

## LAB ASSIGNMENT-2

NAME : S.NADHIYA

ROLL NO : 2403A510C6

BRANCH : CSE

SUB : AI ASSISTED CODING

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		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-Ordinator)	
		Dr. T. Sampath Kumar	
		Dr. Pramoda Patro	
		Dr. Brij Kishor Tiwari	
		Dr. J. Ravichander	
		Dr. Mohammand Ali Shaik	
		Dr. Anirodh Kumar	
		Mr. S. Naresh Kumar	
		Dr. RAJESH VELPULA	
		Mr. Kundhan Kumar	
		Ms. Ch. Rajitha	
		Mr. M. Prakash	
		Mr. B. Raju	
		Intern 1 (Dharma teja)	
		Intern 2 (Sai Prasad)	
		Intern 3 (Sowmya)	
		NS_2 (Mounika)	
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week1 - Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	24CSBTB01 To 24CSBTB39
Assignment Number: 2.1 (Present assignment number)/24 (Total number of assignments)			

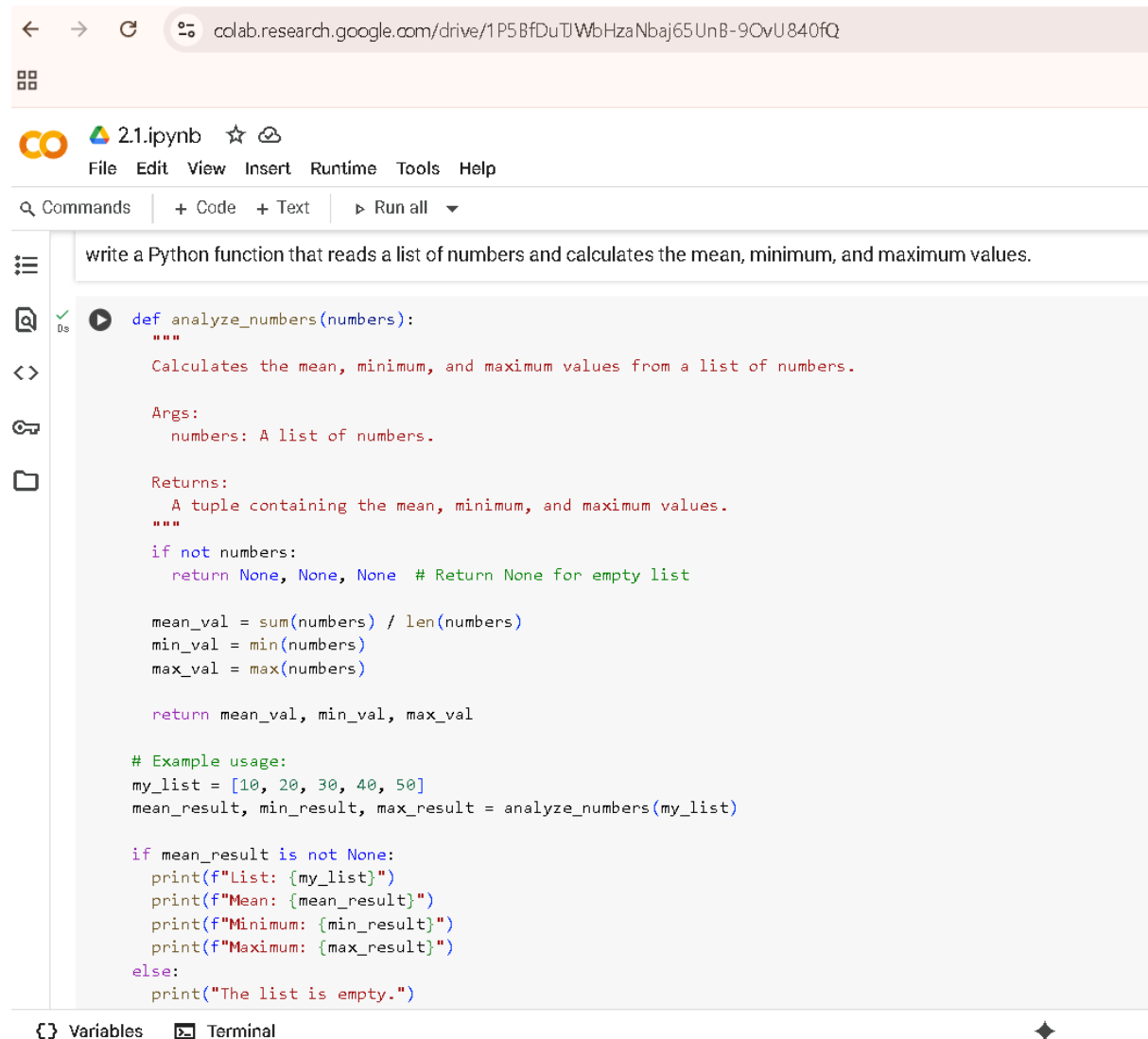
Screen shots:

Task – 01

- Use Google Gemini in Colab to write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

Expected Output #1

- Functional code with correct output and screenshot



The screenshot shows a Google Colab notebook interface. The browser address bar at the top displays the URL: `colab.research.google.com/drive/1P5BfDuTjWbHzaNbaj65UnB-9CvU840fQ`. The notebook is titled `2.1.ipynb`. The menu bar includes `File`, `Edit`, `View`, `Insert`, `Runtime`, `Tools`, and `Help`. Below the menu bar is a toolbar with `Commands`, `+ Code`, `+ Text`, and `Run all`. The main code cell contains the following Python code:

```
write a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

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

    Args:
        numbers: A list of numbers.

    Returns:
        A tuple containing the mean, minimum, and maximum values.
    """
    if not numbers:
        return None, None, None # Return None for empty list

    mean_val = sum(numbers) / len(numbers)
    min_val = min(numbers)
    max_val = max(numbers)

    return mean_val, min_val, max_val

# Example usage:
my_list = [10, 20, 30, 40, 50]
mean_result, min_result, max_result = analyze_numbers(my_list)

if mean_result is not None:
    print(f"List: {my_list}")
    print(f"Mean: {mean_result}")
    print(f"Minimum: {min_result}")
    print(f"Maximum: {max_result}")
else:
    print("The list is empty.")
```

At the bottom of the interface, there are tabs for `Variables` and `Terminal`.

```
else:
    print("The list is empty.")
```

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

## Task – 02

● Compare Gemini and Copilot outputs for a Python function that checks whether a number is an Armstrong number. Document the steps, prompts, and outputs.

Expected Output #2

● Side-by-side comparison table with observations and screenshots

## Gemini AI

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

    Args:
        number: An integer.

    Returns:
        True if the number is an Armstrong number, False otherwise.
    """
    # Convert the number to a string to find the number of digits
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0

    # Calculate the sum of the digits raised to the power of the number of digits
    for digit in num_str:
        sum_of_powers += int(digit) ** num_digits

