AI ASSISTED CODING

Lab Assignment-2.1

H.T.No:2303A510B4

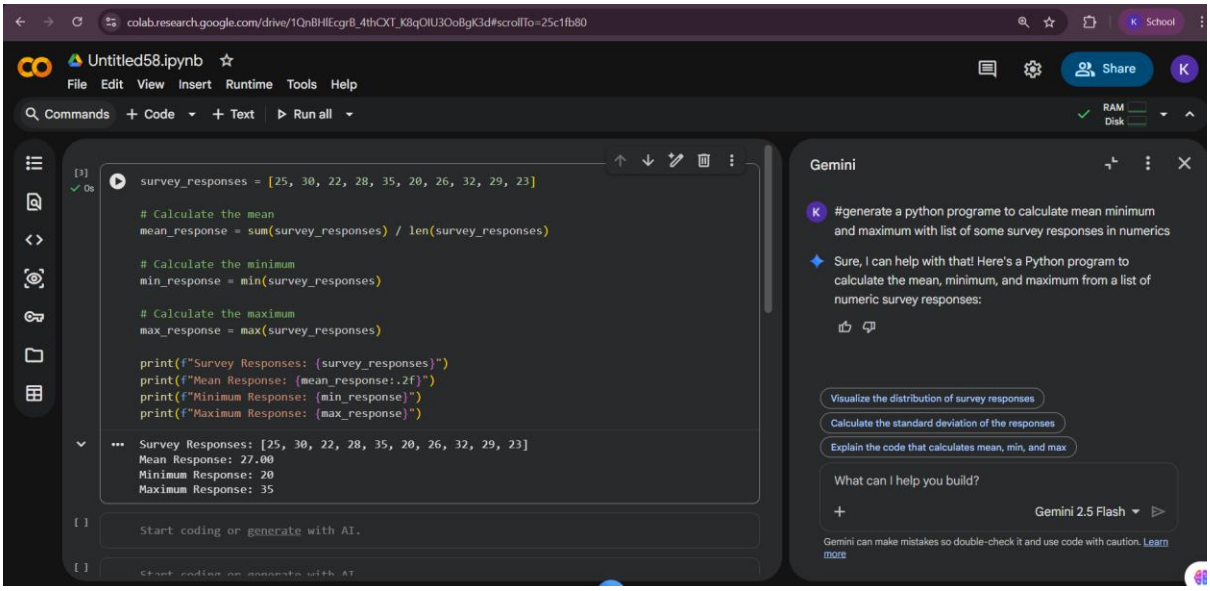
Batch:14

Task 1: Statistical Summary for Survey Data

#Prompt

generate a python programe to calculate mean minimum and maximum with list of some survey responses in numerics

Code:



```
survey_responses = [25, 30, 22, 28, 35, 20, 26, 32, 29, 23]

# Calculate the mean
mean_response = sum(survey_responses) / len(survey_responses)

# Calculate the minimum
min_response = min(survey_responses)

# Calculate the maximum
max_response = max(survey_responses)

print(f"Survey Responses: {survey_responses}")
print(f"Mean Response: {mean_response:.2f}")
print(f"Minimum Response: {min_response}")
print(f"Maximum Response: {max_response}")
```

```
Survey Responses: [25, 30, 22, 28, 35, 20, 26, 32, 29, 23]
Mean Response: 27.00
Minimum Response: 20
Maximum Response: 35
```

Gemini

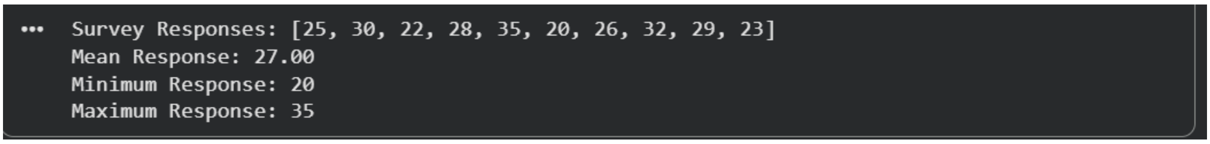
#generate a python programe to calculate mean minimum and maximum with list of some survey responses in numerics

Sure, I can help with that! Here's a Python program to calculate the mean, minimum, and maximum from a list of numeric survey responses:

Gemini can make mistakes so double-check it and use code with caution. Learn more

colab.research.google.com/drive/1QnBHlEcgrB_4thCXT_K8qOIU3OoBgK3d#scrollTo=25c1fb80
School
Untitled58.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Share
Commands
+ Code
+ Text
Run all
RAM
Disk
[3]
0s
Start coding or generate with AI.
Visualize the distribution of survey responses
Calculate the standard deviation of the responses
Explain the code that calculates mean, min, and max
What can I help you build?
Gemini 2.5 Flash

Output:



```
•••  Survey Responses: [25, 30, 22, 28, 35, 20, 26, 32, 29, 23]
     Mean Response: 27.00
     Minimum Response: 20
     Maximum Response: 35
```

Justification:

Google Gemini efficiently generated a correct Python function to compute the mean, minimum, and maximum from survey data.

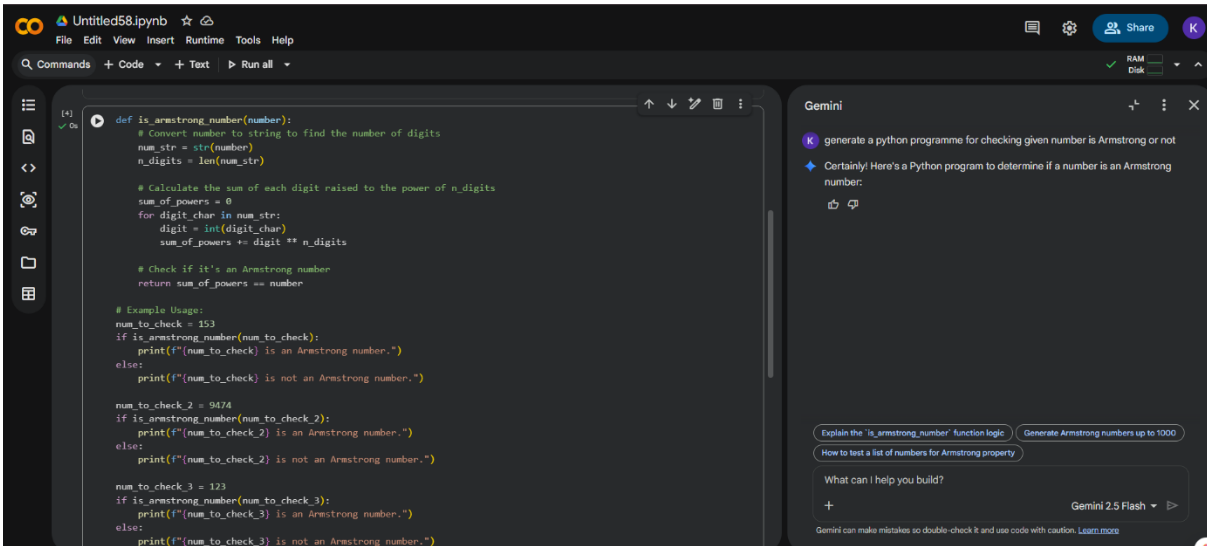
The output displayed in Colab verified the accuracy of the generated logic with sample inputs.

Task-2: Armstrong Number – AI Comparison

#Prompt:

Using google colab

Code:


Untitled58.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM
Disk
[4]
0s
Gemini
Explain the `is_armstrong_number` function logic
Generate Armstrong numbers up to 1000
How to test a list of numbers for Armstrong property
What can I help you build?
Gemini 2.5 Flash

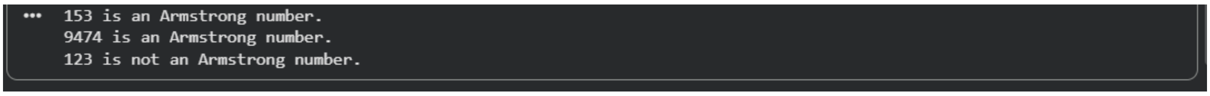
```python
def is_armstrong_number(number):
    # Convert number to string to find the number of digits
    num_str = str(number)
    n_digits = len(num_str)

    # Calculate the sum of each digit raised to the power of n_digits
    sum_of_powers = 0
    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** n_digits

    # Check if it's an Armstrong number
    return sum_of_powers == number

# Example Usage:
num_to_check = 153
if is_armstrong_number(num_to_check):
    print(f"{num_to_check} is an Armstrong number.")
else:
    print(f"{num_to_check} is not an Armstrong number.")

num_to_check_2 = 9474
if is_armstrong_number(num_to_check_2):
    print(f"{num_to_check_2} is an Armstrong number.")
else:
    print(f"{num_to_check_2} is not an Armstrong number.")

num_to_check_3 = 123
if is_armstrong_number(num_to_check_3):
    print(f"{num_to_check_3} is an Armstrong number.")
else:
    print(f"{num_to_check_3} is not an Armstrong number.")
```

