

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE			DEPARTMENT OF COMPUTER SCIENCE ENGINEERING
Program Name: B. Tech		Assignment Type: Lab	Academic Year: 2025-2026
CourseCode	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week1 – Monday	Batch	23CSBTB47B
Name	Yashaswini Mittapalli	Hall Ticket No	2303A54049

Assignment Number: 1.3(Present assignment number)/24(Total number of assignments)

Q. No.	Question	Expected Time to complete
1	<p>Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI</p> <p>Lab Objectives:</p> <ul style="list-style-type: none"> ❖ 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. <p>Lab Outcomes (LOs):</p> <p>After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> ❖ 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. <p>Task 1: Statistical Summary for Survey Data</p> <p>❖ Scenario: You are a data analyst intern working with survey responses stored as numerical lists.</p> <p>❖ Task: Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.</p> <p>➤ Prompt : You are an AI coding assistant. Write a Python function in Google Colab that: Accepts a list of numbers as input Calculates the mean, minimum, and maximum values</p>	Week1 - Monday

Prints the results clearly

Then run the function on the list [12, 7, 9, 21, 5] and show the output.

❖ **Expected Output:**

Input List: [12, 7, 9, 21, 5]
Mean: 10.80
Minimum: 5
Maximum: 21

➤ Output shown in Colab

```
# Run the function with the provided list
my_list = [12, 7, 9, 21, 5]
analyze_numbers(my_list)

...
--- Number Analysis Results ---
Input List: [12, 7, 9, 21, 5]
Mean: 10.80
Minimum: 5
Maximum: 21
```

➤ Screenshot of Gemini prompt and result

The screenshot shows a Google Colab notebook titled "lab21-4049.ipynb". The code cell contains a Python function named `analyze_numbers` which calculates the mean, minimum, and maximum of a list of numbers. A tooltip from Gemini provides instructions for running the function on the list [12, 7, 9, 21, 5]. The AI also asks for help building the function.

```
import statistics

def analyze_numbers(numbers):
    """
    Calculates and prints the mean, minimum, and maximum of a list of numbers.

    Args:
        numbers: A list of numerical values.

    Returns:
        None
    """
    if not numbers:
        print("The list is empty. Cannot calculate statistics.")
        return

    mean_value = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

    print("... Number Analysis Results ...")
    print("Input List: (numbers)")
    print(f"Mean: {mean_value:.2f}")
    print(f"Minimum: {min_value}")
    print(f"Maximum: {max_value}")
    print("-----")
```

You are an AI coding assistant. Write a Python function in Google Colab that: Accepts a list of numbers Calculates mean, minimum, and maximum values Prints the results clearly Run the function on [12, 7, 9, 21, 5] and show the output

💡 Sure, I can help you with that! Here's a Python function that calculates the

Explain the analyze_numbers function How to handle an empty list in the function Run

What can I help you build?

+ Gemini 2.5 Flash >

Task 2: Armstrong Number – AI Comparison

❖ **Scenario:**

You are evaluating AI tools for numeric validation logic.

❖ **Task:**

Generate an **Armstrong number checker** using **Gemini** and **GitHub Copilot**.

Compare their outputs, logic style, and clarity.

❖ **Expected Output:**

153 IS an Armstrong number.
123 IS not an Armstrong number.

Side-by-side comparison table

Point	Google Colab Version	Code (Return-based)
Function ending	print("153 is an Armstrong number")	Same both
Function usage	Can't reuse result in other logic	Can reuse in if, loops, tests
Main logic location	Logic + output mixed together	Logic inside function, output outside

➤ **Screenshots of prompts and generated code**

The screenshot shows a Google Colab notebook interface. On the left, there is a code editor with a file named 'lab1.2-4049.py'. The code contains a function 'calculate_statistics' and a function 'is_armstrong_number'. The 'is_armstrong_number' function takes a number as input, converts it to a string, calculates the length of the string (which is the power), and then iterates through each digit, raising it to the calculated power and summing the results. It returns True if the total equals the original number, and False otherwise. Below this, a loop checks if 153 and 123 are Armstrong numbers and prints the results. On the right, there is a sidebar titled 'Python list statistics' with a task description: 'Write a Python function that checks if a number is an Armstrong number. Test it with 153 and 123, and print the results clearly.' Below the task, there is a section for '3. Clear output: Prints the results with descriptive labels and proper formatting (mean shows 2 decimal places)'. It shows the output for the list [12, 7, 9, 21, 5] with mean 10.80, minimum 5, and maximum 21. There is also a note about handling empty lists.

```
Moya > lab1.2-4049.py < ...
...
25
26 test_list = [12, 7, 9, 21, 5]
27 calculate_statistics(test_list)
28
29
30 # Task 2
31 ...
32 Prompt: Armstrong Number Checker
33 Write a Python function that checks if a number is an Armstrong number.
34 Test it with 153 and 123, and print the results clearly.
35 ...
36
37 def is_armstrong_number(number):
38     num_str = str(number)
39     power = len(num_str)
40     total = sum(int(digit) ** power for digit in num_str)
41     return total == number
42
43 for num in [153, 123]:
44     if is_armstrong_number(num):
45         print(f"{num} is an Armstrong number")
46     else:
47         print(f"{num} is not an Armstrong number")
...
Python list statistics | Python + | ...
You are an AI coding assistant.
Write a Python function in Google Colab that:
3. Clear output: Prints the results with descriptive labels and proper formatting (mean shows 2 decimal places)
When run on the list [12, 7, 9, 21, 5], the output shows:
• Mean: 10.80 (correct: 54 / 5 = 10.8)
• Minimum: 5
• Maximum: 21
The function also includes error handling for empty lists. You can easily use this function with any list of numbers by calling calculate_statistics(your_list) in Google Colab or any Python environment.
...
Task: Armstrong Number Checker
Write a Python function that checks if a number is an Armstrong number.
...

