    return {
        'odd': sum(n for n in numbers if n % 2),
        'even': sum(n for n in numbers if n % 2 == 0)
    }
#sample input:
result = calculate_sums([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

print(f"Odd Sum: {result['odd']}")
print(f"Even Sum: {result['even']}")
```

Sample output:

```
Odd Sum: 25
Even Sum: 30
```

Task 2: Area Calculation Explanation:

Given Prompt:

Write a Python function called calculate_area that serves as a single entry point for finding the area of a circle, square, rectangle, triangle, and trapezoid. Use a dictionary of lambda functions to handle the logic for each shape type and **kwargs to accept varying inputs, ensuring the entire solution is under 15 lines of code.

Code:

```python
import math

def calculate_area(shape, **kwargs):
    formulas = {
        'circle': lambda r: math.pi * r**2,
        'square': lambda s: s**2,
        'rectangle': lambda l, w: l * w,
        'triangle': lambda b, h: 0.5 * b * h,
        'trapezoid': lambda a, b, h: 0.5 * (a + b) * h
    }
    if shape not in formulas: return "Shape not supported"
    return formulas[shape](*kwargs.values())

# Sample Input:
print(f"Circle: {calculate_area('circle', r=5):.2f}")
print(f"Rectangle: {calculate_area('rectangle', l=10, w=5)}")
```

Sample input:

```
calculate area('rectangle', l=10, w=5)

calculate_area('circle', r=5)
```

Sample output:

```
Circle: 78.54
Rectangle: 50
```

Explanation:

This code uses a single function to calculate the area of various shapes by looking them up in a predefined list. Instead of using many "if" statements, it stores the math formulas in a dictionary and picks the right one based on the name you provide. It uses a special Python feature called **kwargs to collect whatever numbers you provide, like radius or height, and plugs them into the formula. This makes the code very short, organized, and easy to update with new shapes.

Task 3: Prompt Sensitivity Experiment

Given Prompt variation:

1. Write a concise Python function to check if a string and number is a palindrome.
   only one code

code 1:

```
def is_palindrome(value):
    s = str(value).replace(" ", "").lower()
    return s == s[::-1]

# Example usage
user_input = input("Enter a word, phrase, or number: ")
if is_palindrome(user_input):
    print(f"'{user_input}' is a palindrome!")
else:
    print(f"'{user_input}' is not a palindrome.")
```

sample input:

```
Enter a word, phrase, or number: dad
```

Sample output:

```
'dad' is a palindrome!
```

2. **give python code on pallindrome only one code**

Code 2:

```
def is_palindrome(s):
    # Remove spaces and convert to lowercase
    s = s.replace(" ", "").lower()
    return s == s[::-1]

# Example usage
word = input("Enter a word or phrase: ")
if is_palindrome(word):
    print(f"'{word}' is a palindrome!")
else:
    print(f"'{word}' is not a palindrome.")
```

Simple input:

```
Enter a word or phrase: dad
```

Sample output:

```
'dad' is a palindrome!
```

3. Given a string as input, write a program to determine whether the string is a palindrome or not.

Code 3:

```python
def is_palindrome(s: str) -> bool:
    s = "".join(char.lower() for char in s if char.isalnum())

    left, right = 0, len(s) - 1

    while left < right:
        if s[left] != s[right]:
            return False
        left += 1
        right -= 1

    return True

# Testing the logic
if __name__ == "__main__":
    user_input = input("Enter a string: ")
    if is_palindrome(user_input):
        print(f"'{user_input}' is a palindrome.")
    else:
        print(f"'{user_input}' is not a palindrome.")
```

Sample Input:

```
Enter a string: ksb
```

Sample output:

```
ksb' is not a palindrome.
```

Task 4: Tool Comparison Reflection:

Gemini is a great choice for students, hobbyists, or anyone on a tight budget. Its huge context window and Agent Mode make it ideal for exploring large codebases without paying anything.

GitHub Copilot fits best for professional teams that want a reliable, no-friction tool. It works smoothly across IDEs and connects well with GitHub's security, pull requests, and enterprise workflows.

Cursor AI is perfect for power users and startups who care about speed. If you frequently work with large projects or need fast multi-file changes, Cursor delivers productivity that regular

code plugins still can’t match.