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 – Tuesday**	**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)			

	Question	***Expected Time to compl***

		ete
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.	Week1 - Monday
	Task 1: Cleaning Sensor Data ❖ **Scenario**: ❖ You are cleaning IoT sensor data where negative values are invalid. ❖ **Task:** Use Gemini in Colab to generate a function that filters out all negative numbers from a list. ❖ Expected Output: ➢ Before/after list ➢ Screenshot of Colab execution ➢ **OUTPUT** ➢ **Prompt :** Write a Python function to remove negative numbers from a list ❖ **Code**	

```python
def remove_negative_numbers_simple(input_list):
    filtered_list = []
    for num in input_list:
        if num >= 0:
            filtered_list.append(num)
    return filtered_list


my_list = [1, -2, 3, -4, 5, 0, -10]
filtered_list = remove_negative_numbers_simple(my_list)
print(f"Original list: {my_list}")
print(f"List without negative numbers: {filtered_list}")
```

```
Original list: [1, -2, 3, -4, 5, 0, -10]
List without negative numbers: [1, 3, 5, 0]
```

code execution

```
Original list: [1, -2, 3, -4, 5, 0, -10]
List without negative numbers: [1, 3, 5, 0]
```

Explanation

- The function loops through each number in the input list.
- It checks if the number is **0 or positive**.
- Only non-negative numbers are added to a new list.
- Finally, it returns the filtered list without negative values.

Task 2: String Character Analysis

- **Scenario:**
  You are building a text-analysis feature.

- **Task:**
  Use Gemini to generate a Python function that counts vowels, consonants, and digits in a string.

- **Expected Output:**
  - Working function
  - Sample inputs and outputs
  - **OUTPUT**
  - **PROMPT :** generate a Python function that: Counts vowels Counts consonants Counts digits
  - **CODE EXECUTION**

```python
def analyze_string_simple(input_string):
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0
    digit_count = 0

    for char in input_string:
        char_lower = char.lower()
        if char_lower.isalpha():  # Check if it's an alphabet
            if char_lower in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
        elif char_lower.isdigit():  # Check if it's a digit
            digit_count += 1

    return vowel_count, consonant_count, digit_count

# Example usage:
text = "Hello World 123!"
v, c, d = analyze_string_simple(text)
print(f"Original string: '{text}'")
print(f"Vowels: {v}")
print(f"Consonants: {c}")
print(f"Digits: {d}")
```

Code Output

```
Original string: 'Hello World 123!'
Vowels: 3
Consonants: 7
Digits: 3
```

- **Explanation**
- The function goes through each character in the given string.
- If the character is a letter, it checks whether it is a vowel or a consonant.
- If the character is a digit, it increases the digit count.
- Finally, it returns the counts of vowels, consonants, and digits.

Task 3: Palindrome Check – Tool Comparison

- **Scenario**:
  You must decide which AI tool is clearer for string logic.

- **Task:**
  Generate a palindrome-checking function using Gemini and Copilot, then compare the results.

- **Expected Output:**
  - Side-by-side code comparison
  - Observations on clarity and structure
  - **Output**

- **Prompt : generating palidrome code in github copilot , gemini ai and comparison ?**
- **Code Execution (gemini)**

```
def is_palindrome(text):
  cleaned_text = "".join(char.lower() for char in text if char.isalnum())
  return cleaned_text == cleaned_text[::-1]

print(f"'madam' is a palindrome: {is_palindrome('madam')}")
print(f"'racecar' is a palindrome: {is_palindrome('racecar')}")
print(f"'hello' is a palindrome: {is_palindrome('hello')}")
print(f"'Python' is a palindrome: {is_palindrome('Python')}")
```

code output

```
'madam' is a palindrome: True
'racecar' is a palindrome: True
'hello' is a palindrome: False
'Python' is a palindrome: False
```

Code Execution (github copilot)

```
#generate python code for palidrome checking
def is_palindrome(s):
    """Check if a string is a palindrome."""
    cleaned_str = ''.join(c.lower() for c in s if c.isalnum())
    return cleaned_str == cleaned_str[::-1]

# Example usage
if __name__ == "__main__":
    test_strings = ["A man, a plan, a canal: Panama", "racecar", "hello", "No 'x'
    for s in test_strings:
        result = is_palindrome(s)
        print(f'"{s}" is a palindrome: {result}')
```