K generate a python programme for checking given number is Armstrong or not

Certainly! Here's a Python program to determine if a number is an Armstrong number:

Gemini can make mistakes so double-check it and use code with caution. Learn more

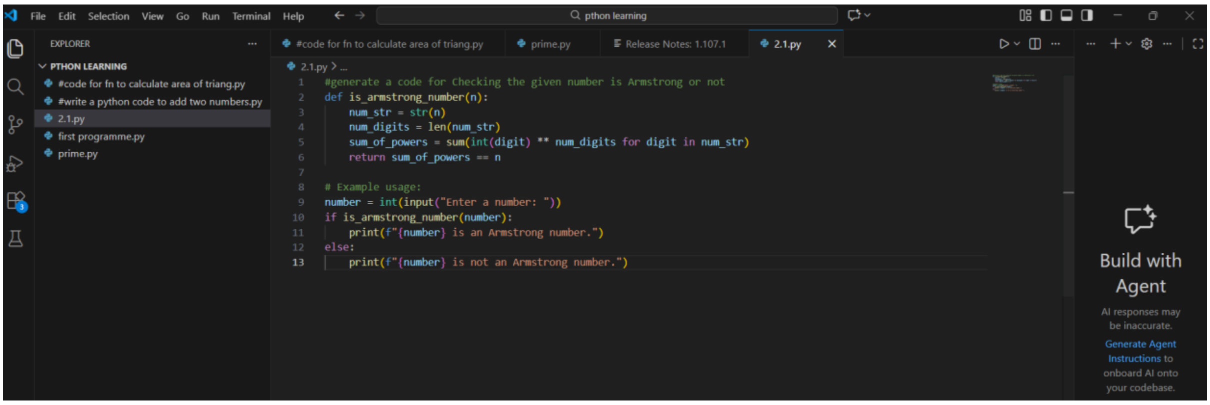
Output:



```
153 is an Armstrong number.
9474 is an Armstrong number.
123 is not an Armstrong number.
```

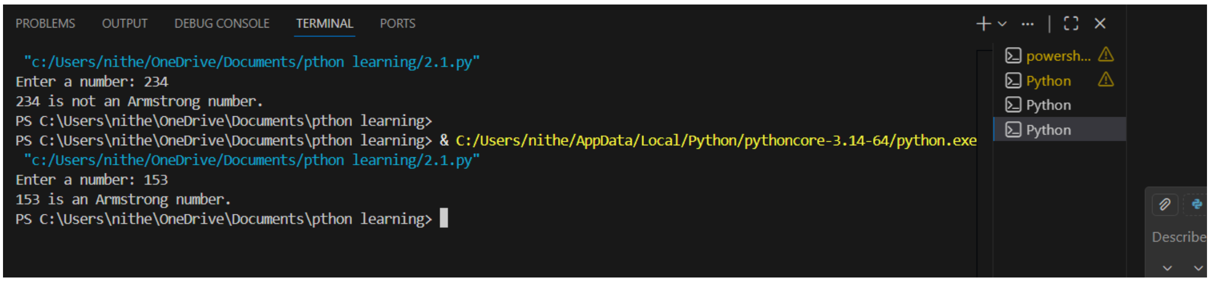
Using Github copilot

Code:


File Edit Selection View Go Run Terminal Help
pthon learning
EXPLORER
PTHON LEARNING
#code for fn to calculate area of triang.py
#write a python code to add two numbers.py
2.1.py
first programme.py
prime.py
#code for fn to calculate area of triang.py
prime.py
Release Notes: 1.107.1
2.1.py
2.1.py > ...
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.

```python
#generate a code for Checking the given number is Armstrong or not
def is_armstrong_number(n):
    num_str = str(n)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == n

# Example usage:
number = int(input("Enter a number: "))
if is_armstrong_number(number):
    print(f"{number} is an Armstrong number.")
else:
    print(f"{number} is not an Armstrong number.")
```

Output:



```
 "c:/Users/nithe/OneDrive/Documents/pthon learning/2.1.py"
Enter a number: 234
234 is not an Armstrong number.
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
PS C:\Users\nithe\OneDrive\Documents\pthon learning> & C:/Users/nithe/AppData/Local/Python/pythoncore-3.14-64/python.exe
 "c:/Users/nithe/OneDrive/Documents/pthon learning/2.1.py"
Enter a number: 153
153 is an Armstrong number.
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
```

Justification:

Gemini generated a clear, beginner-friendly Armstrong number solution with step-by-step logic and explanation.

