AI ASSISTANT CODING

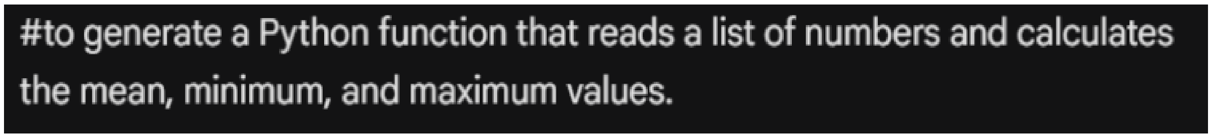
ASSIGNMENT-2.1

Humera Mahaveen

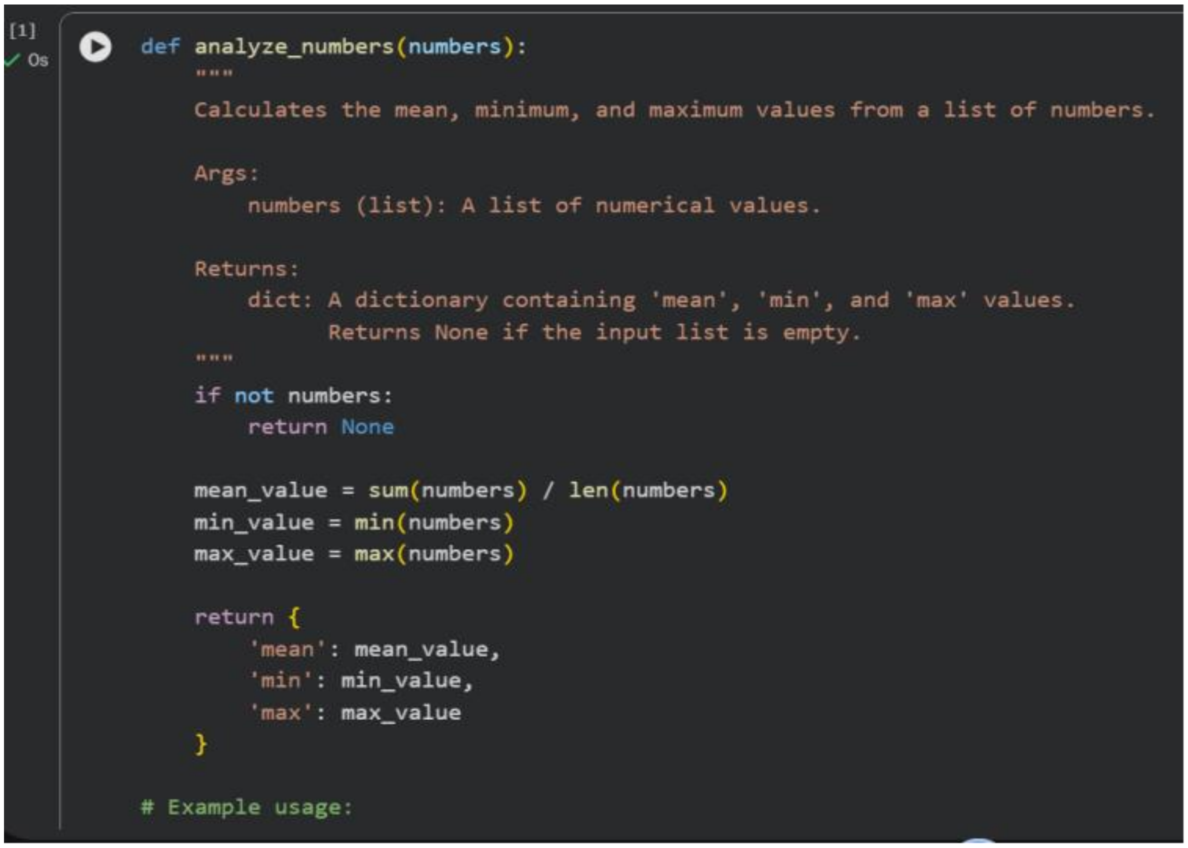
2303A53030

B-46

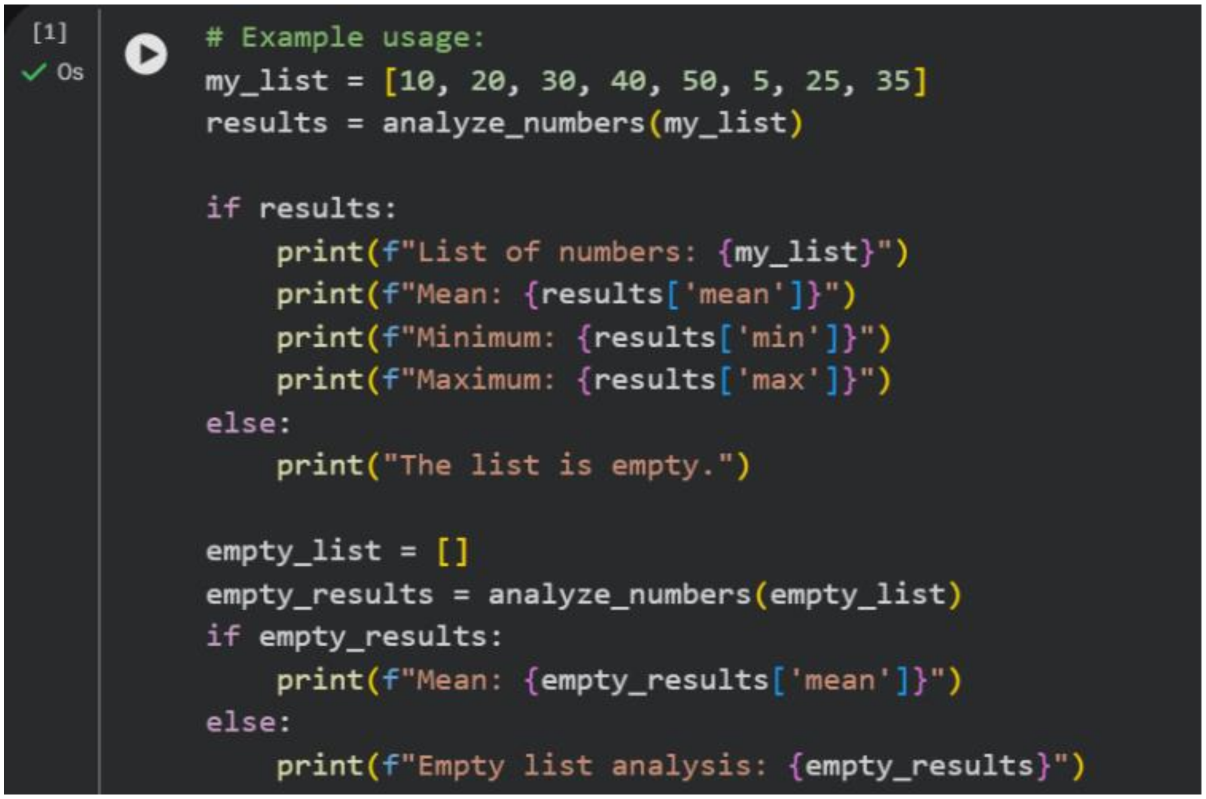
Task 1: Statistical Summary for Survey Data



#to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

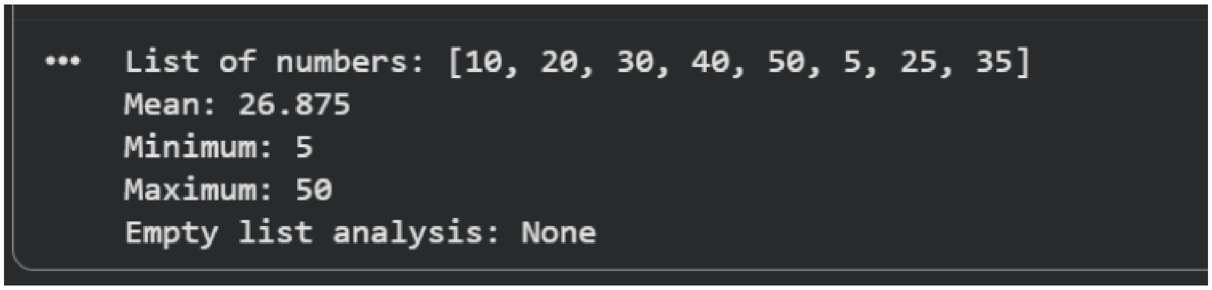


```
[1]
def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum values from a list of numbers.

    Args:
        numbers (list): A list of numerical values.

    Returns:
        dict: A dictionary containing 'mean', 'min', and 'max' values.
              Returns None if the input list is empty.
    """
    if not numbers:
        return None

    mean_value = sum(numbers) / len(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

    return {
        'mean': mean_value,
        'min': min_value,
        'max': max_value
    }

# Example usage:
```



```
[1]
✓ 0s

# Example usage:
my_list = [10, 20, 30, 40, 50, 5, 25, 35]
results = analyze_numbers(my_list)

if results:
    print(f"List of numbers: {my_list}")
    print(f"Mean: {results['mean']}")
    print(f"Minimum: {results['min']}")
    print(f"Maximum: {results['max']}")
else:
    print("The list is empty.")

empty_list = []
empty_results = analyze_numbers(empty_list)
if empty_results:
    print(f"Mean: {empty_results['mean']}")
else:
    print(f"Empty list analysis: {empty_results}")
```

OUTPUT:

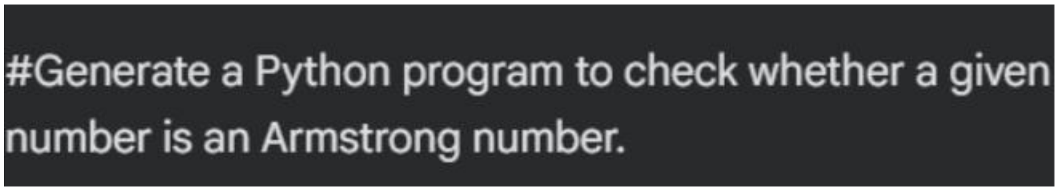


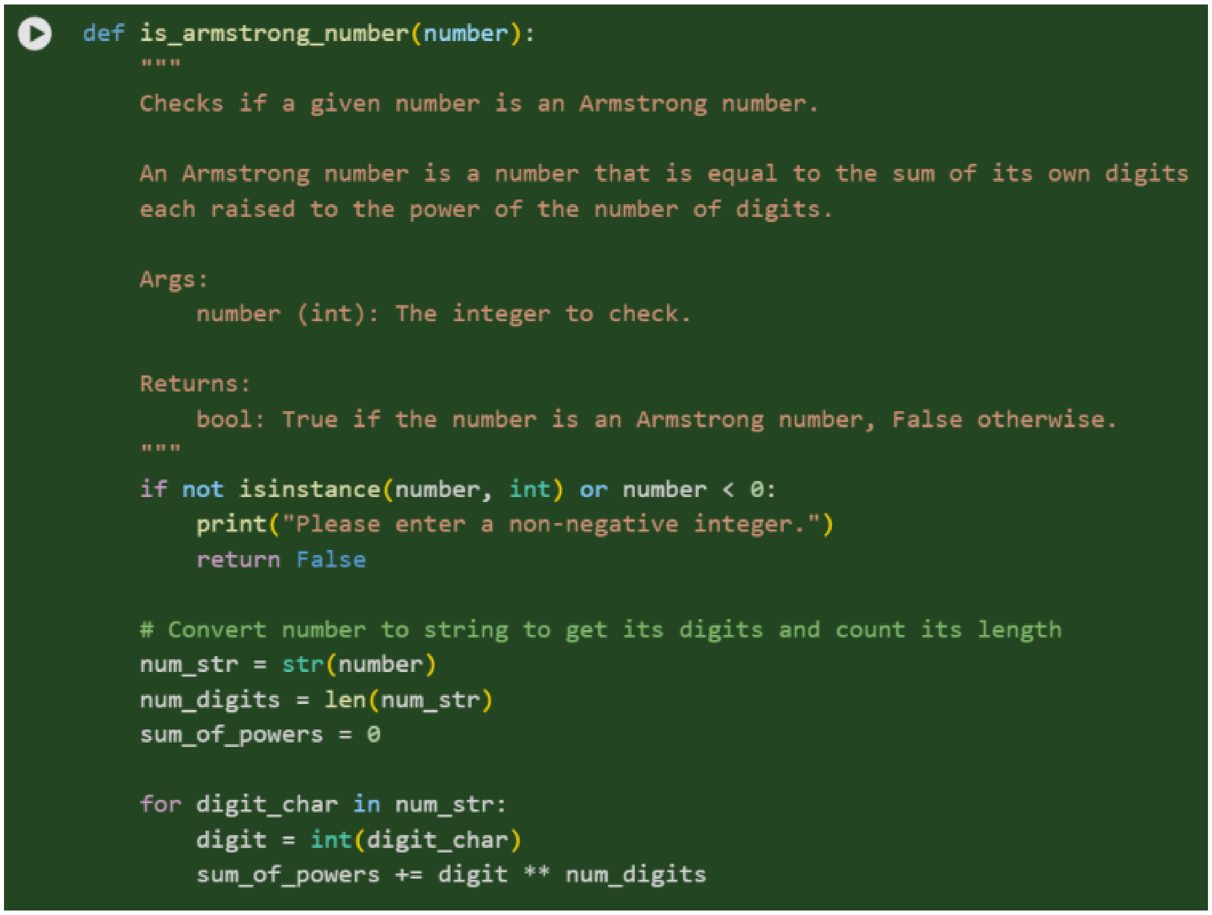
```
List of numbers: [10, 20, 30, 40, 50, 5, 25, 35]
Mean: 26.875
Minimum: 5
Maximum: 50
Empty list analysis: None
```

