

Lab Assignment 2.1

Name:D.BHARATH KUMAR

Hallticket:2303A51097

Batch:02

Task 1: Statistical Summary for Survey Data

Scenario:

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

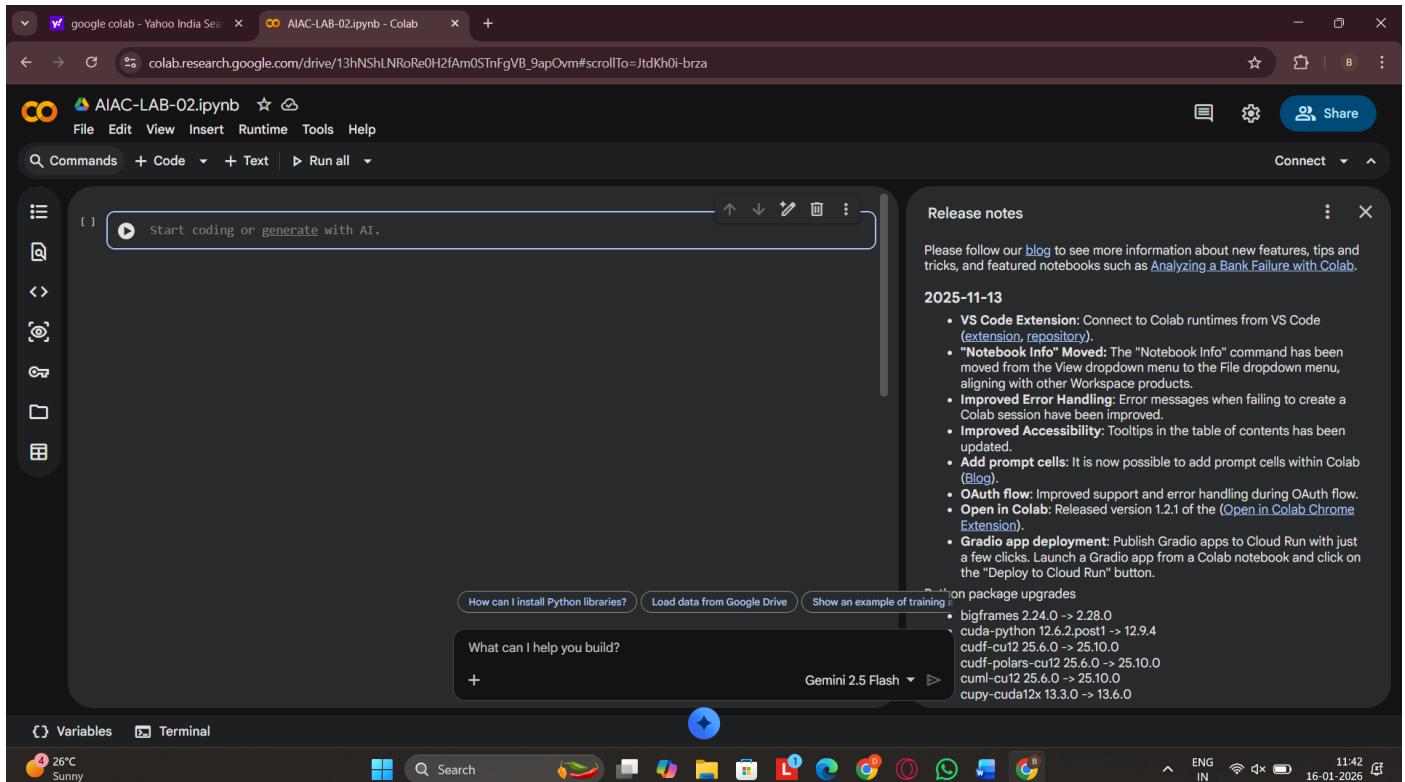
Task:

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

Expected Output:

- Correct Python function
- Output shown in Colab
- Screenshot of Gemini prompt and result

Google Colab Workspace Screenshots:



How can I install Python libraries?

Load data from Google Drive

Show an example of training a

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

+

Gemini 2.5 Flash ▾ ➤



```
'''  
    Calculates the mean, minimum, and maximum of a list of numbers.  
  
Args:  
    numbers (list): A list of numerical values.  
  
Returns:  
    tuple: A tuple containing (mean, minimum, maximum).  
    Returns (None, None, None) if the list is empty.  
'''  
if not numbers:  
    return None, None, None  
  
mean_value = sum(numbers) / len(numbers)  
min_value = min(numbers)  
max_value = max(numbers)  
  
return mean_value, min_value, max_value  
  
# Example usage:  
my_list = [10, 20, 30, 40, 50, 5, 15]  
mean, minimum, maximum = calculate_stats(my_list)  
  
print(f"List: {my_list}")  
print(f"Mean: {mean}")  
print(f"Minimum: {minimum}")  
print(f"Maximum: {maximum}")  
  
# Example with an empty list  
empty_list = []
```

B Generate a python function that reads a list of numbers and calculates the mean ,minimum, and maximum values

◆ Certainly! Here's a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values, along with an example of how to use it.