Code output

```
PS C:\Ai_coding> & C:/Users/mukes/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Ai
_2.py
"A man, a plan, a canal: Panama" is a palindrome: True
"racecar" is a palindrome: True
"hello" is a palindrome: False
"No 'x' in Nixon" is a palindrome: True
PS C:\Ai_coding>
```

Comparison

Step-by-step comparison of both palindrome codes:

1. **Function Definition:**
   Both codes define a function named `is_palindrome` that takes a string as input.

2. **Text Cleaning:**
   In both codes, the input string is converted to lowercase and all non-alphanumeric characters are removed to ensure accurate palindrome checking.

3. **Palindrome Logic:**
   Each code compares the cleaned string with its reverse using string slicing (`[::-1]`) to determine if it is a palindrome.

4. **Return Value:**
   Both functions return `True` if the string is a palindrome and `False` otherwise.

5. **Testing the Function:**
   The first code directly calls the function with `print()` statements, while the second code tests multiple strings inside an `if __name__ == "__main__"` block.

6. **Code Structure:**
   The first code is simple and straightforward, whereas the second code is better structured, includes a docstring, and is more suitable for larger programs.

Task 4: Code Explanation Using AI

- **Scenario:**
  You are reviewing unfamiliar code written by another developer.

- **Task:**
  Ask Gemini to explain a Python function (prime check OR palindrome check) line by line.

❖ **Expected Output:**

- Code snippet
- AI explanation
- Student comments on understanding
- **Output**
- **Prompt :** prime check this Python function line by line
- **Code execution**

```
def is_prime(number):

    if number <= 1:
        return False

    for i in range(2 (variable) i: int + 1):
        if number % i == 0:
            return False
    return True

# Example usage:
print(f"Is 7 prime? {is_prime(7)}")
print(f"Is 10 prime? {is_prime(10)}")
print(f"Is 1 prime? {is_prime(1)}")
print(f"Is 2 prime? {is_prime(2)}")
```

- **Code output**

```
Is 7 prime? True
Is 10 prime? False
Is 1 prime? False
Is 2 prime? True
```

Explanation (step by step) :

Step-by-step execution explanation for `is_prime(number)` function:

Let's trace `is_prime(7)`:

1. `def is_prime(number):`: The function `is_prime` is called with `number = 7`.
2. `if number <= 1:`: `7 <= 1` is `False`. So, this condition is skipped.
3. `for i in range(2, int(number**0.5) + 1):`: The loop starts. `number**0.5` (square root of 7) is approximately `2.64`. `int(2.64)` is `2`. So the range is `range(2, 2 + 1)`, which means `range(2, 3)`. The loop will run for `i = 2`.
4. `i = 2` **(first iteration of the loop)**:
   - `if number % i == 0:`: `7 % 2` (7 divided by 2) is `1` (remainder is 1). `1 == 0` is `False`. So, this condition is skipped.
5. The loop finishes because `i` reached the end of the range.
6. `return True`: Since no divisor was found, the function returns `True`.

Thus, `is_prime(7)` correctly returns `True`.

Let's trace `is_prime(10)`:

1. `def is_prime(number):`: The function `is_prime` is called with `number = 10`.
2. `if number <= 1:`: `10 <= 1` is `False`. So, this condition is skipped.
3. `for i in range(2, int(number**0.5) + 1):`: The loop starts. `number**0.5` (square root of 10) is approximately `3.16`. `int(3.16)` is `3`. So the range is `range(2, 3 + 1)`, which means `range(2, 4)`. The loop will run for `i = 2, 3`.
4. `i = 2` **(first iteration of the loop)**:
   - `if number % i == 0:`: `10 % 2` (10 divided by 2) is `0` (remainder is 0). `0 == 0` is `True`. So, this condition is met.
   - `return False`: The function immediately returns `False` because a divisor (2) was found.

Thus, `is_prime(10)` correctly returns `False`.

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