

# AI ASSISTED CODING

## Assignment – 2.1

**Name : SHAIK NAVED AHMED**

**Roll number: 2303A54053**

**Batch : 47B**

**Lab 2: Exploring Additional AI Coding Tools beyond Copilot – Gemini (Colab) and Cursor AI**

**Week: 1**

**Day: Monday**

**Task 1: Statistical Summary for Survey Data**

**Scenario**

You are a data analyst intern working with survey responses stored as numerical lists.

**PROMPT - Write a Python function that takes a list of numbers and returns the mean, minimum, and maximum values. Also show sample output."**

Python

```
def get_list_stats(numbers):  
    """  
    Calculates the mean, minimum, and maximum of a list of numbers.  
    """  
    if not numbers:  
        return None, None, None  
  
    # Calculate statistics  
    mean_val = sum(numbers) / len(numbers)  
    min_val = min(numbers)  
    max_val = max(numbers)  
  
    return mean_val, min_val, max_val  
  
# Sample Usage  
data = [10, 20, 30, 40, 50]  
mean, minimum, maximum = get_list_stats(data)  
  
print(f"List: {data}")  
print(f"Mean: {mean}")  
print(f"Minimum: {minimum}")  
print(f"Maximum: {maximum}")
```

### Sample Output

When you run the code above, the output will look like this:

Plaintext

```
List: [10, 20, 30, 40, 50]
Mean: 30.0
Minimum: 10
Maximum: 50
```

## Task 2: Armstrong Number – AI Comparison

### Scenario

You are evaluating AI tools for numeric validation logic.

**PROMPT:** generate an armstrong number checker code

Gemini code:

Python

```
def is_armstrong(number):
    # Convert to string to easily iterate over digits
    str_num = str(number)
    num_digits = len(str_num)

    # Calculate the sum of digits raised to the power of num_digits
    total_sum = sum(int(digit) ** num_digits for digit in str_num)

    # Check if the sum equals the original number
    return total_sum == number

# --- Testing the function ---
test_numbers = [153, 370, 9474, 123]

for n in test_numbers:
    if is_armstrong(n):
        print(f"{n} is an Armstrong number.")
    else:
        print(f"{n} is not an Armstrong number.")
```

GITHUB CO PILOT CODE:

```

1  #!/usr/bin/env python3
2  import sys
3
4  def is_armstrong(n: int) -> bool:
5      if n < 0:
6          return False
7      s = str(n)
8      power = len(s)
9      return sum(int(d) ** power for d in s) == n
10
11 def main() -> None:
12     if len(sys.argv) < 2:
13         print("Usage: python armstrong_checker.py <number>", file=sys.stderr)
14         sys.exit(2)
15     try:
16         n = int(sys.argv[1])
17     except ValueError:
18         print(f"Invalid integer: {sys.argv[1]!r}", file=sys.stderr)
19         sys.exit(2)
20     print(f"{n}: {'Armstrong' if is_armstrong(n) else 'Not Armstrong'}")
21
22 if __name__ == "__main__":
23     main()

```

Aspect	GitHub Copilot Code	Gemini Code
Input Method	Command-line argument using sys.argv	User input using input()
Execution Style	Script-style (terminal-based, professional)	Interactive (beginner-friendly)
Output Format	<number>: Armstrong / Not Armstrong	<number> is / is not an Armstrong number!
Negative Number Handling	Explicitly checks n < 0 and returns False	✗ No explicit negative number

Aspect	GitHub Copilot Code	Gemini Code
Error Handling	✓ Robust (handles missing input & invalid integers)	✗ No error handling (crashes if input is invalid)
Function Definition	Uses type hints (n: int -> bool)	No type hints
Logic Used	Converts number to string, uses digit power sum	Same logic (string + digit power sum)
Reusability	✓ High (modular, reusable, CLI-ready)	⚠ Medium (tied to user input)
Clarity of Code	More professional, slightly complex	Very simple and easy to understand
Best For	Production scripts, automation, CLI tools	Beginners, learning concepts

