|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE** | | | | | **DEPARTMENT OF COMPUTER SCIENCE ENGINEERING** | | | | |
| **ProgramName:**B. Tech | | | | **Assignment Type: Lab** | | | **AcademicYear:**2025-2026 | | |
| **CourseCoordinatorName** | | | | Venkataramana Veeramsetty | | | | | |
| **Instructor(s)Name** | | | | |  | | --- | | Dr. V. Venkataramana (Co-ordinator) | | Dr. T. Sampath Kumar | | Dr. Pramoda Patro | | Dr. Brij Kishor Tiwari | | Dr.J.Ravichander | | Dr. Mohammand Ali Shaik | | Dr. Anirodh Kumar | | Mr. S.Naresh Kumar | | Dr. RAJESH VELPULA | | Mr. Kundhan Kumar | | Ms. Ch.Rajitha | | Mr. M Prakash | | Mr. B.Raju | | Intern 1 (Dharma teja) | | Intern 2 (Sai Prasad) | | Intern 3 (Sowmya) | | NS\_2 ( Mounika) | | | | | | |
| **CourseCode** | | | 24CS002PC215 | **CourseTitle** | | AI Assisted Coding | | | |
| **Year/Sem** | | | II/I | **Regulation** | | R24 | | | |
| **Date and Day**  **of Assignment** | | | Week1 - Wednesday | **Time(s)** | |  | | | |
| **Duration** | | | 2 Hours | **Applicableto**  **Batches** | | 24CSBTB01 To 24CSBTB39 | | | |
| **AssignmentNumber:2.3**(Present assignment number)/**24**(Total number of assignments) | | | | | | | | | |
|  | | | | | | | | | |
|  | **Q.No.** | **Question** | | | | | | ***ExpectedTime***  ***to complete*** |  |
|  | 1 | Lab 2: Exploring Additional AI Coding Tools – 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 Description#1**   * Use Google Gemini in Colab to write a function that reads a CSV file and calculates mean, min, max.   **Expected Output#1**  CODE:  import pandas as pd  def analyze\_csv\_data(file\_path):    """    Reads a CSV file and calculates mean, min, and max of numerical columns.    Args:      file\_path: The path to the CSV file.    Returns:      A dictionary containing the mean, min, and max for each numerical column.    """    try:      df = pd.read\_csv(file\_path)      numerical\_cols = df.select\_dtypes(include=['number'])      if numerical\_cols.empty:        print("No numerical columns found in the CSV.")        return None      analysis\_results = {}      for col in numerical\_cols.columns:        analysis\_results[col] = {            'mean': numerical\_cols[col].mean(),            'min': numerical\_cols[col].min(),            'max': numerical\_cols[col].max()        }      return analysis\_results    except FileNotFoundError:      print(f"Error: File not found at {file\_path}")      return None    except Exception as e:      print(f"An error occurred: {e}")      return None  # Example usage with your file:  file\_path = '/content/Book3.csv'  analysis = analyze\_csv\_data(file\_path)  if analysis:    for col, stats in analysis.items():      print(f"Analysis for column '{col}':")      print(f"  Mean: {stats['mean']:.2f}")      print(f"  Min: {stats['min']}")      print(f"  Max: {stats['max']}")  **OUTPUT :**  **Analysis for column 'min':**  **Mean: 10.50**  **Min: 1**  **Max: 20**  **Analysis for column 'max':**  **Mean: 119.75**  **Min: 3**  **Max: 767**  **CODE SCREENSHOT:**    **OUTPUT SCREENSHOT:**    **Task Description#2**   * Compare Gemini and Copilot outputs for a palindrome check function.   **Expected Output#2**    **The code form google colab:**  def is\_palindrome(s):    """Checks if a string is a palindrome."""    # Remove spaces and convert to lowercase for case-insensitive check    s = s.replace(" ", "").lower()    return s == s[::-1]  # Example usage  print(is\_palindrome("racecar"))  print(is\_palindrome("hello"))  print(is\_palindrome("A man a plan a canal Panama"))  **Observation:**   * This function removes only spaces (" ") and converts the string to lowercase before checking for the palindrome property. * It will consider punctuation and other characters when determining if a string is a palindrome. For example, is\_palindrome("racecar!") would return False**.