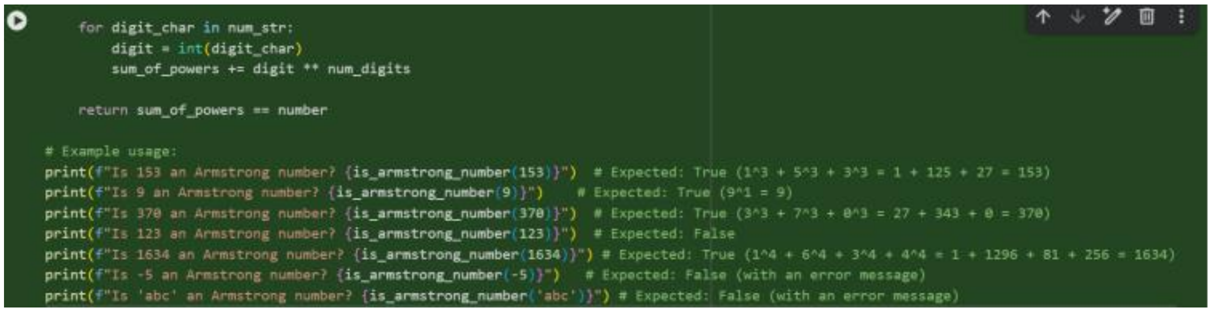
OBSERVATION:

Gemini generated a correct Python function to calculate the mean, minimum, and maximum values from a list.
The output in Colab was accurate and clearly displayed.
Overall, Gemini made the task easy by providing clear and usable code quickly.

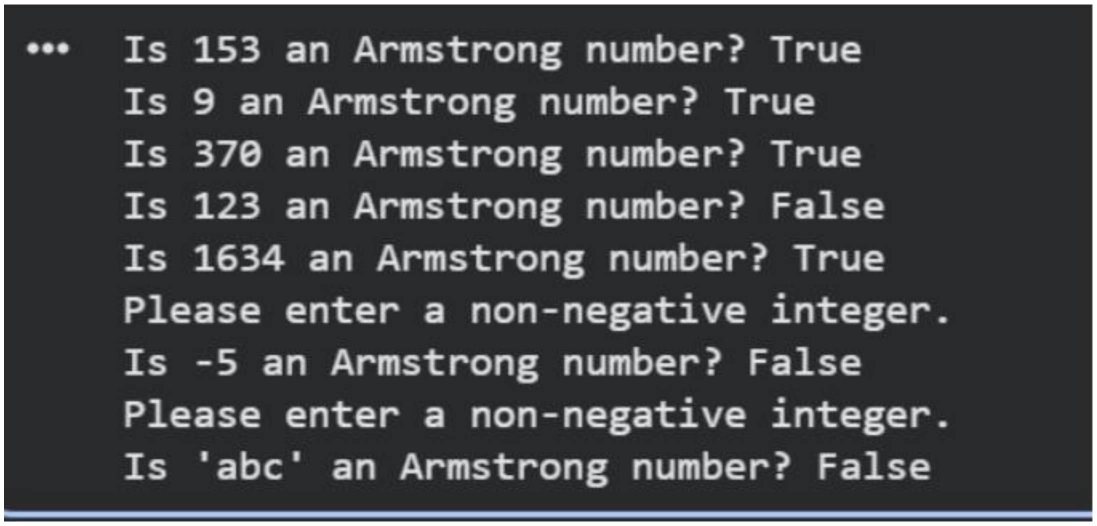
TASK 2: Armstrong Number – AI Comparison


#Generate a Python program to check whether a given number is an Armstrong number.

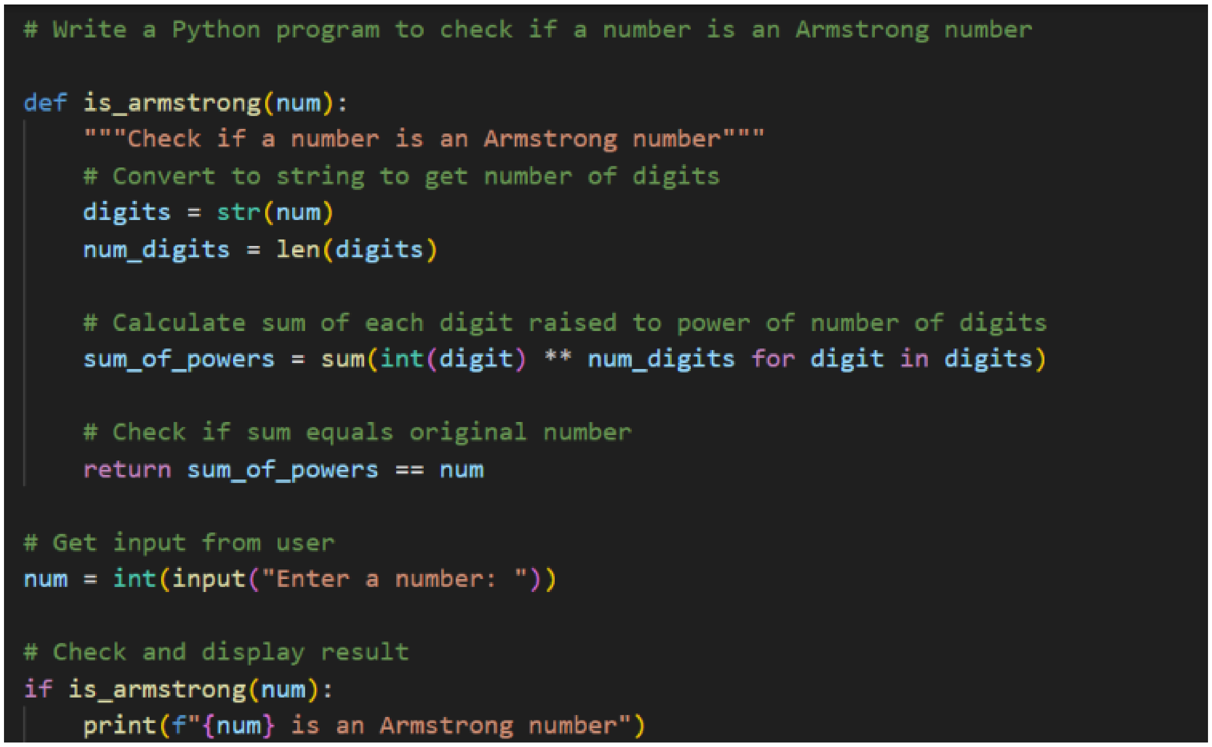

```python
def is_armstrong_number(number):
    """
    Checks if a given number is an Armstrong number.

    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 (int): The integer to check.

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        print("Please enter a non-negative integer.")
        return False

    # Convert number to string to get its digits and count its length
    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
```


```python
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** num_digits

    return sum_of_powers == number

# Example usage:
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}")  # Expected: True (1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153)
print(f"Is 9 an Armstrong number? {is_armstrong_number(9)}")    # Expected: True (9^1 = 9)
print(f"Is 370 an Armstrong number? {is_armstrong_number(370)}")  # Expected: True (3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370)
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}")  # Expected: False
print(f"Is 1634 an Armstrong number? {is_armstrong_number(1634)}") # Expected: True (1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634)
print(f"Is -5 an Armstrong number? {is_armstrong_number(-5)}")   # Expected: False (with an error message)
print(f"Is 'abc' an Armstrong number? {is_armstrong_number('abc')}") # Expected: False (with an error message)
```