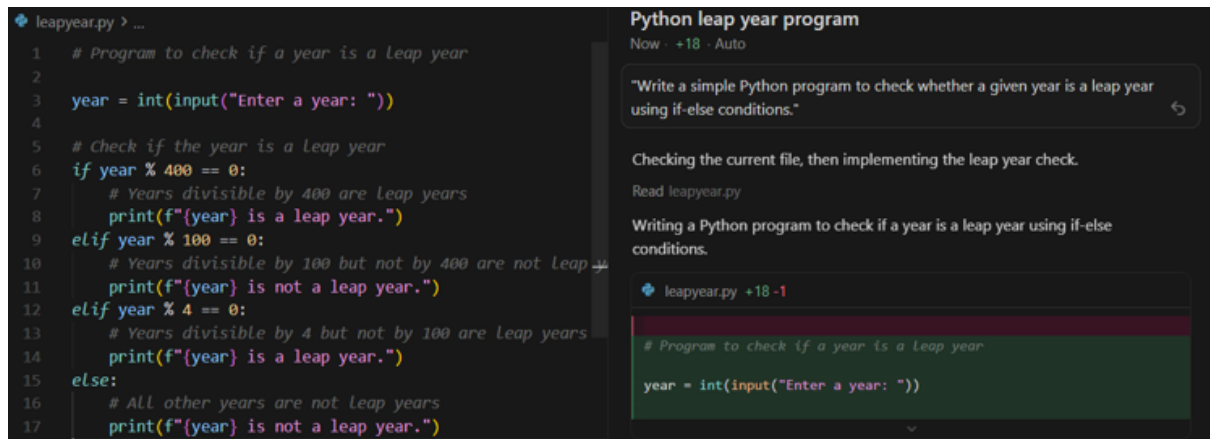
### Task 3: Leap Year Validation Using Cursor AI

#### ❖ Scenario:

You are validating a calendar module for a backend system.

**PROMPT1 :** "Write a simple Python program to check whether a given year is a leap year using if-else conditions.

**CODE:**



INPUT: 2024

OUTPUT: LEAP YEAR

PROMPT 2: Generate a professional Python program using functions and error handling to validate whether a given year is a leap year for a backend system.

CODE:

def **is\_leap\_year**(year: int) -> bool:

    if year % 400 == 0:

        return True

    if year % 100 == 0:

        return False

    if year % 4 == 0:

        return True

    return False

2

def **validate\_year**(year) -> dict:

    try:

        year\_int = int(year)

        is\_leap = is\_leap\_year(year\_int)

        return {

            'year': year\_int

            'is\_leap': is\_leap,

            'status': 'success'

        }

    except (TypeError, ValueError) as e:

        return {

```
'year': year,
'is_leap': False,
'status': 'error',
'message': str(e)
}
```

INPUT /OUTPUT:

```
{
'year': 2024,
'is_leap': True,
'status': 'success'
}
```

EXPLANATION: **Prompt 1** produces a direct printed result, while **Prompt 2** produces a structured and validated output.

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

❖ Scenario: Company policy requires developers to write logic before using AI.

CODE: numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)

```
even_sum = 0
```

```
odd_sum = 0
```

```
for num in numbers:
```

```
    if num % 2 == 0:
```

```
        even_sum = even_sum + num
```

```
    else:
```

```
        odd_sum = odd_sum + num
```

```
print("Sum of even numbers:", even_sum)
```

```
print("Sum of odd numbers:", odd_sum)
```


REFACTORING CODE PROMPT: Refactor the given Python program that calculates the sum of odd and even numbers in a tuple.

Optimize the code using Python best practices, reduce lines of code, and improve readability without changing the output.

CODE:

## ✓ Refactored Code

python

 Copy code

```
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9)

even_sum = sum(num for num in numbers if num % 2 == 0)
odd_sum = sum(num for num in numbers if num % 2 != 0)

print(f"Sum of even numbers: {even_sum}")
print(f"Sum of odd numbers: {odd_sum}")
```

OUTPUT:

Sum of even numbers:

Sum of odd numbers:

EXPLANATION:

1. Replaced manual loops with Python's built-in `sum()` and generator expressions.
2. Reduced code length and improved clarity without changing the program output.