

AI Assisted Coding

LAB ASSIGNMENT -02

Name: G.Abhiram

Hallticket:2303A51087

Batch:02

Task 1: Statistical Summary for Survey

Data Scenario:

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

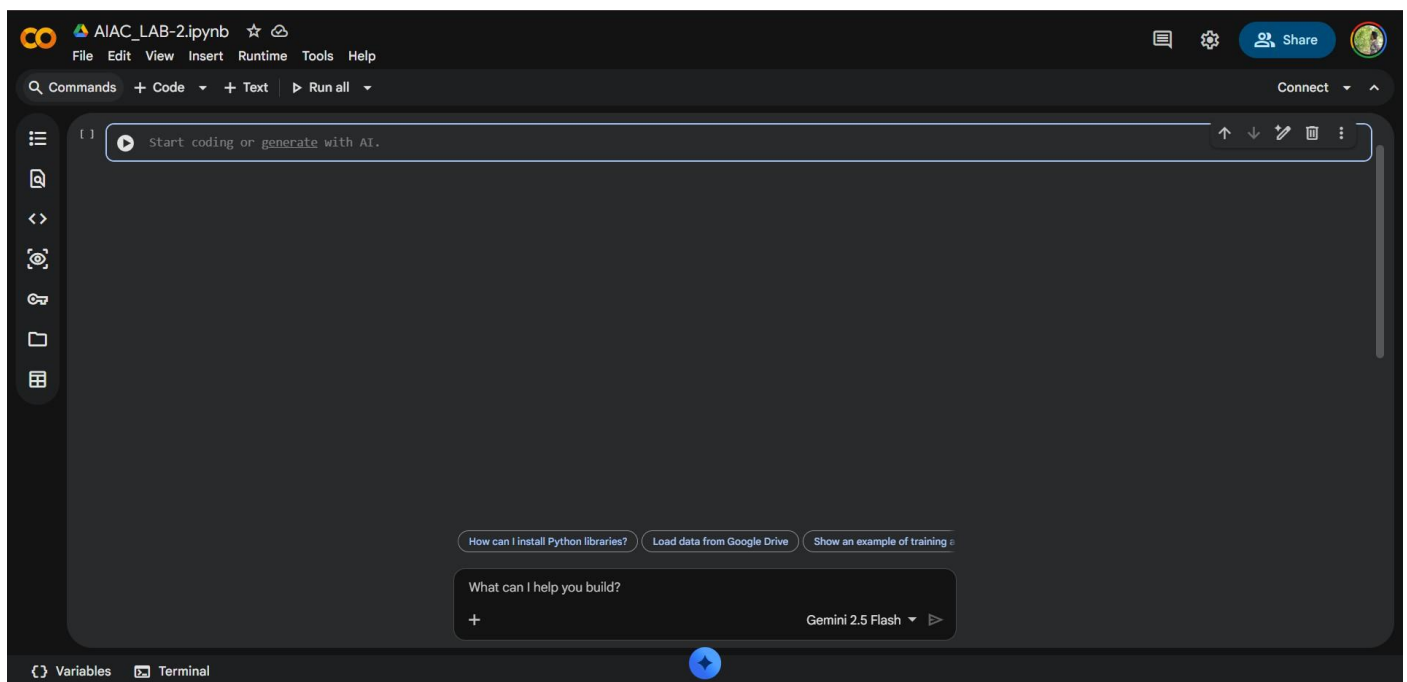
Task:

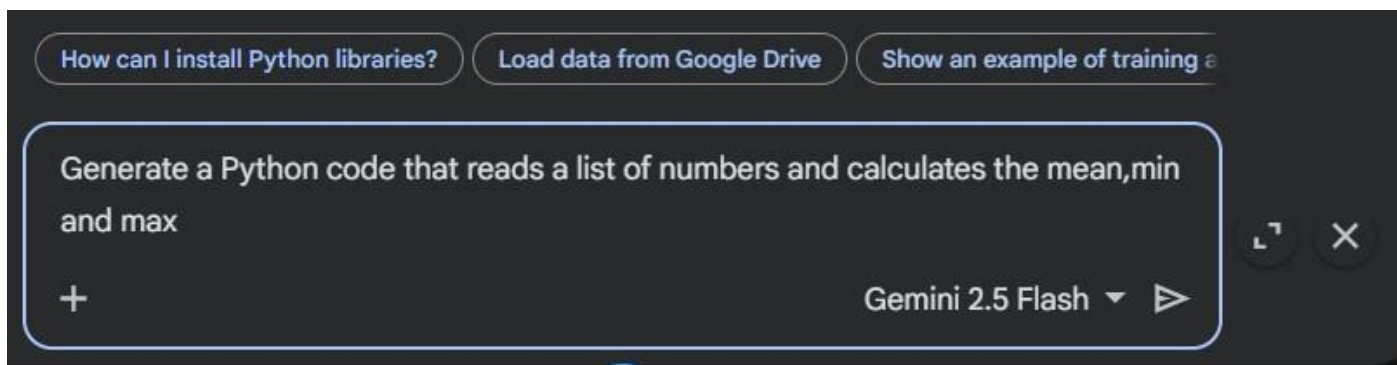
- Use Google Gemini in Colab to generate a Python function that reads a list of numbers and calculates the mean, minimum, and maximum values.

Expected Output:

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

Google Colab Workspace Screenshots:





```
AIAC_LAB-2.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
RAM Disk

import statistics
def analyze_numbers(numbers):
    """
    Calculates the mean, minimum, and maximum of 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 = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

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

data = [10, 20, 30, 40, 50, 5, 25]
analysis_results = analyze_numbers(data)
```

Code:

```
import statistics
```

```
def analyze_numbers(numbers):
```

```
    """
```

```
    Calculates the mean, minimum, and maximum of 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 = statistics.mean(numbers)
```

```
min_value = min(numbers)
```

```
max_value = max(numbers)
```

```
return {
```

```
    'mean': mean_value,
```

```
    'min': min_value,
```

```
    'max': max_value
```

```
}
```

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

```
analysis_results = analyze_numbers(data)
```

```
if analysis_results:
```

```
    print(f"Original List: {data}")
```

```
    print(f"Mean: {analysis_results['mean']}")
```

```
    print(f"Minimum: {analysis_results['min']}")
```

```
    print(f"Maximum: {analysis_results['max']}")
```

```
else:
```

```
    print("The list was empty.")
```

```
empty_data = []
```

```
empty_results = analyze_numbers(empty_data)
```

```
if empty_results:
```

```
    print(f"Original List: {empty_data}")
```

```
    print(f"Mean: {empty_results['mean']}")
```

```
    print(f"Minimum: {empty_results['min']}")
```

```
print(f"Maximum: {empty_results['max']}")
```