    # Check if the sum is equal to the original number
    return sum_of_powers == number

# Example usage:
num_to_check = 153 # Example Armstrong number
if is_armstrong(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 = 10 # Example non-Armstrong number
if is_armstrong(num_to_check):
    print(f"{num_to_check} is an Armstrong number.")
else:
    print(f"{num_to_check} is not an Armstrong number.")
```

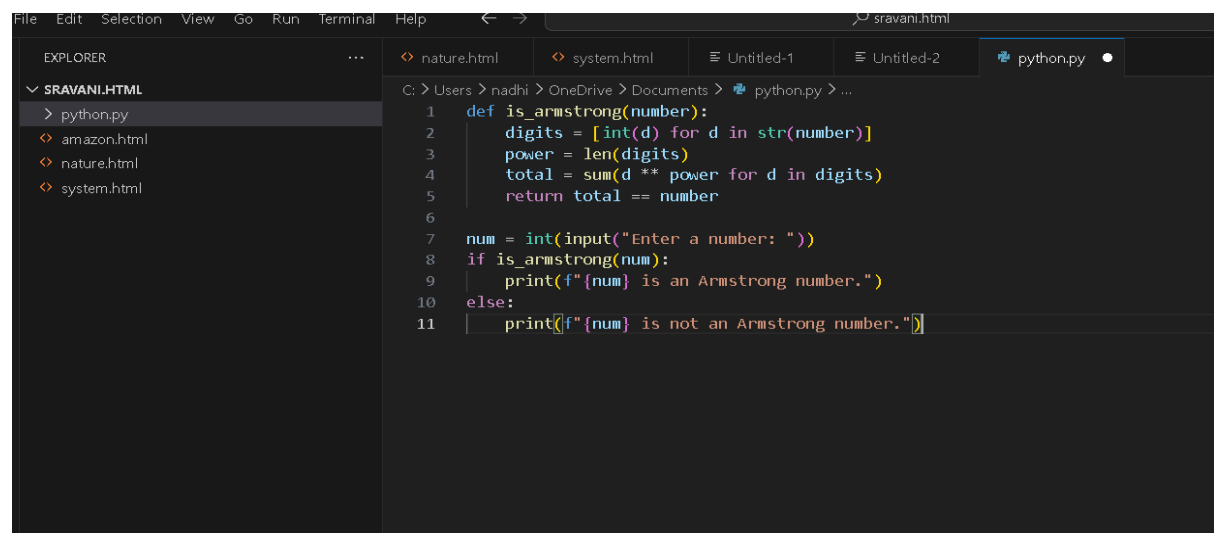


```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
# Example usage:
num_to_check = 153 # Example Armstrong number
if is_armstrong(num_to_check):
    print(f"{num_to_check} is an Armstrong number.")
else:
    print(f"{num_to_check} is not an Armstrong number.")

num_to_check = 10 # Example non-Armstrong number
if is_armstrong(num_to_check):
    print(f"{num_to_check} is an Armstrong number.")
else:
    print(f"{num_to_check} is not an Armstrong number.")

153 is an Armstrong number.
10 is not an Armstrong number.
```

## Copilot Code:



```
File Edit Selection View Go Run Terminal Help
EXPLORER
SRAVANI.HTML
python.py
amazon.html
nature.html
system.html
nature.html system.html
python.py
C:\Users\nadhi> OneDrive\Documents> python.py
1 def is_armstrong(number):
2     digits = [int(d) for d in str(number)]
3     power = len(digits)
4     total = sum(d ** power for d in digits)
5     return total == number
6
7 num = int(input("Enter a number: "))
8 if is_armstrong(num):
9     print(f"{num} is an Armstrong number.")
10 else:
11     print(f"{num} is not an Armstrong number.")
```

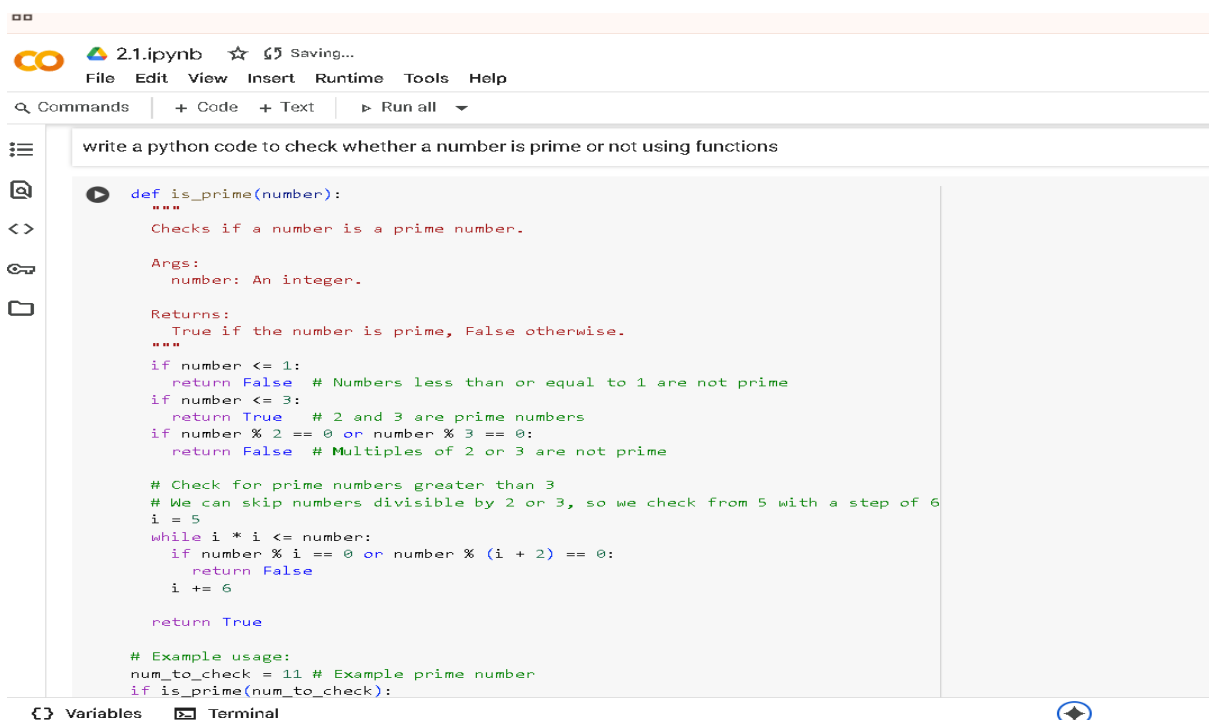
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\nadhi\sraivani.html> & C:/Users/nadhi/anaconda3/python.exe c:/Users/nadhi/OneDrive/Documents/python.py
Enter a number: 14
14 is not an Armstrong number.
```