► Accept and run ✓ Accept ✕ Cancel

What can I help you build?
Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Gemini 2.5 Flash ▾ ➤

Variables Terminal

24°C Mostly cloudy

Search

ENG IN

16-01-2026

Code:

```
def calculate_stats(numbers):
```

.....

Calculates the mean, minimum, and maximum of a list of numbers.

Args:

numbers (list): A list of numerical values.

Returns:

tuple: A tuple containing (mean, minimum, maximum).

Returns (None, None, None) if the list is empty.

....

if not numbers:

 return None, None, None

 mean_value = sum(numbers) / len(numbers)

 min_value = min(numbers)

 max_value = max(numbers)

 return mean_value, min_value, max_value

Example usage:

```
my_list = [10, 20, 30, 40, 50, 5, 25]
```

```
mean, minimum, maximum = calculate_stats(my_list)
```

```
print(f"List: {my_list}")
```

```
print(f"Mean: {mean}")
```

```
print(f"Minimum: {minimum}")
```

```
print(f"Maximum: {maximum}")
```

Example with an empty list

```
empty_list = []
```

```
mean_empty, min_empty, max_empty = calculate_stats(empty_list)
```

```
print(f"\nEmpty List Stats: Mean={mean_empty}, Min={min_empty}, Max={max_empty}")
```

output:

Original List: [10, 20, 30, 40, 50, 5, 25]

Mean: 25.714285714285715

Minimum: 5

Maximum: 50

Original List: []

The list was empty.

Task 2: Armstrong Number – AI Comparison

Scenario:

- You are evaluating AI tools for numeric validation logic.

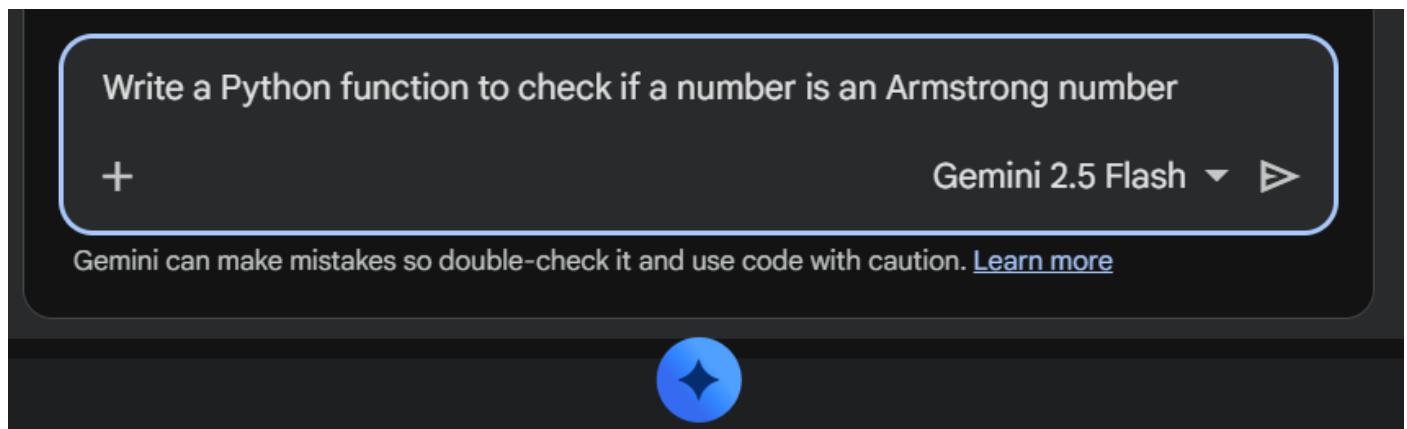
Task:

- Generate an Armstrong number checker using Gemini and GitHub Copilot.
- Compare their outputs, logic style, and clarity.

Expected Output:

- Side-by-side comparison table
- Screenshots of prompts and generated code

Google Colab Workspace Screenshots:



The screenshot shows a Google Colab notebook titled "AIAC-LAB-02.ipynb". A code cell contains Python code for checking if a number is Armstrong. A tooltip from Gemini 2.5 Flash provides a function template and explains what an Armstrong number is. The system bar at the bottom shows weather, date, and battery information.

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

    An Armstrong number (also known as a narcissistic number, plenary number, or pluperfect digital invariant) is a number that is the sum of its own digits each raised to the power of the number of digits.