GitHub Copilot produced a more concise and optimized implementation but with minimal explanation.

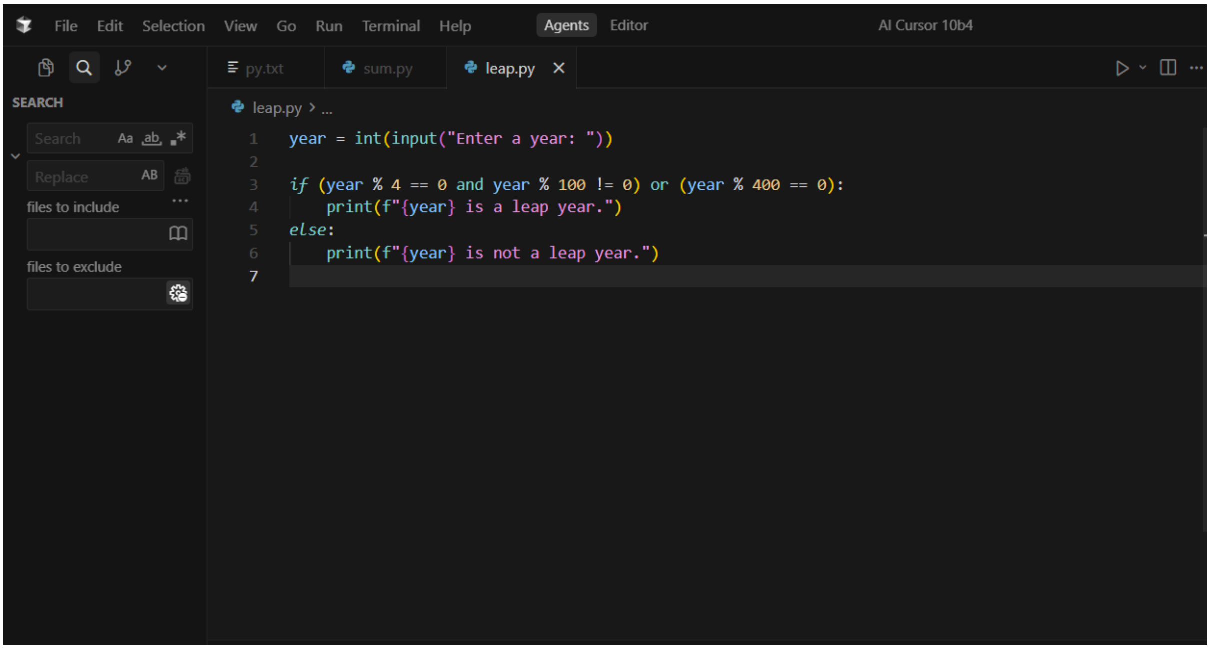
This comparison shows that Gemini prioritizes clarity and learning, while Copilot focuses on efficiency and developer productivity.

Task-3: Leap Year Validation Using Cursor AI

#Prompt

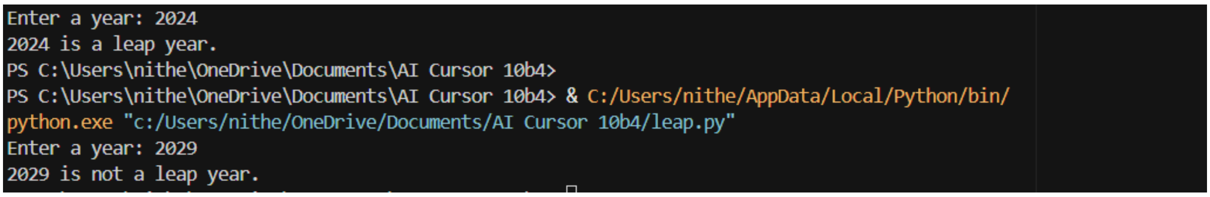
Generate a python code for checking weather the given year is leap year or not

Code:



```python
year = int(input("Enter a year: "))

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")

```

Output:



```
Enter a year: 2024
2024 is a leap year.
PS C:\Users\nithe\OneDrive\Documents\AI Cursor 10b4>
PS C:\Users\nithe\OneDrive\Documents\AI Cursor 10b4> & C:/Users/nithe/AppData/Local/Python/bin/
python.exe "c:/Users/nithe/OneDrive/Documents/AI Cursor 10b4/leap.py"
Enter a year: 2029
2029 is not a leap year.
```

Justification:

Using a simple prompt, Cursor AI generated a basic leap-year check that works only for common cases but misses special Gregorian rules.