## Task – 03

- Ask Gemini to explain a Python function (e.g., `is_prime(n)` or `is_palindrome(s)`) line by line.
- Choose either a prime-checking or palindrome-checking function and document the explanation provided by Gemini.

Expected Output #3

- Detailed explanation with the code snippet and Gemini's response



The screenshot shows a Jupyter Notebook interface with a single cell containing a Python function `is_prime`. The function checks if a number is prime by testing divisibility from 2 up to the square root of the number. The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for running, saving, and other actions, and a sidebar with icons for file management. The code is as follows:

```
def is_prime(number):
    """
    Checks if a number is a prime number.

    Args:
        number: An integer.

    Returns:
        True if the number is prime, False otherwise.
    """
    if number <= 1:
        return False # Numbers less than or equal to 1 are not prime
    if number <= 3:
        return True # 2 and 3 are prime numbers
    if number % 2 == 0 or number % 3 == 0:
        return False # Multiples of 2 or 3 are not prime

    # Check for prime numbers greater than 3
    # We can skip numbers divisible by 2 or 3, so we check from 5 with a step of 6
    i = 5
    while i * i <= number:
        if number % i == 0 or number % (i + 2) == 0:
            return False
        i += 6

    return True

# Example usage:
num_to_check = 11 # Example prime number
if is_prime(num_to_check):
```



The screenshot shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar with buttons for Commands, Code, Text, and Run all. The notebook contains a Python function `is_prime` that checks if a number is prime. The function starts with `i = 5` and increments `i` by 6 in a while loop. It returns `False` if the number is divisible by `i` or `(i + 2)`, and `True` otherwise. Below the function, there are two example usages: one for `num_to_check = 11` (prime) and one for `num_to_check = 15` (not prime). The output of the notebook shows the results of these checks: "11 is a prime number." and "15 is not a prime number."

```
# Check for prime numbers greater than 3
# We can skip numbers divisible by 2 or 3, so we check from 5 with a step of 6
def is_prime(number):
    i = 5
    while i * i <= number:
        if number % i == 0 or number % (i + 2) == 0:
            return False
        i += 6
    return True

# Example usage:
num_to_check = 11 # Example prime number
if is_prime(num_to_check):
    print(f"{num_to_check} is a prime number.")
else:
    print(f"{num_to_check} is not a prime number.")

num_to_check = 15 # Example non-prime number
if is_prime(num_to_check):
    print(f"{num_to_check} is a prime number.")
else:
    print(f"{num_to_check} is not a prime number.")
```

11 is a prime number.  
15 is not a prime number.