    For example:
    - 153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153
    - 9474 = 9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 + 2401 + 256 = 9474

    Args:
        number (int): The integer to check.

    Returns:
        bool: True if the number is an Armstrong number.
    """
    if not isinstance(number, int) or number < 0:
        return False # Armstrong numbers are always positive

    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
```

B Write a Python function to check if a number is an Armstrong number
◆ Certainly! Here's a Python function that checks if a number is an Armstrong number. An Armstrong number is one that is equal to the sum of its own digits each raised to the power of the number of digits.
▶ Accept and run ✓ Accept ✕ Cancel

What can I help you build?
+ Gemini 2.5 Flash ▶
Gemini can make mistakes, so double-check it and use code with caution. [Learn more](#)

Code:

```
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.

For example, 371 is an Armstrong number because $3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$.

Args:

number (int): The integer to check.

Returns:

bool: True if the number is an Armstrong number, False otherwise.

Returns False for non-positive numbers.

.....

if not isinstance(number, int) or number < 0:

return False

```
if number == 0:  
    return True  
  
# 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  
temp_number = number  
  
while temp_number > 0:  
    digit = temp_number % 10  
    sum_of_powers += digit ** num_digits  
    temp_number //= 10  
  
return sum_of_powers == number  
  
print(f"Is 9 an Armstrong number? {is_armstrong_number(9)}")  
print(f"Is 10 an Armstrong number? {is_armstrong_number(10)}")  
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}")  
print(f"Is 370 an Armstrong number? {is_armstrong_number(370)}")  
print(f"Is 371 an Armstrong number? {is_armstrong_number(371)}")  
print(f"Is 407 an Armstrong number? {is_armstrong_number(407)}")  
print(f"Is 1634 an Armstrong number? {is_armstrong_number(1634)}")  
print(f"Is 1000 an Armstrong number? {is_armstrong_number(1000)}")  
print(f"Is -153 an Armstrong number? {is_armstrong_number(-153)}")  
print(f"Is 0 an Armstrong number? {is_armstrong_number(0)}")
```

output:

Is 9 an Armstrong number? True

Is 10 an Armstrong number? False

Is 153 an Armstrong number? True

Is 370 an Armstrong number? True

Is 371 an Armstrong number? True

Is 407 an Armstrong number? True

Is 1634 an Armstrong number? True

Is 1000 an Armstrong number? False

Is -153 an Armstrong number? False

Is 0 an Armstrong number? True

Vs code :

```
C:\Users\arell\Music\aiac\lab_1.py
1 #Write a Python function to check if a number is an Armstrong number
→| def is_armstrong_number(n):
    # Convert the number to a string to easily iterate over digits
    num_str = str(n)
    # Calculate the number of digits
    num_digits = len(num_str)
    # Calculate the sum of each digit raised to the power of the number of digits
    total = sum(int(digit) ** num_digits for digit in num_str)
    # Check if the total equals the original number
    return total == n
```

```
assg_02.py > ...
1 #Write a Python function to check if a number is an Armstrong number
2 def is_armstrong(number):
3     num_str = str(number) # Convert the number to string to easily iterate over digits
4     num_digits = len(num_str) # Get the number of digits
5     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str) # Calculate the sum of each digit raised to the power of num_digits
6     return sum_of_powers == number # Check if the sum of powers is equal to the original number
7 # Take input from the user
8 num = int(input("Enter a number to check if it is an Armstrong number: "))
9 if is_armstrong(num):
10     print(f"{num} is an Armstrong number.")
11 else:
12     print(f"{num} is not an Armstrong number.")
```

Code:

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

```
def is_armstrong(number):
```

```
    num_str = str(number) # Convert the number to string to easily iterate over digits
```

```
    num_digits = len(num_str) # Get the number of digits
```

```
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str) # Calculate the sum of each digit raised to the power of num_digits
```

```

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

# Take input from the user

num = int(input("Enter a number to check if it is an Armstrong number: "))

if is_armstrong(num):

    print(f"{num} is an Armstrong number.")

else:

    print(f"{num} is not an Armstrong number.")

```

output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_02.py
Enter a number to check if it is an Armstrong number: 121
121 is not an Armstrong number.

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_02.py
Enter a number to check if it is an Armstrong number: 153
153 is an Armstrong number.

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

Task 3: Leap Year Validation Using Cursor AI

Scenario:

- You are validating a calendar module for a backend system.

Task:

- Use Cursor AI to generate a Python program that checks whether a given year is a leap year.
- Use at least two different prompts and observe changes in code.

Expected Output:

- Two versions of code
- Sample inputs/outputs
- Brief comparison

Version - 1:

Instruction Given:

Write a Python function to check if a year is a leap year and include test cases.