**     **The code from copilot:**  def is\_palindrome(s):      s = ''.join(c.lower() for c in s if c.isalnum())      return s == s[::-1]  # Example usage:  text = "A man, a plan, a canal: Panama"  print(is\_palindrome(text))  # Output: True  **Observaton:**   * This function is more robust as it removes all non-alphanumeric characters (isalnum()) and converts the string to lowercase. * It is designed to handle cases with punctuation, spaces, and other symbols, considering only the letters and numbers for the palindrome check. For example, is\_palindrome("A man, a plan, a canal: Panama") would correctly return True.   **Task Description#3**   * Ask Gemini to explain a Python function (to calculate area of various shapes) line by line..   **Expected Output#3**  **Code:**  import math  def calculate\_area(shape, \*\*kwargs):    """    Calculates the area of various shapes.    Args:      shape: A string representing the shape (e.g., 'circle', 'square', 'triangle').      \*\*kwargs: Keyword arguments for the shape's dimensions (e.g., radius, side, base, height).    Returns:      The area of the shape, or None if the shape is not supported or dimensions are missing.    """  **if shape == 'circle':**      radius = kwargs.get('radius')      if radius is not None:        return math.pi \* radius\*\*2  **else:**        return None    elif shape == 'square':      side = kwargs.get('side')      if side is not None:        return side\*\*2      else:        return None    elif shape == 'triangle':      base = kwargs.get('base')      height = kwargs.get('height')      if base is not None and height is not None:        return 0.5 \* base \* height      else:        return None    else:      return None  # Example usage  print(f"Area of circle with radius 5: {calculate\_area('circle', radius=5)}")  print(f"Area of square with side 4: {calculate\_area('square', side=4)}")  print(f"Area of triangle with base 6 and height 3: {calculate\_area('triangle', base=6, height=3)}")  print(f"Area of rectangle with length 5 and width 4: {calculate\_area('rectangle', length=5, width=4)}") # Unsupported shape  print(f"Area of circle without radius: {calculate\_area('circle')}") # Missing dimension  **Explanation:**  Here is a line-by-line explanation of the calculate\_area function:  import math  **This line imports the math module,** which provides access to mathematical functions and constants, such as math.pi for the value of Pi, needed for calculating the area of a circle.  **def calculate\_area(shape, \*\*kwargs):**  This line defines a function named calculate\_area that takes two arguments:   * **shape:** A required positional argument that expects a string representing the name of the shape (e.g., 'circle', 'square', 'triangle'). * **\*\*kwargs:** This is a special syntax in Python that allows a function to accept an arbitrary number of keyword arguments. These keyword arguments will be collected into a dictionary named kwargs. This is used to pass the dimensions of the shapes (like radius, side, base, height).     """   Calculates the area of various shapes.  **Args:**     shape: A string representing the shape (e.g., 'circle', 'square', 'triangle').     **\*\*kwargs:** Keyword arguments for the shape's dimensions (e.g., radius, side, base, height).    **Returns:**     The area of the shape, or None if the shape is not supported or dimensions are missing.   """  This is a docstring, which explains what the function does, its arguments (Args), and what it returns (Returns). It's good practice to include docstrings for functions.  **if shape == 'circle':**  This line starts a conditional block. It checks if the value of the shape argument is the string 'circle'.      radius = kwargs.get('radius')  If the shape is 'circle', this line attempts to get the value associated with the keyword argument 'radius' from the kwargs dictionary. The .get() method is used here, which is safer than direct dictionary access (kwargs['radius']) because it returns None if the key ('radius') is not found, instead of raising a KeyError.  **if radius is not None:**       return math.pi \* radius\*\*2  This checks if a radius value was successfully retrieved from kwargs. If radius is not None, it calculates the area of a circle using the formula π \* radius² and returns the result. math.pi provides the value of Pi.  **else:**       return None  If radius is None (meaning the 'radius' keyword argument was not provided when calling the function for a circle), the function returns None to indicate that the area could not be calculated.  **elif shape == 'square':**  This is another conditional check. If the shape was not 'circle', it checks if the shape is 'square'.      side = kwargs.get('side')  If the shape is 'square', this line gets the value for the 'side' keyword argument from kwargs.      if side is not None:       return side\*\*2  If a side value was provided, it calculates the area of a square using the formula side² and returns the result.  **else:**       return None  If the 'side' keyword argument was missing for a square, the function returns None.  **elif shape == 'triangle':**  This checks if the shape is 'triangle'.      base = kwargs.get('base')     height = kwargs.get('height')  If the shape is 'triangle', it attempts to get the values for both 'base' and 'height' keyword arguments from kwargs.  **if base is not None and height is not None:**       return 0.5 \* base \* height  This checks if both base and height values were provided. If they are, it calculates the area of a triangle using the formula 0.5 \* base \* height and returns the result.  **else:**       return None  If either 'base' or 'height' (or both) were missing for a triangle, the function returns None.  **else:**     return None  This is the final else block. If the shape argument does not match 'circle', 'square', or 'triangle', the function returns None because it doesn't know how to calculate the area of the provided shape.  # Example usage **print(f"Area of circle with radius 5:** {calculate\_area('circle', radius=5)}") **print(f"Area of square with side 4:** {calculate\_area('square', side=4)}") **print(f"Area of triangle with base 6 and height 3:** {calculate\_area('triangle', base=6, height=3)}") **print(f"Area of rectangle with length 5 and width 4:** {calculate\_area('rectangle', length=5, width=4)}") # Unsupported shape **print(f"Area of circle without radius:** {calculate\_area('circle')}") # Missing dimension  These lines demonstrate how to call the calculate\_area function with different shapes and dimensions and print the results. The f"" syntax is used for f-strings, which allow embedding expressions inside string literals. The last two examples show what happens when an unsupported shape is given or when necessary dimensions are missing.  **Task Description#4**   * Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of squares).   **Expected Output#4**   * **Screenshots of working environments:**   **Python code to calculate the sum of the squares in cursor ai:**    **Task Description#5**   * Student need to write code to calculate sum of add number and even numbers in the list   **Expected Output#5**  **Student written code:**  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  even\_sum = 0  odd\_sum = 0  for number in numbers:       if number % 2 == 0:            even\_sum += number       elif number % 2 != 0:            odd\_sum += number  print("Sum of even numbers:", even\_sum)  print("Sum of odd numbers:", odd\_sum)  **OUTPUT:**      **Refactored code written by student with improved logic :**  # Define the list of numbers  numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  # Initialize variables to store the sums  even\_sum = 0  odd\_sum = 0  # Loop through each number in the list  for number in numbers:    # Check if the number is even    if number % 2 == 0:      # If it's even, add it to the even\_sum      even\_sum += number    else:      # If it's not even (it's odd), add it to the odd\_sum      odd\_sum += number  # Print the results  print(f"Sum of even numbers: {even\_sum}")  print(f"Sum of odd numbers: {odd\_sum}")  **OUTPUT:**    **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**  **Evaluation Criteria:**   | **Criteria** | **Max Marks** | | --- | --- | | Successful Use of Gemini in Colab (Task#1 & #2) | 1.0 | | Code Explanation Accuracy (Gemini) (Task#3) | 0.5 | | Cursor AI Setup and Usage (Task#4) | 0.5 | | Refactoring and Improvement Analysis (Task#5) | 0.5 | | **Total** | **2.5 Marks** | | | | | | | Week1 - Wednesday |  |