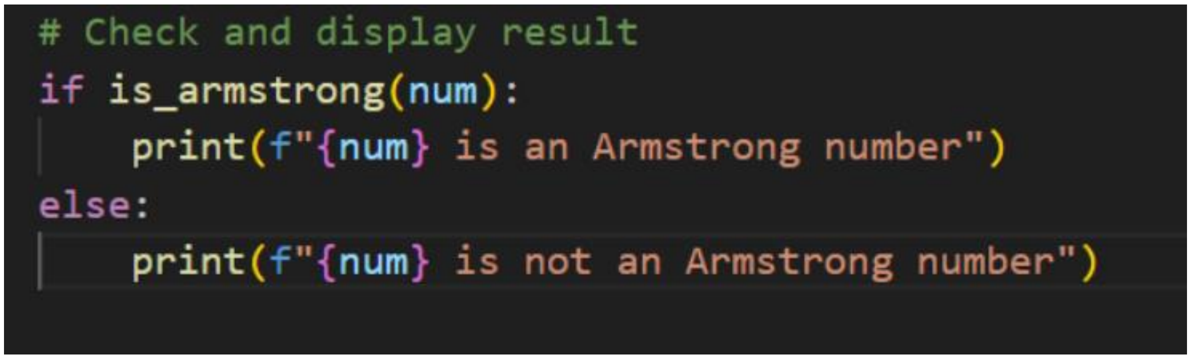
OUTPUT:



```
•••  Is 153 an Armstrong number? True
     Is 9 an Armstrong number? True
     Is 370 an Armstrong number? True
     Is 123 an Armstrong number? False
     Is 1634 an Armstrong number? True
     Please enter a non-negative integer.
     Is -5 an Armstrong number? False
     Please enter a non-negative integer.
     Is 'abc' an Armstrong number? False
```

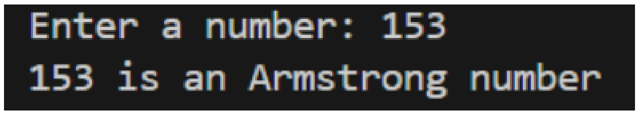


```python
# Write a Python program to check if a number is an Armstrong number

def is_armstrong(num):
    """Check if a number is an Armstrong number"""
    # Convert to string to get number of digits
    digits = str(num)
    num_digits = len(digits)

    # Calculate sum of each digit raised to power of number of digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in digits)

    # Check if sum equals original number
    return sum_of_powers == num

# Get input from user
num = int(input("Enter a number: "))

# Check and display result
if is_armstrong(num):
    print(f"{num} is an Armstrong number")
```



```python
# Check and display result
if is_armstrong(num):
    print(f"{num} is an Armstrong number")
else:
    print(f"{num} is not an Armstrong number")
```

OUTPUT:

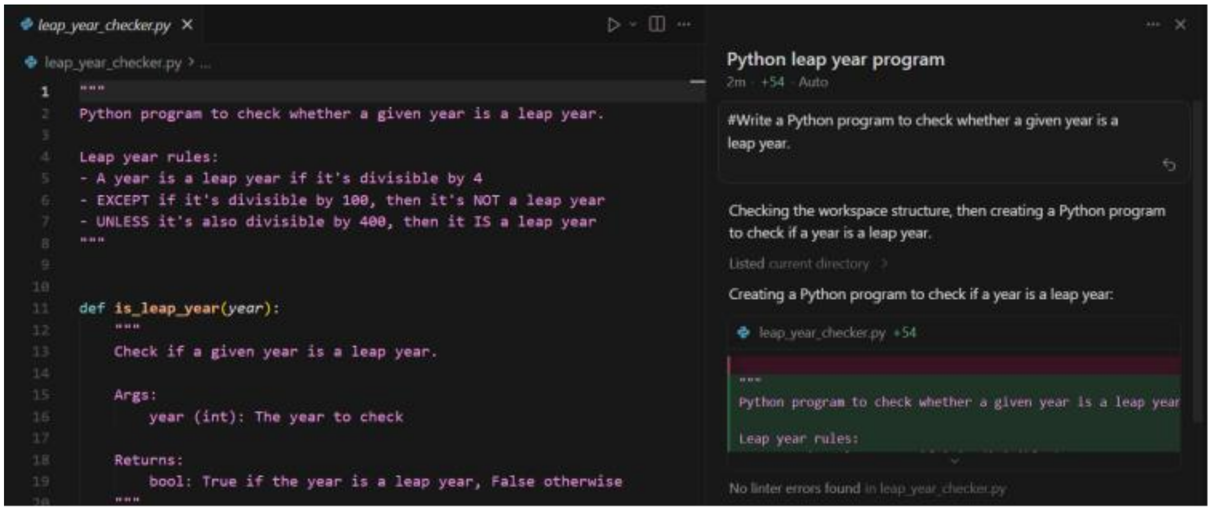


```
Enter a number: 153
153 is an Armstrong number
```

OBSERVATION:

Both Gemini and GitHub Copilot correctly generated Armstrong number checker programs.
Gemini’s code is easy to understand with step-by-step logic, making it suitable for beginners.
GitHub Copilot’s code is shorter and more optimized but slightly harder to understand.
Overall, Gemini focuses on clarity while Copilot focuses on efficiency.

Task 3: Leap Year Validation Using Cursor AI



```python
"""
Python program to check whether a given year is a leap year.

Leap year rules:
- A year is a leap year if it's divisible by 4
- EXCEPT if it's divisible by 100, then it's NOT a leap year
- UNLESS it's also divisible by 400, then it IS a leap year
"""


def is_leap_year(year):
    """
    Check if a given year is a leap year.

    Args:
        year (int): The year to check

    Returns:
        bool: True if the year is a leap year, False otherwise
    """
```

Python leap year program

2m · +54 · Auto

#Write a Python program to check whether a given year is a leap year.

Checking the workspace structure, then creating a Python program to check if a year is a leap year.