File Edit Selection View Go Run Terminal Help

aiac

asg_02.py

```

1  """#Write a Python function to check if a number is an Armstrong number
2  def is_armstrong(number):
3      num_str = str(number) # Convert the number to string to easily iterate
4      num_digits = len(num_str) # Get the number of digits
5      sum_of_powers = sum(int(digit) ** num_digits for digit in num_str) # Calculate the sum of powers
6      return sum_of_powers == number # Check if the sum of powers is equal to the number
7  # Take input from the user
8  num = int(input("Enter a number to check if it is an Armstrong number: "))
9  if is_armstrong(num):
10     print(f"{num} is an Armstrong number.")
11 else:
12     print(f"{num} is not an Armstrong number.")""

```

Python leap year function with tests

Now - Auto

Write a Python function to check if a year is a leap year and include test cases.

Planning next moves

Add a follow-up

oo Agent Auto

New update available Later Install Now

Cursor Tab In 12, Col 51 Spaces: 4 UTF-8 CRLF Python

File Edit Selection View Go Run Terminal Help

aiac

asg_02.py

```

1  """#Write a Python function to check if a number is an Armstrong number
2  def is_armstrong(number):
3      num_str = str(number) # Convert the number to string to easily iterate
4      num_digits = len(num_str) # Get the number of digits
5      sum_of_powers = sum(int(digit) ** num_digits for digit in num_str) # Calculate the sum of powers
6      return sum_of_powers == number # Check if the sum of powers is equal to the number
7  # Take input from the user
8  num = int(input("Enter a number to check if it is an Armstrong number: "))
9  if is_armstrong(num):
10     print(f"{num} is an Armstrong number.")
11 else:
12     print(f"{num} is not an Armstrong number.")""

```

Write a Python function to check if a year is a leap year

```

15  def is_leap_year(year):
16      """
17          Check if a year is a leap year.
18
19          A year is a leap year if:
20          - It is divisible by 4, AND
21          - It is NOT divisible by 100, OR
22          - It is divisible by 400
23
24          Args:
25              year (int): The year to check
26
27          Returns:
28              bool: True if the year is a leap year, False otherwise
29      """
30      if year % 400 == 0:
31          return True
32      elif year % 100 == 0:
33          return False
34      elif year % 4 == 0:
35          return True
36      else:
37          return False

```

Test cases

```

41  if __name__ == "__main__":
42      # Test case 1: Year divisible by 400 (Leap year)
43      assert is_leap_year(2000) == True, "2000 should be a leap year"
44      print("✓ Test 1 passed: 2000 is a leap year")
45
46      # Test case 2: Year divisible by 100 but not 400 (not Leap year)
47      assert is_leap_year(1900) == False, "1900 should not be a leap year"
48      print("✗ Test 2 failed: 1900 is a leap year")

```

Python leap year function with tests

Now +63 - Auto

Write a Python function to check if a year is a leap year and include test cases.

Checking the current file for context.

Read asg_02.py

Adding a leap year function with test cases to asg_02.py.

Read asg_02.py

Adding a leap year function with test cases to the file:

asg_02.py +63

```

print(f"{num} is not an Armstrong number.")""

```

write a Python function to check if a year is a leap year

Checking for linting issues and running the tests:

No linter errors found in asg_02.py

Run command: cd python

\$ cd c:\Users\arell\Music\aiac && python asg_02.py

Use Allowlist Skip Allowlist 'cd' + 1 gel Run

Waiting for Approval.

Stop Ctrl+Shift+Q Review

Reject, suggest, follow up?

oo Agent Auto

New update available Later Install Now

Cursor Tab In 12, Col 40 Spaces: 4 UTF-8 CRLF Python

Code:

Write a Python function to check if a year is a leap year

```
def is_leap_year(year):
```

.....

Check if a year is a leap year.

A year is a leap year if:

- It is divisible by 4, AND
- It is NOT divisible by 100, OR
- It is divisible by 400

Args:

year (int): The year to check

Returns:

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

.....

if year % 400 == 0:

return True

elif year % 100 == 0:

return False

elif year % 4 == 0:

return True

else:

return False

Test cases

if __name__ == "__main__":

Test case 1: Year divisible by 400 (leap year)

assert is_leap_year(2000) == True, "2000 should be a leap year"

print("✓ Test 1 passed: 2000 is a leap year")

