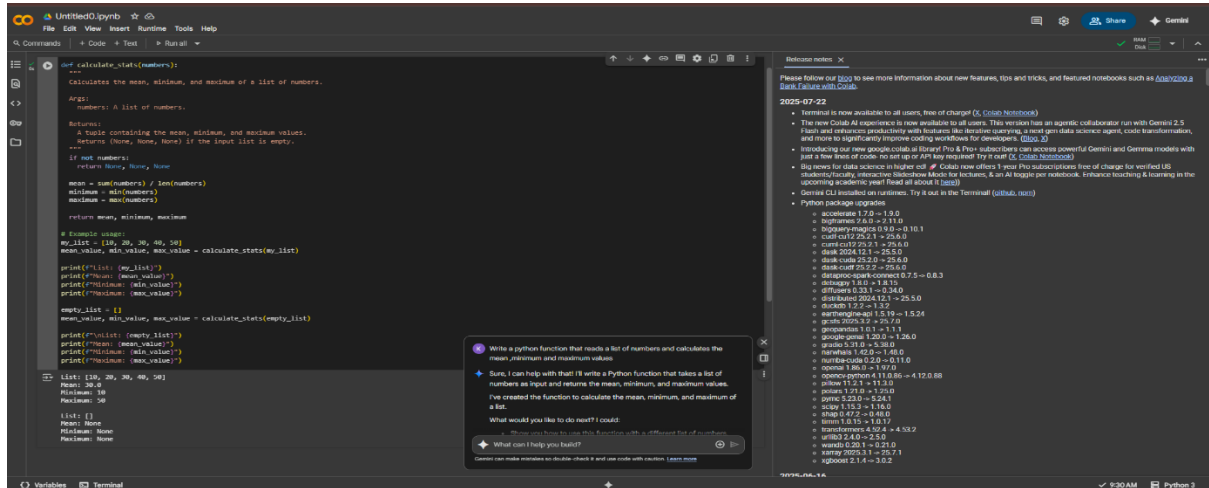


Assignment:2

Name: N.JYOSHNA SRI

2403A52106

Task:1



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

    Args:
        numbers: A list of numbers.

    Returns:
        A tuple containing the mean, minimum, and maximum values.
        returns (None, None, None) if the input list is empty.

    If not numbers:
        return None, None, None

    mean = sum(numbers) / len(numbers)
    minimum = min(numbers)
    maximum = max(numbers)

    return mean, minimum, maximum

# Example usage:
my_list = [10, 20, 30, 40, 50]
mean_value, min_value, max_value = calculate_stats(my_list)

print(f"List: {my_list}")
print(f"Mean: {mean_value}")
print(f"Minimum: {min_value}")
print(f"Maximum: {max_value}")

# Empty list:
my_list = []
mean_value, min_value, max_value = calculate_stats(empty_list)

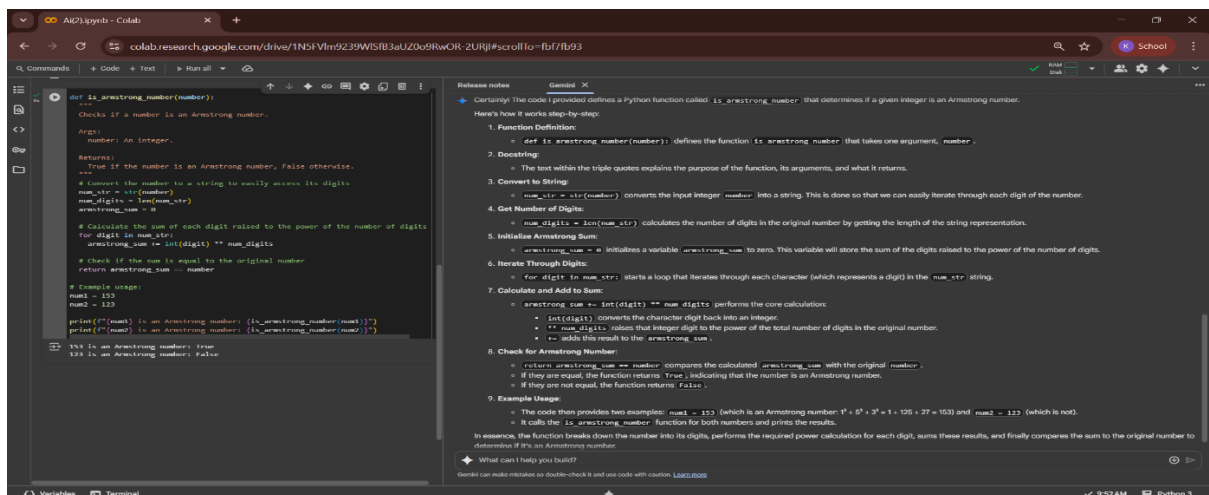
print(f"List: {empty_list}")
print(f"Mean: {mean_value}")
print(f"Minimum: {min_value}")
print(f"Maximum: {max_value}")