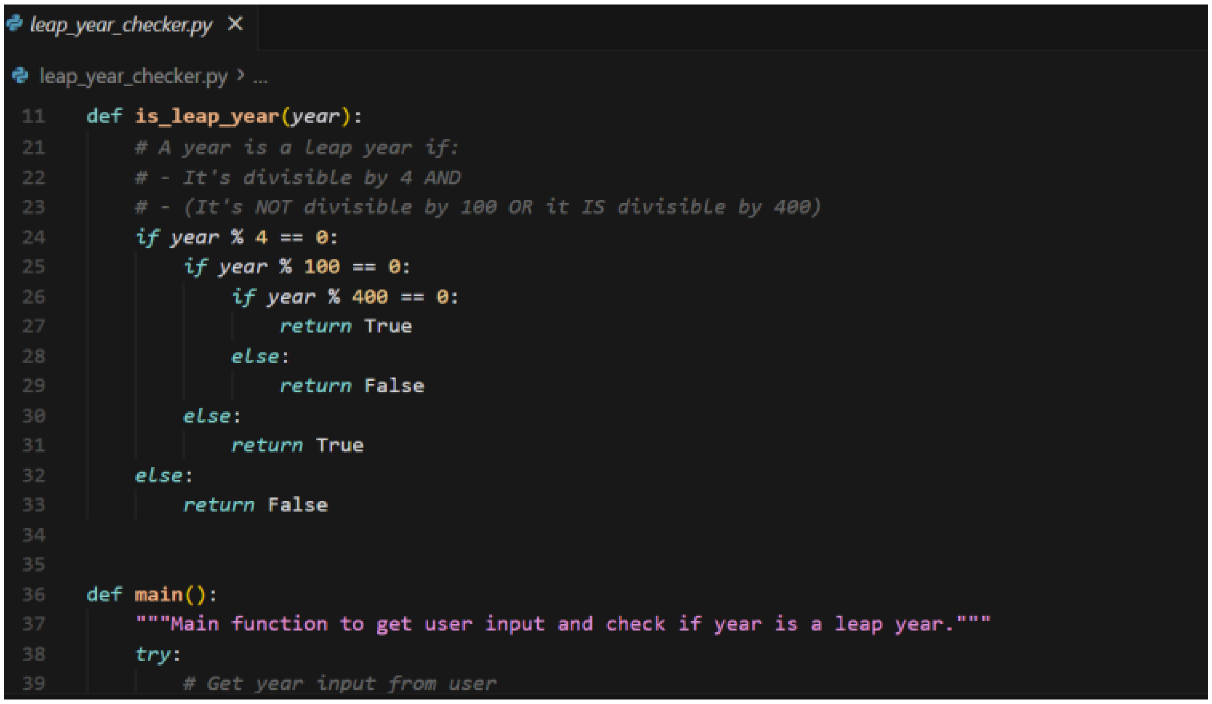
Listed current directory

Creating a Python program to check if a year is a leap year:

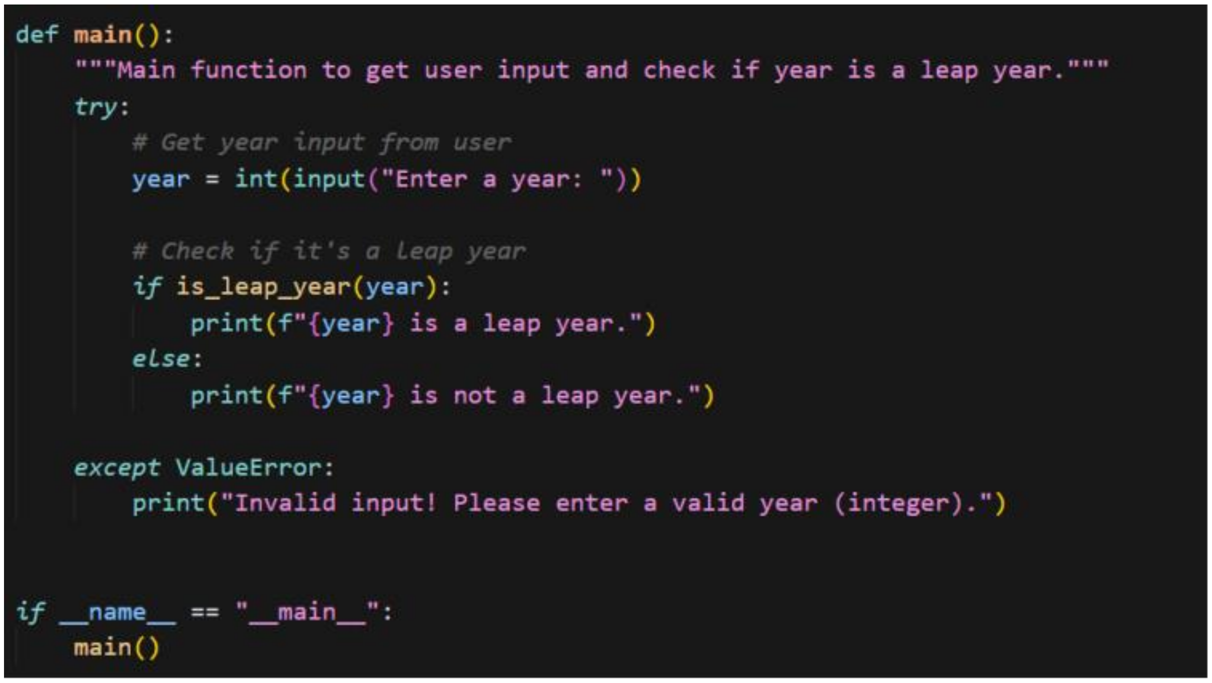
leap_year_checker.py +54

```
"""
Python program to check whether a given year is a leap year

Leap year rules:
```

No linter errors found in leap_year_checker.py

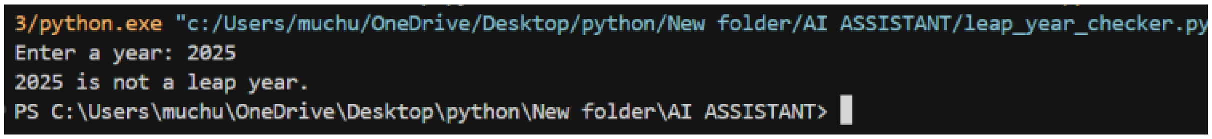


```python
def is_leap_year(year):
    # A year is a leap year if:
    # - It's divisible by 4 AND
    # - (It's NOT divisible by 100 OR it IS divisible by 400)
    if year % 4 == 0:
        if year % 100 == 0:
            if year % 400 == 0:
                return True
            else:
                return False
        else:
            return True
    else:
        return False


def main():
    """Main function to get user input and check if year is a leap year."""
    try:
        # Get year input from user
```



```python
def main():
    """Main function to get user input and check if year is a leap year."""
    try:
        # Get year input from user
        year = int(input("Enter a year: "))

        # Check if it's a leap year
        if is_leap_year(year):
            print(f"{year} is a leap year.")
        else:
            print(f"{year} is not a leap year.")

    except ValueError:
        print("Invalid input! Please enter a valid year (integer).")


if __name__ == "__main__":
    main()
```