Test case 2: Year divisible by 100 but not 400 (not leap year)

assert is_leap_year(1900) == False, "1900 should not be a leap year"

print("✓ Test 2 passed: 1900 is not a leap year")

```
# Test case 3: Year divisible by 4 but not 100 (leap year)
assert is_leap_year(2020) == True, "2020 should be a leap year"
print("✓ Test 3 passed: 2020 is a leap year")

# Test case 4: Year divisible by 4 but not 100 (leap year)
assert is_leap_year(2024) == True, "2024 should be a leap year"
print("✓ Test 4 passed: 2024 is a leap year")

# Test case 5: Year not divisible by 4 (not leap year)
assert is_leap_year(2021) == False, "2021 should not be a leap year"
print("✓ Test 5 passed: 2021 is not a leap year")

# Test case 6: Year not divisible by 4 (not leap year)
assert is_leap_year(2023) == False, "2023 should not be a leap year"
print("✓ Test 6 passed: 2023 is not a leap year")

# Test case 7: Year divisible by 400 (leap year)
assert is_leap_year(1600) == True, "1600 should be a leap year"
print("✓ Test 7 passed: 1600 is a leap year")

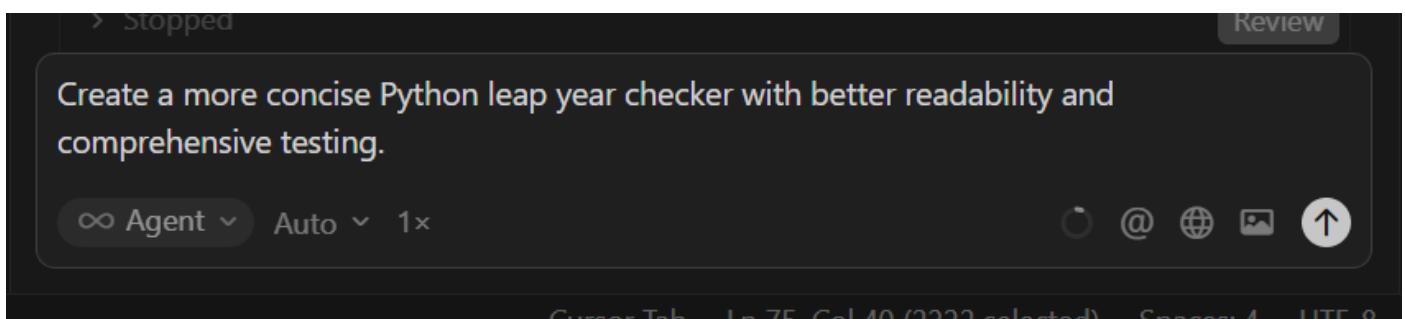
# Test case 8: Year divisible by 100 but not 400 (not leap year)
assert is_leap_year(1800) == False, "1800 should not be a leap year"
print("✓ Test 8 passed: 1800 is not a leap year")

print("\nAll test cases passed! ✓")
```

```
15 |     assert is_leap_year(2021) == False, "2021 should not be a leap year"
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ▾
servers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_02.py
✓ Test 1 passed: 2000 is a leap year
✓ Test 2 passed: 1900 is not a leap year
✓ Test 3 passed: 2020 is a leap year
✓ Test 4 passed: 2024 is a leap year
✓ Test 5 passed: 2021 is not a leap year
✓ Test 6 passed: 2023 is not a leap year
✓ Test 7 passed: 1600 is a leap year
✓ Test 8 passed: 1800 is not a leap year

All test cases passed! ✓
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Version—02



Code":

```
# Write a Python function to check if a year is a leap year
```

```
def is_leap_year(year):
```

```
    """
```

Check if a year is a leap year.

A year is a leap year if it is divisible by 4, except when it's divisible

by 100 unless it's also divisible by 400.

Args:

year (int): The year to check

Returns:

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

```
    """
```

```
return (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0)
```

```
# Comprehensive test cases
```

```
if __name__ == "__main__":
```

```
# Test cases organized by category
```

```
test_cases = [
```

```
# Centuries divisible by 400 (leap years)
```

```
(2000, True, "Century divisible by 400"),
```

```
(1600, True, "Century divisible by 400"),
```

```
(2400, True, "Century divisible by 400"),
```

```
# Centuries not divisible by 400 (not leap years)
```

```
(1900, False, "Century not divisible by 400"),
```

```
(1800, False, "Century not divisible by 400"),
```

```
(2100, False, "Century not divisible by 400"),
```

```
# Regular years divisible by 4 (leap years)
```

```
(2020, True, "Year divisible by 4"),
```

```
(2024, True, "Year divisible by 4"),
```

```
(2016, True, "Year divisible by 4"),
```

```
(2004, True, "Year divisible by 4"),
```

```
# Regular years not divisible by 4 (not leap years)
```

```
(2021, False, "Year not divisible by 4"),
```

```
(2023, False, "Year not divisible by 4"),
```

```
(2019, False, "Year not divisible by 4"),
```

```
(2022, False, "Year not divisible by 4"),
```

```
# Edge cases
```

```
(1, False, "Year 1"),
```

```
(4, True, "Year 4 (first leap year)'),
```

```
(100, False, "Year 100"),  
(400, True, "Year 400"),  
]  
  