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

List: []
Mean: None
Minimum: None
Maximum: None
```

Task:2

Gemini



```
def is_armstrong_number(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 easily access its digits
    num_str = str(number)
    num_digits = len(num_str)
    armstrong_sum = 0

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

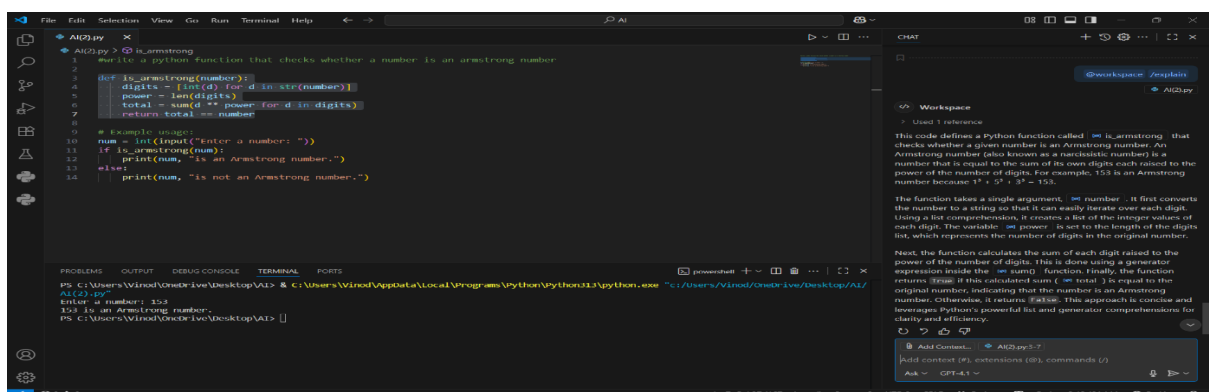
    # Check if the sum is equal to the original number
    return armstrong_sum == number

# Example usage:
num1 = 153
num2 = 123

print(f"{num1} is an Armstrong number: {is_armstrong_number(num1)}")
print(f"{num2} is an Armstrong number: {is_armstrong_number(num2)}")

This is an Armstrong number: True
123 is an Armstrong number: False
```

Copilot



```
def is_armstrong_number(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 easily access its digits
    num_str = str(number)
    num_digits = len(num_str)
    armstrong_sum = 0

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

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

# Example usage:
num1 = 153
num2 = 123

print(f"{num1} is an Armstrong number: {is_armstrong_number(num1)}")
print(f"{num2} is an Armstrong number: {is_armstrong_number(num2)}")

This is an Armstrong number: True
123 is an Armstrong number: False
```

Task:3

K Explain a python function ex: is _Palindrme

✦ I can explain what a Python function is and how it works in general.

A Python function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Here's a simple example of a Python function:

```
def greet(name):
    """This function greets the person passed in as a parameter."""
    print("Hello, " + name + ". Good morning!")

# Calling the function
greet('Alice')
```

Explanation:

- `def greet(name):` : This line defines a function named `greet` that accepts one argument, `name`.
- `"""This function greets the person passed in as a parameter."""` : This is a docstring, which explains what the function does.
- `print("Hello, " + name + ". Good morning!")` : This is the body of the function. It prints a greeting message using the `name` parameter.
- `greet('Alice')` : This line calls the `greet` function and passes the string 'Alice' as the argument for the `name` parameter.

When you run this code, the output will be:

```
Hello, Alice. Good morning!
```

This is a basic example, but functions can be much more complex, taking multiple arguments, performing calculations, and returning values. They are a fundamental part of writing efficient and organized Python code.

If you have a specific function you'd like me to explain, please provide the code for it!

👍 🗨

✦ What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

✓ 10:05 AM Python 3

Task:4

File Edit Selection View Go Run Terminal Help

AI(2).py