OUTPUT:

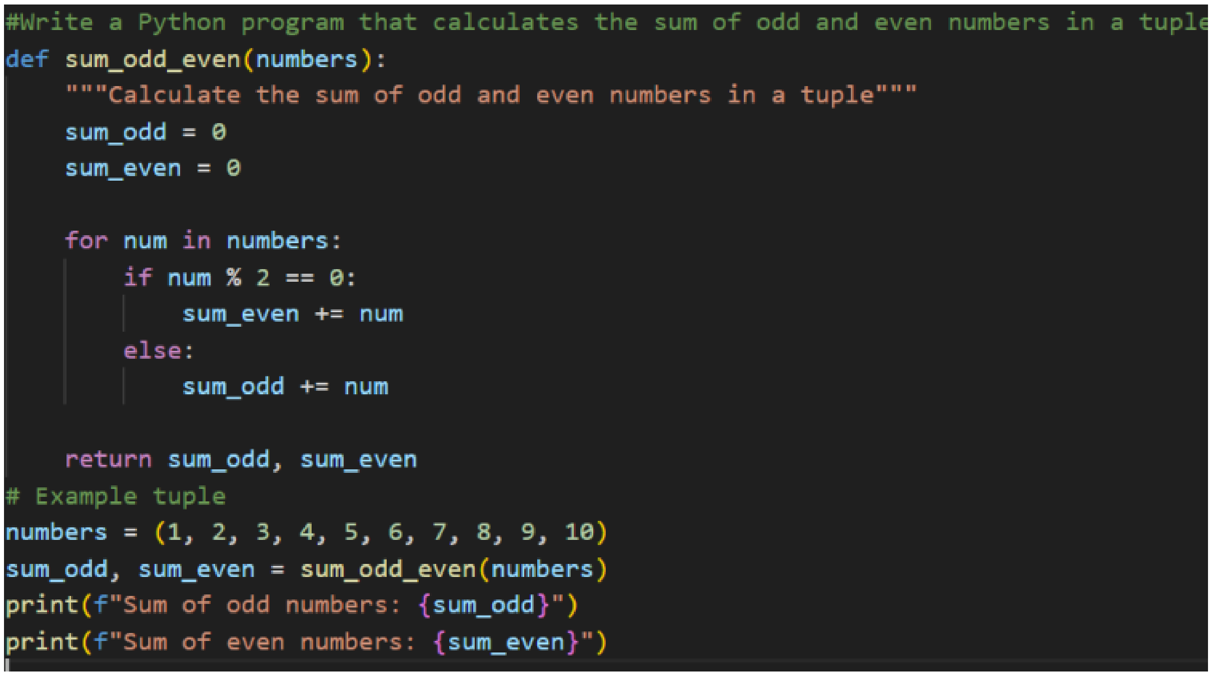


```
3/python.exe "c:/Users/muchu/OneDrive/Desktop/python/New folder/AI ASSISTANT/leap_year_checker.py
Enter a year: 2025
2025 is not a leap year.
PS C:\Users\muchu\OneDrive\Desktop\python\New folder\AI ASSISTANT> 
```

OBSERVATION:

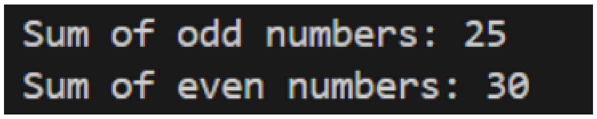
Cursor AI generated correct leap year validation code for both prompts.
With a simple prompt, the code was basic and direct, while the detailed prompt produced cleaner and more reusable function-based code.
This shows that Cursor AI changes its coding style based on how the prompt is written.

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



```python
#Write a Python program that calculates the sum of odd and even numbers in a tuple
def sum_odd_even(numbers):
    """Calculate the sum of odd and even numbers in a tuple"""
    sum_odd = 0
    sum_even = 0

    for num in numbers:
        if num % 2 == 0:
            sum_even += num
        else:
            sum_odd += num

    return sum_odd, sum_even
# Example tuple
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
sum_odd, sum_even = sum_odd_even(numbers)
print(f"Sum of odd numbers: {sum_odd}")
print(f"Sum of even numbers: {sum_even}")
```

OUTPUT:



```
Sum of odd numbers: 25
Sum of even numbers: 30
```

OBSERVATION:

The original code worked correctly but was lengthy.
After AI refactoring, the code became cleaner and more efficient.
AI improved readability without changing the logic