```

The screenshot shows a Jupyter Notebook interface with a dark theme. The code cell contains a function named `is_armstrong` which checks if a given number is an Armstrong number. The function converts the number to a string, calculates the sum of its digits raised to the power of the number of digits, and compares it to the original number. If they are equal, it prints that the number is an Armstrong number; otherwise, it prints that it is not. The code also includes a test section at the bottom with two calls to `is_armstrong` for the numbers 153 and 123.

```
def is_armstrong(number):
    """
    Checks if a given number is an Armstrong number.

    An Armstrong number is a number that is equal to the sum of its own digits
    each raised to the power of the number of digits.

    Args:
        number: An integer.

    Returns:
        True if the number is an Armstrong number, False otherwise.
    """
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** num_digits

    if sum_of_powers == number:
        print(f"{number} is an Armstrong number.")
        return True
    else:
        print(f"{number} is not an Armstrong number.")
        return False

# Test the function with 153 and 123
is_armstrong(153)
is_armstrong(123)
```

Task 3: Leap Year Validation Using Cursor AI

❖ Scenario:

You are validating a calendar module for a backend system.

❖ Task:

Use **Cursor AI** to generate a Python program that checks whether a given year is a leap year.

Use **at least two different prompts** and observe changes in code.

Input:

1900

Output:

1900 is not a leap year

Input:

2000

Output:

2000 is a leap year

Input:

2024

Output:

2024 is a leap year

Input:

2100

Output:

2100 is not a leap year

➤ Two versions of code

```

lab1.2-4049.py

Moya > lab1.2-4049.py > ...
46     else:
47         print(f"{num} is not an Armstrong number")
48
49
50 # Task 3
51 ...
52 Prompt:
53 Write a Python function is_leap_year(year) that returns True if the year is a leap year, otherwise False.
54 Test it with 2000, 2021, 2024, and 2100.
55 ...
56
57 def is_leap_year(year):
58     if year % 4 == 0:
59         if year % 100 == 0:
60             return year % 400 == 0
61         return True
62     return False
63
64 for year in [2000, 2021, 2024, 2100]:
65     if is_leap_year(year):
66         print(f"{year} is a leap year")
67     else:
68         print(f"{year} is not a leap year")
69

```

Task: Armstrong Number Checker
Write a Python function that checks if a number is an Armstrong number.

Test Results:

- 153 is an Armstrong number because $1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$
- 123 is not an Armstrong number because $1^3 + 2^3 + 3^3 = 1 + 8 + 27 = 36 \neq 123$

The function correctly identifies Armstrong numbers by checking if a number equals the sum of its digits each raised to the power of the total number of digits. You can use this function with any positive integer by calling `is_armstrong_number(your_number)`.

Thought for 5s

```

lab2.1-4054.py > ⚡ check_leap_year
def check_leap_year(year):
    Returns:
        None: Prints detailed explanation of leap year status
    """
    print(f"\nChecking year: {year}")
    # Step 1: Check if divisible by 4
    if year % 4 != 0:
        print(f" {year} is not divisible by 4.")
        print(f" Therefore, {year} is NOT a leap year.")
        return
    print(f" {year} is divisible by 4.")

    # Step 2: If divisible by 4, check if it's a century year (divisible by 100)
    if year % 100 == 0:
        print(f" {year} is a century year (divisible by 100).")

    # Step 3: For century years, must be divisible by 400
    if year % 400 == 0:
        print(f" {year} is also divisible by 400.")
        print(f" Therefore, {year} IS a leap year.")
    else:
        print(f" {year} is not divisible by 400.")
        print(f" Therefore, {year} is NOT a leap year.")

    else:
        print(f" {year} is not a century year.")
        print(f" Therefore, {year} IS a leap year.")

# Test the function with given years
test_years = [1900, 2000, 2024, 2100]
print("Leap Year Analysis:")
print("-" * 50)
for year in test_years:
    check_leap_year(year)

```

➤ Brief comparison :

Key Difference in Code Style

```

if year % 4 == 0:
    if year % 100 == 0:
        return year % 400 == 0
    else:
        return True
else:
    return False

```

Compact style

```
return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```

Task 4: Student Logic + AI Refactoring (Odd/Even Sum)

❖ Scenario:

Company policy requires developers to write logic before using AI.

❖ Task:

Write a Python program that calculates the **sum of odd and even numbers in a tuple**, then refactor it using any AI tool.

Prompt:

Write a Python program to find the sum of odd numbers and sum of even numbers in a tuple using basic logic.

Then refactor the code using an AI tool to make it simpler.

Show:

1. Original code

```
# Original code |
def calculate_sums_original(numbers):
    even_sum = 0
    odd_sum = 0

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

    return even_sum, odd_sum
```

2. Refactored code

```
# Refactored code (AI Version)
def calculate_sums_refactored(numbers):
    even_sum = sum(num for num in numbers if num % 2 == 0)
    odd_sum = (variable) even_sum: int else: if num % 2 != 0)
    return even_sum, odd_sum

test_tuple = (1, 2, 3, 4, 5, 6)
```

3. Short explanation of changes

Input: (1, 2, 3, 4, 5, 6)

❖ **Expected Output:**

Original Code Output:

Even sum: 12

Odd sum: 9

Refactored Code Output:

Even sum: 12

Odd sum: 9

➤ **Explanation of improvements**

1. Code is **simpler and easier to read**
2. No need to write loops manually
3. Uses Python's built-in `sum()` function for clarity