```
1 #Write a python function that calculates the sum of the first N Natural numbers
2
3 def sum_natural_numbers_iterative(n):
4     """
5     Calculate the sum of first N natural numbers using iteration.
6
7     Args:
8         n (int): A positive integer
9
10    Returns:
11        int: Sum of first N natural numbers (1 + 2 + 3 + ... + n)
12    """
13    if n <= 0:
14        return 0
15
16    total = 0
17    for i in range(1, n + 1):
18        total += i
19    return total
20
21
22 def sum_natural_numbers_formula(n):
23     """
24     Calculate the sum of first N natural numbers using mathematical formula.
25     Formula: sum = n * (n + 1) / 2
26
27     Args:
28         n (int): A positive integer
29
30    Returns:
31        int: Sum of first N natural numbers
32    """
33    if n <= 0:
34        return 0
35
36    return n * (n + 1) // 2
37
```

Calculate sum of first N natural numbers

I'll help you write a Python function that calculates the sum of the first N natural numbers. Let me first read the current file to see what's already there.

Read AI(2).py

I can see the file currently only contains the comment. I'll implement a Python function that calculates the sum of the first N natural numbers. There are several approaches we can use:

1. Using a loop (iterative approach)
2. Using the mathematical formula: $\text{sum} = n(n+1)/2$ (most efficient)
3. Using recursion

I'll implement all three approaches to give you options:

AI(2).py +72 ✓

Perfect! I've implemented three different Python functions to calculate the sum of the first N natural numbers:

Functions Implemented:

1. `sum_natural_numbers_iterative(n)` - Uses a simple loop to add numbers from 1 to n
2. `sum_natural_numbers_formula(n)` - Uses the mathematical formula: $n * (n + 1) / 2$ (most efficient)
3. `sum_natural_numbers_recursive(n)` - Uses recursion to calculate the sum

Key Features:

- **Input validation:** All functions handle non-positive numbers by returning 0
- **Documentation:** Each function has clear docstrings explaining parameters and return values
- **Test cases:** The code includes example usage with test cases for N = 5, 10, and 100

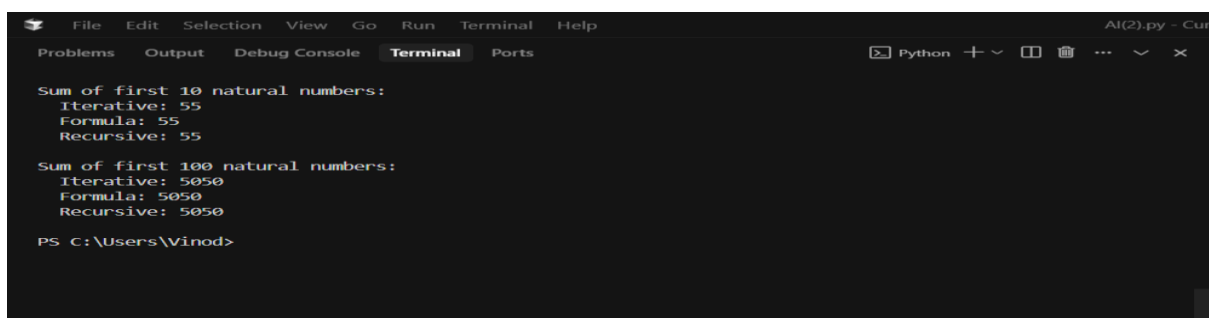
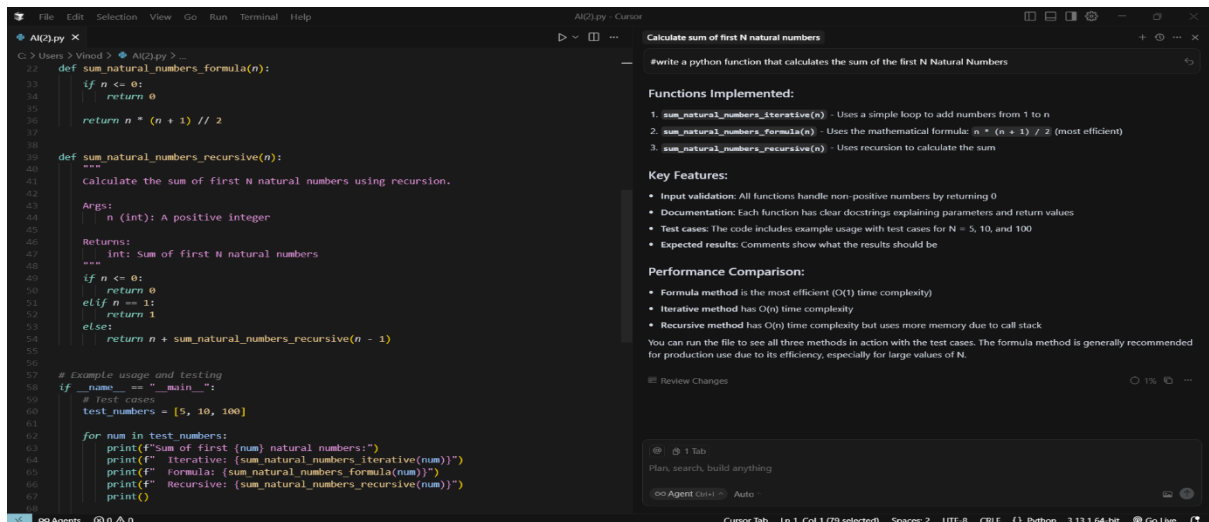
Comment on this file to share what you think about this code.

🗨 1 tab

Plan, search, build anything

Go Agent (Ctrl) Auto

Cursor Tab Ln 1, Col 1 (79 selected) Spaces: 2 UTF-8 CRLF Python 3.13.1 64-bit Go Live



Task:5