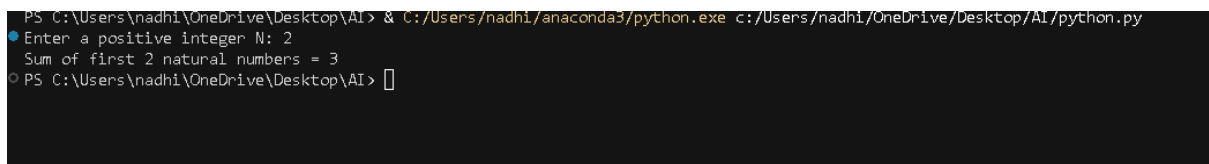
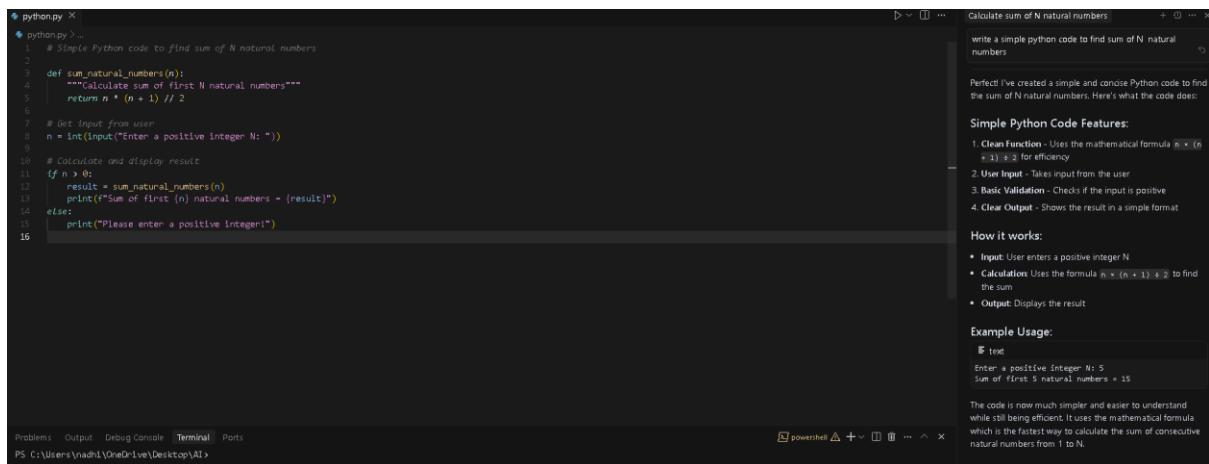
## Task – 04

- Install and configure Cursor AI. Use it to generate a Python function (e.g., sum of the first N natural numbers) and test its output.
- Optionally, compare Cursor AI's generated code with Gemini's output.