A more detailed prompt led to a correct, reusable solution that follows all leap-year conditions.

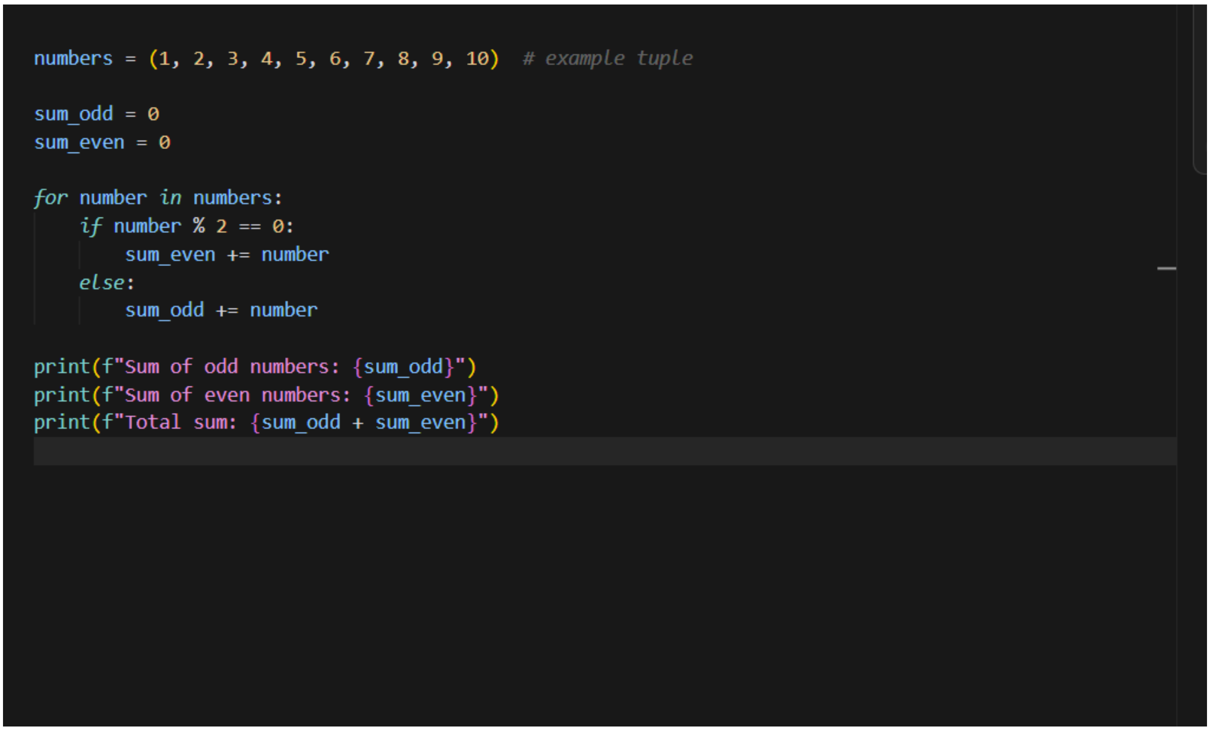
Task-4: Student Logic + AI Refactoring (Odd/Even Sum)

Without AI

#Prompt

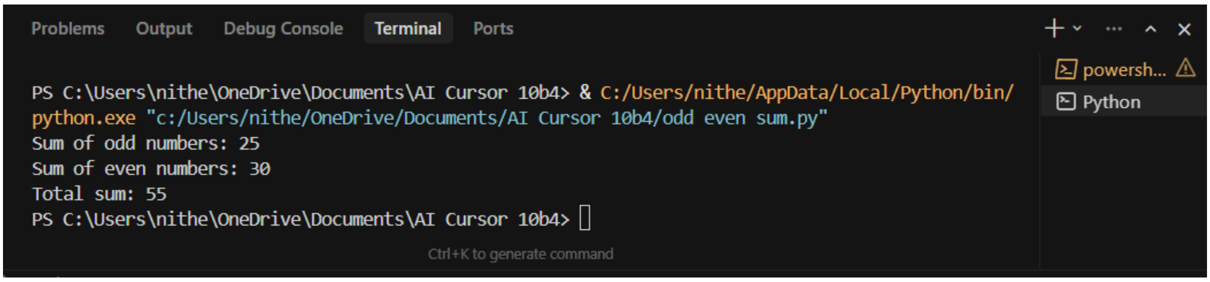
Write a Python program manually to calculate the sum of odd numbers and the sum of even numbers from a given tuple using basic control structures