# Run all tests
```

```
passed = 0  
failed = 0
```

for year, expected, description in test_cases:

```
    result = is_leap_year(year)  
  
    status = "✓" if result == expected else "✗"  
  
    if result == expected:  
        passed += 1  
  
        print(f"{status} {description}: {year} -> {result}")  
  
    else:  
        failed += 1  
  
        print(f"{status} {description}: {year} -> {result} (expected {expected})")
```

Summary

```
print("\n'*50")  
  
print(f"Tests passed: {passed}/{len(test_cases)}")  
  
if failed > 0:  
    print(f"Tests failed: {failed}/{len(test_cases)}")  
  
else:  
    print("All tests passed! ✓")
```

output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_02.py
✓ Century divisible by 400: 2000 -> True
✓ Century divisible by 400: 1600 -> True
✓ Century divisible by 400: 2400 -> True
✓ Century not divisible by 400: 1900 -> False
✓ Century not divisible by 400: 1800 -> False
✓ Century not divisible by 400: 2100 -> False
✓ Year divisible by 4: 2020 -> True
✓ Year divisible by 4: 2024 -> True
✓ Year divisible by 4: 2016 -> True
✓ Year divisible by 4: 2004 -> True
✓ Year not divisible by 4: 2021 -> False
✓ Year not divisible by 4: 2023 -> False
✓ Year not divisible by 4: 2019 -> False
✓ Year not divisible by 4: 2022 -> False
✓ Year 1: 1 -> False
✓ Year 4 (first leap year): 4 -> True
✓ Year 100: 100 -> False
✓ Year 400: 400 -> True

=====
Tests passed: 18/18
All tests passed! ✓
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

Brief comparison:

Aspect	Version 1 (Basic)	Version 2 (Concise)
Logic	Nested if-else	Single return with logical operators
Lines of code	More (≈ 12)	Fewer (≈ 5)
Readability	Step-by-step	Compact
Beginner friendly	Yes	No
Style	Traditional	Pythonic
Conditions	Separate checks	Combined logic
Testing	Basic tests	Comprehensive tests

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

Scenario:

- Company policy requires developers to write logic before using AI.

Task:

- Write a Python program that calculates the sum of odd and even numbers in a tuple, then refactor it using any AI tool.

Expected Output:

- Original code

- Refactored code
- Explanation of improvements

Original Code written by me:

```

▽ def even_odd_sum_tuple(num):
    even_sum = 0
    odd_sum = 0
    ▽ for i in range(1, num+1):
        ▽ if i%2 == 0:
            even_sum += i
        ▽ else:
            odd_sum += i
    return even_sum, odd_sum
user_input = int(input("Enter a number: "))
even, odd = even_odd_sum_tuple(user_input)
print(f"Sum of even numbers: {even}")
print(f"Sum of odd numbers: {odd}")

```

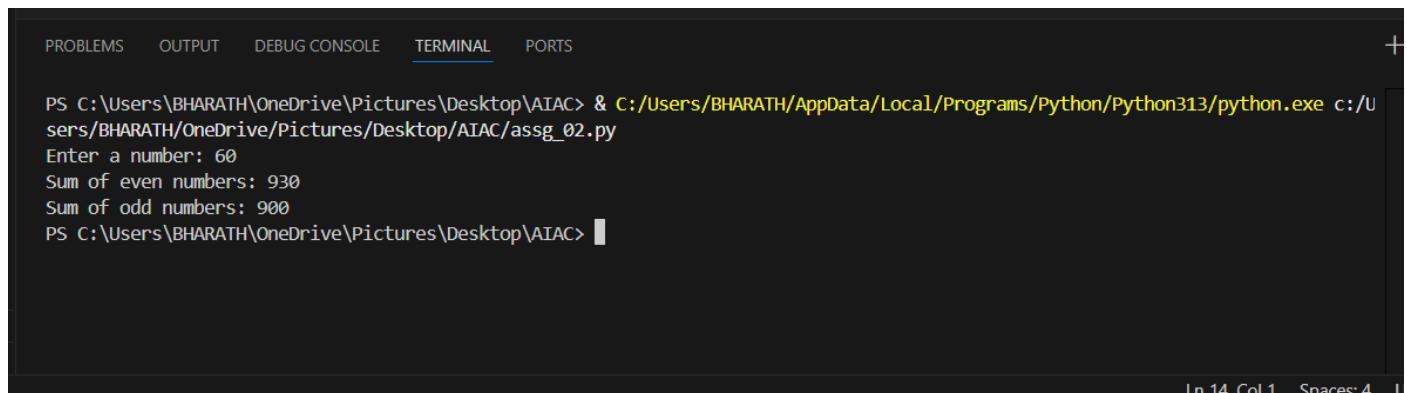
Code:

```

def even_odd_sum_tuple(num):
    even_sum = 0
    odd_sum = 0
    for i in range(1, num+1):
        if i%2 == 0:
            even_sum += i
        else:
            odd_sum += i
    return even_sum, odd_sum
user_input = int(input("Enter a number: "))
even, odd = even_odd_sum_tuple(user_input)
print(f"Sum of even numbers: {even}")
print(f"Sum of odd numbers: {odd}")

```

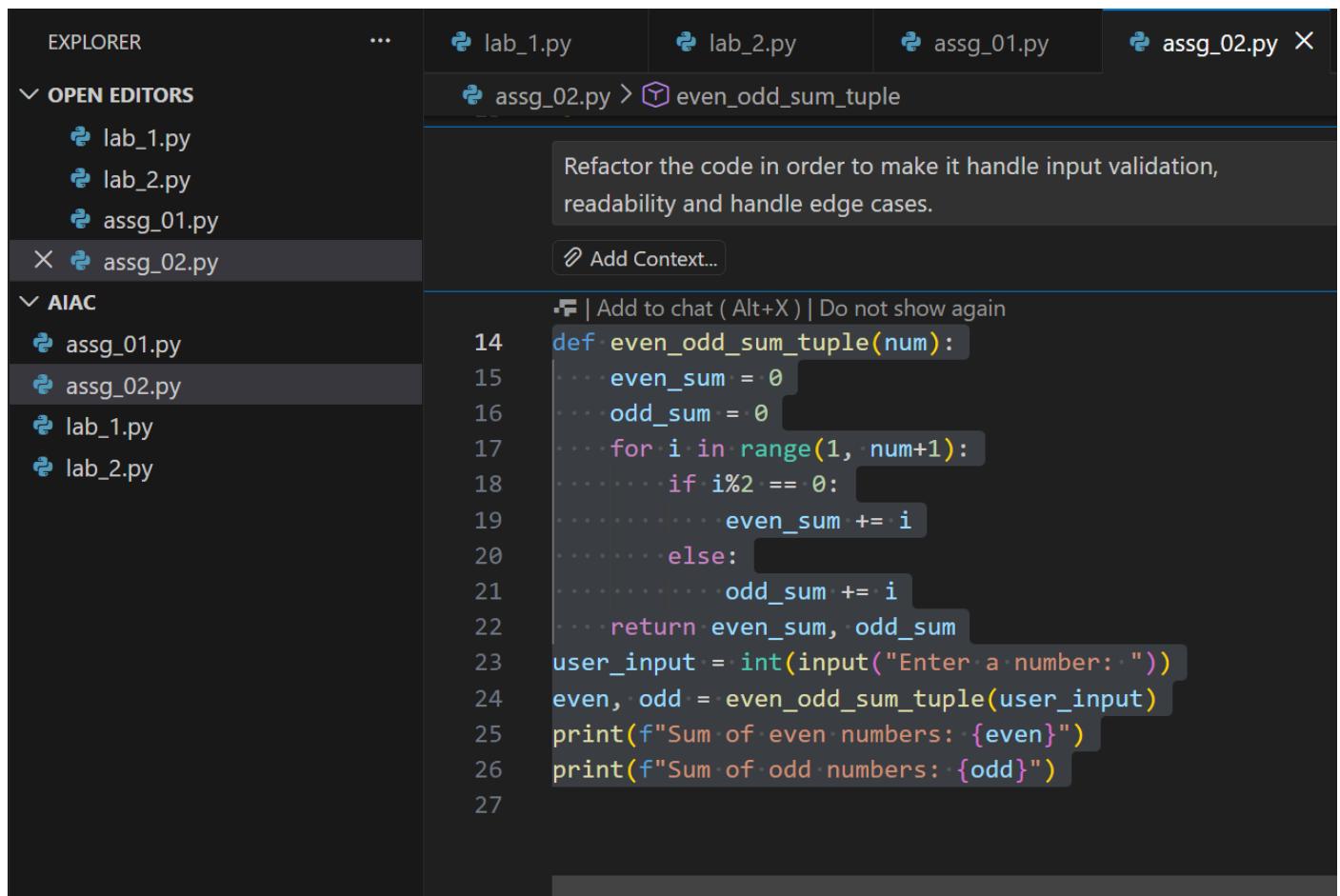
Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS +  
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & c:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_02.py  
Enter a number: 60  
Sum of even numbers: 930  
Sum of odd numbers: 900  
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Ln 14, Col 1 Spaces: 4

Refactoring the code :



EXPLORER ... lab_1.py lab_2.py assg_01.py assg_02.py X
OPEN EDITORS assg_02.py > even_odd_sum_tuple
Assg_02.py
Refactor the code in order to make it handle input validation, readability and handle edge cases.
Add Context...
14 | Add to chat (Alt+X) | Do not show again
14 def even_odd_sum_tuple(num):
15 even_sum = 0
16 odd_sum = 0
17 for i in range(1, num+1):
18 if i%2 == 0:
19 even_sum += i
20 else:
21 odd_sum += i
22 return even_sum, odd_sum
23 user_input = int(input("Enter a number: "))
24 even, odd = even_odd_sum_tuple(user_input)
25 print(f"Sum of even numbers: {even}")
26 print(f"Sum of odd numbers: {odd}")
27

The screenshot shows a code editor interface with multiple tabs at the top: lab_1.py, lab_2.py, assg_01.py, assg_02.py (active), and ... Below the tabs is a toolbar with icons for Modify selected code, Add Context..., and Auto. The main area displays the following Python code:

```
14 ✓ def even_odd_sum_tuple(num):
    even_sum = 0
    odd_sum = 0
    for i in range(1, num+1):
        if i%2 == 0:
            even_sum += i
        else:
            odd_sum += i
15 ✘ """Calculate sum of even and odd numbers from 1 to num.
16 ...
17     Args:
18         num: Positive integer
19 ...
20     Returns:
21         Tuple of (even_sum, odd_sum)
22 ...
23 ✘     if num < 1:
24         return 0, 0
```

The code is annotated with several comments and docstrings. Lines 15, 16, 17, 18, 19, 20, 21, 22, and 23 have red highlights. Line 15 contains a multi-line docstring. Lines 17 through 22 are part of a docstring for the function's parameters and return value. Line 23 has a red highlight under the if condition. The status bar at the bottom right shows 'Keep Undo'.

```
assg_02.py > ...
    print(f"\n{num} is NOT an Armstrong number. : )")
13
14 def even_odd_sum_tuple(num):
15     """Calculate sum of even and odd numbers from 1 to num."""
16
17     Args:
18         num: Positive integer
19
20     Returns:
21         Tuple of (even_sum, odd_sum)
22     """
23
24     if num < 1:
25         return 0, 0
26
27     even_sum = sum(i for i in range(2, num + 1, 2))
28     odd_sum = sum(i for i in range(1, num + 1, 2))
29
30     return even_sum, odd_sum
31
32 def get_positive_integer(prompt):
33     """Get valid positive integer input from user."""
34     while True:
35         try:
36             value = int(input(prompt))
37             if value < 1:
38                 print("Please enter a positive number.")
39             continue
40             return value
41         except ValueError:
42             print("Invalid input. Please enter a valid integer.")
43
44 if __name__ == "__main__":
45     user_input = get_positive_integer("Enter a positive number: ")
46     even, odd = even_odd_sum_tuple(user_input)
47     print(f"Sum of even numbers: {even}")
48     print(f"Sum of odd numbers: {odd}")
49
| Add to chat (Alt+X) | Do not show again
```

Code

```
def even_odd_sum_tuple(num):
    """Calculate sum of even and odd numbers from 1 to num.
```

Args:

num: Positive integer

Returns:

Tuple of (even_sum, odd_sum)

....

if num < 1:

 return 0, 0

even_sum = sum(i for i in range(2, num + 1, 2))

odd_sum = sum(i for i in range(1, num + 1, 2))

return even_sum, odd_sum

def get_positive_integer(prompt):

 """Get valid positive integer input from user."""

 while True:

 try:

 value = int(input(prompt))

 if value < 1:

 print("Please enter a positive number.")

 continue

 return value

 except ValueError:

 print("Invalid input. Please enter a valid integer.")

if __name__ == "__main__":

 user_input = get_positive_integer("Enter a positive number: ")

 even, odd = even_odd_sum_tuple(user_input)

 print(f"Sum of even numbers: {even}")

 print(f"Sum of odd numbers: {odd}")

output:

A screenshot of a terminal window from a code editor. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. The terminal output shows:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_02.py
Enter a positive number: 60
Sum of even numbers: 930
Sum of odd numbers: 900
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Explanation of improvements:

Input validation

- Making sure the user enters a valid integer.
- Handling negative numbers gracefully (return (0,0) or raise an error).

Readability

- Using clear variable names.
- Adding docstrings and comments.

Edge cases

- If input is 0, both sums should be 0.
- If input is negative, we can either reject it or compute sums up to that number (here I'll reject it for clarity).