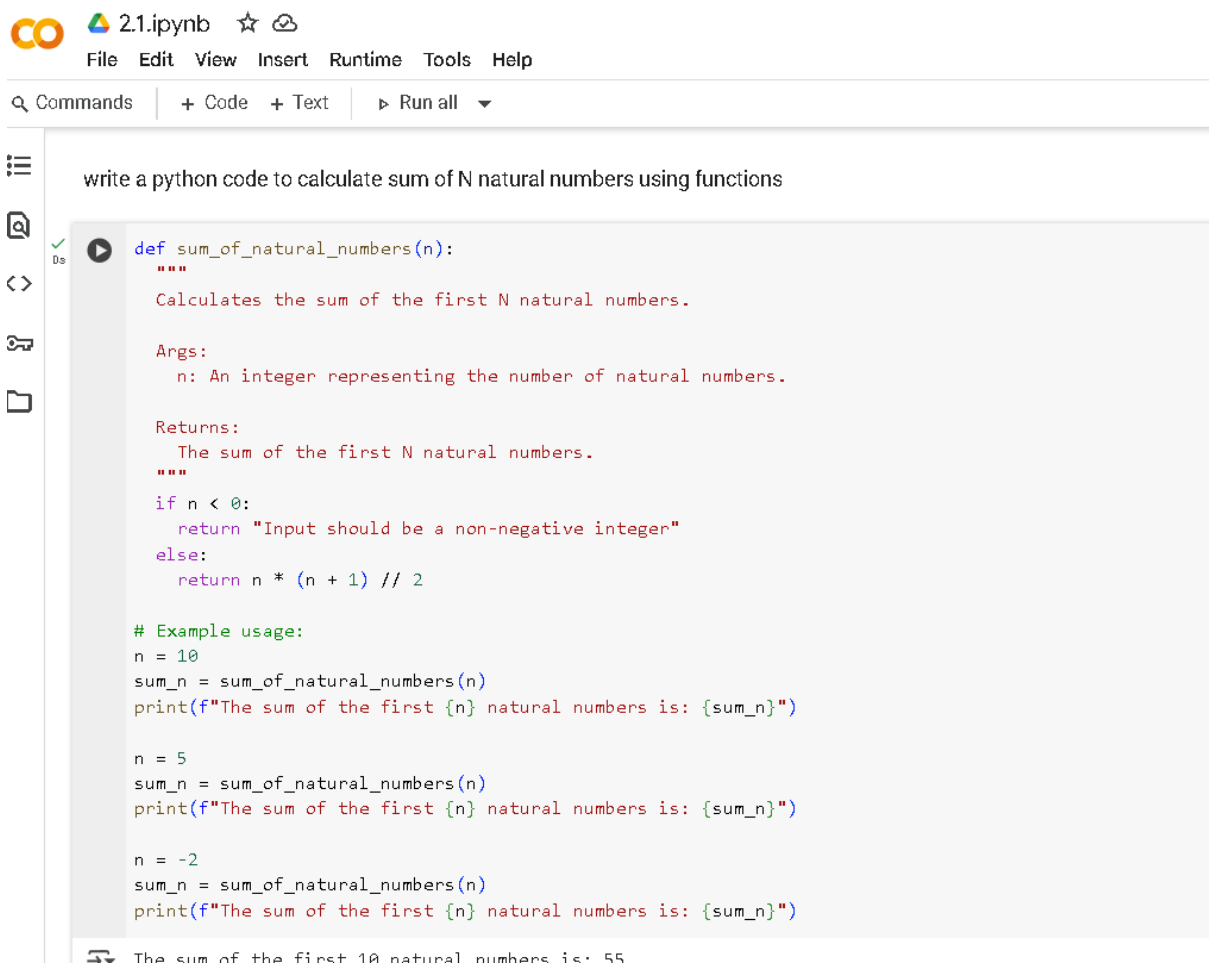
Expected Output #4

- Screenshots of Cursor AI setup, prompts used, and generated code with output

Cursor AI:



## Gemini AI:



```
print(f"The sum of the first {n} natural numbers is: {sum_n}")
```

```
➤ The sum of the first 10 natural numbers is: 55  
The sum of the first 5 natural numbers is: 15  
The sum of the first -2 natural numbers is: Input should be a non-negative integer
```

## Task – 05

- Students need to write a Python program to calculate the sum of odd numbers and even numbers in a given tuple.
- Refactor the code to improve logic and readability.

### Expected Output #5

- Student-written refactored code with explanations and output screenshots.

```
write a Python program to calculate the sum of odd numbers and even numbers in a given tuple
```

```
def sum_odd_even(numbers_tuple):  
    """  
    Calculates the sum of odd and even numbers in a tuple.  
  
    Args:  
        numbers_tuple: A tuple of numbers.  
  
    Returns:  
        A tuple containing the sum of odd numbers and the sum of even numbers.  
    """  
    sum_odd = 0  
    sum_even = 0  
    for number in numbers_tuple:  
        if number % 2 == 0:  
            sum_even += number  
        else:  
            sum_odd += number  
    return sum_odd, sum_even  
  
# Example usage:  
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
odd_sum, even_sum = sum_odd_even(my_tuple)  
  
print(f"Tuple: {my_tuple}")  
print(f"Sum of odd numbers: {odd_sum}")
```

Variables Terminal



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

⇒ Tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
Sum of odd numbers: 25  
Sum of even numbers: 30