Code:



```python
numbers = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  # example tuple

sum_odd = 0
sum_even = 0

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

print(f"Sum of odd numbers: {sum_odd}")
print(f"Sum of even numbers: {sum_even}")
print(f"Total sum: {sum_odd + sum_even}")
```

Output:



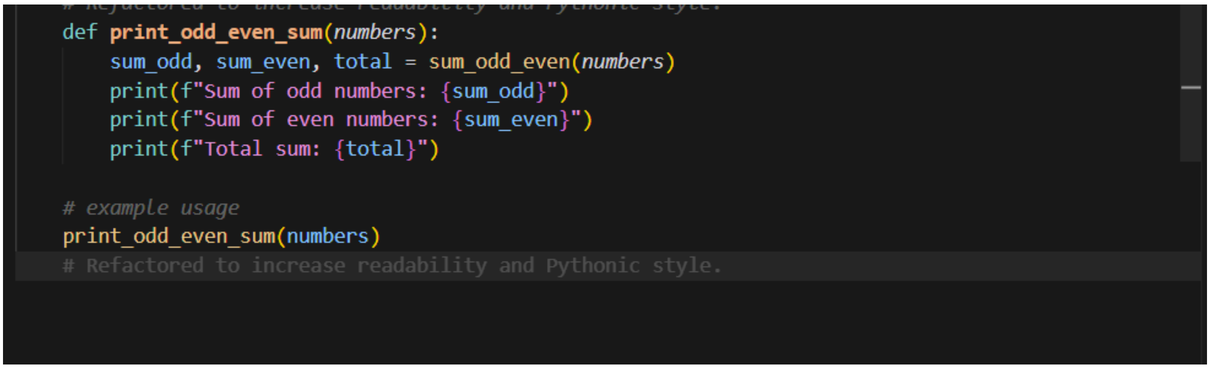
```
PS C:\Users\nithe\OneDrive\Documents\AI Cursor 10b4> & C:/Users/nithe/AppData/Local/Python/bin/
python.exe "c:/Users/nithe/OneDrive/Documents/AI Cursor 10b4/odd even sum.py"
Sum of odd numbers: 25
Sum of even numbers: 30
Total sum: 55
PS C:\Users\nithe\OneDrive\Documents\AI Cursor 10b4> 
```

AI Refactoring:

#Prompt

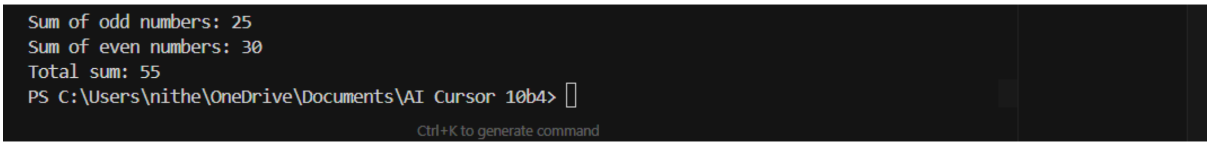
Refactor this Python code to make it more readable, reusable, and Pythonic without changing the output

Code:



```
def print_odd_even_sum(numbers):
    sum_odd, sum_even, total = sum_odd_even(numbers)
    print(f"Sum of odd numbers: {sum_odd}")
    print(f"Sum of even numbers: {sum_even}")
    print(f"Total sum: {total}")

# example usage
print_odd_even_sum(numbers)
# Refactored to increase readability and Pythonic style.
```

Output:



```
Sum of odd numbers: 25
Sum of even numbers: 30
Total sum: 55
PS C:\Users\nithe\OneDrive\Documents\AI Cursor 10b4> 
```

Justification:

Writing the code manually first ensures a clear understanding of the problem and the underlying logic.

It helps demonstrate individual problem-solving skills without relying on AI assistance.