else:

```
print(f"Original List: {empty_data}")
```

```
print("The list was 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.

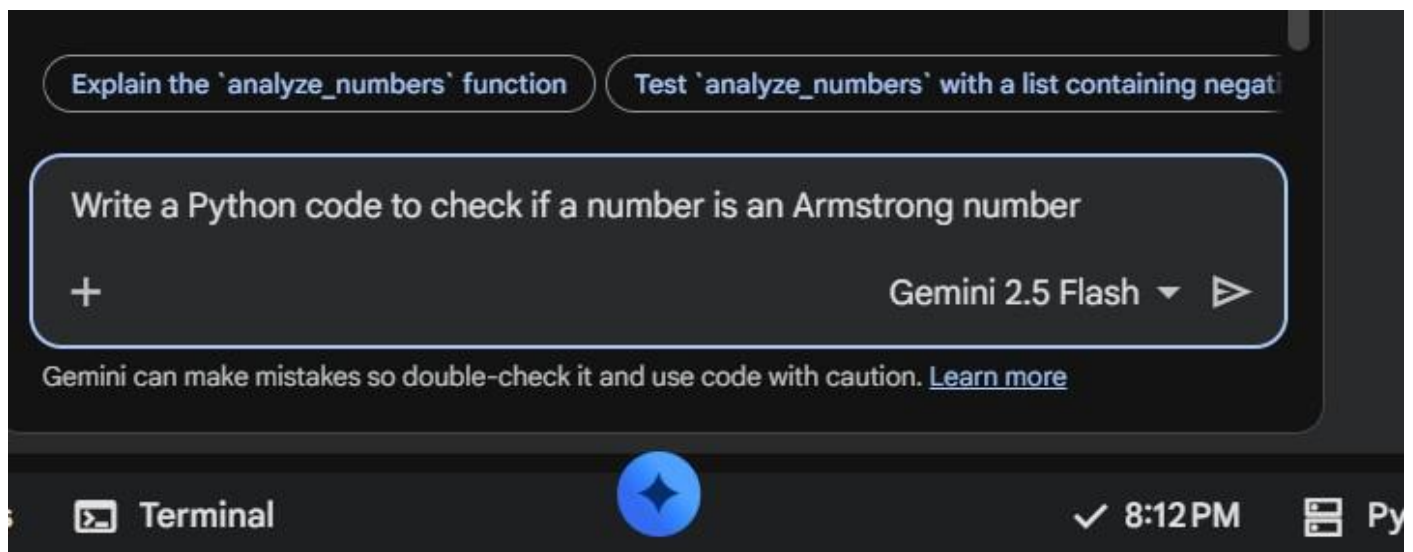
Task:

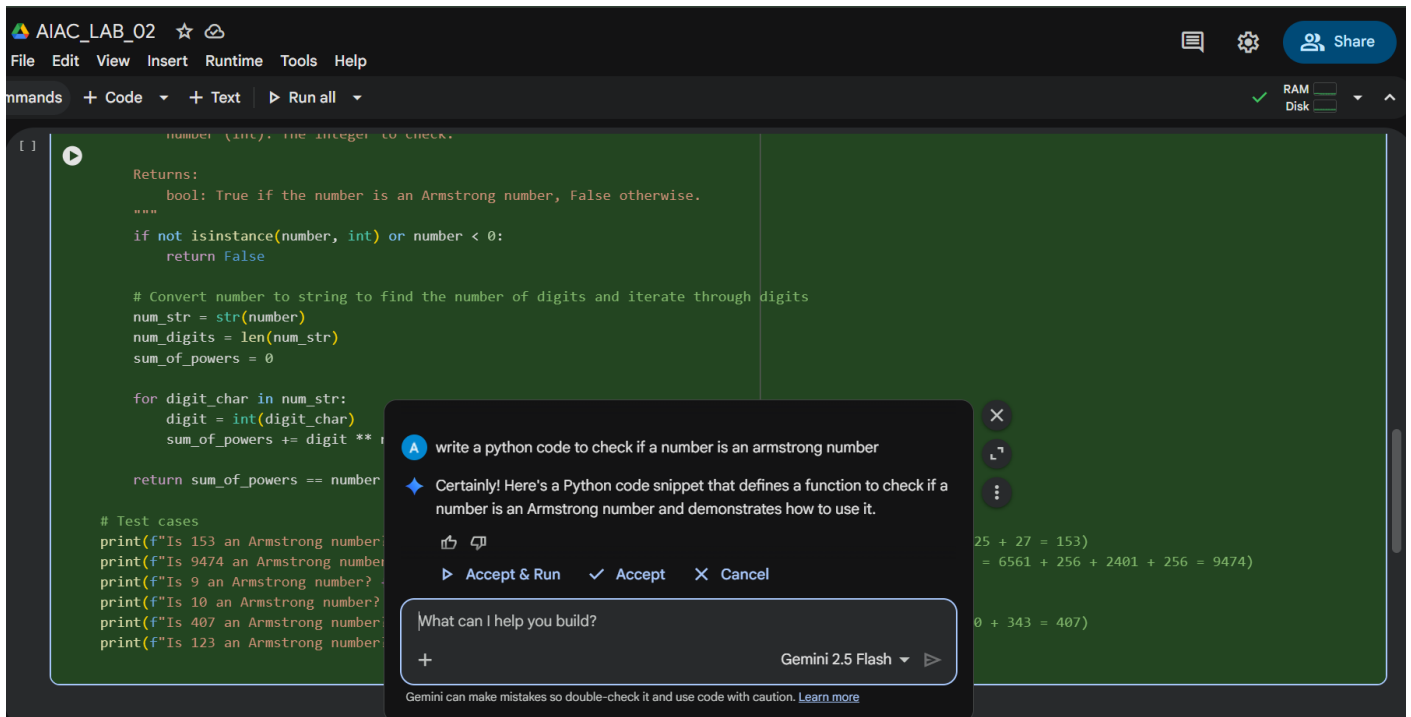
- Generate an Armstrong number checker using Gemini and GitHub Copilot.
- Compare their outputs, logic style, and clarity.

Expected Output:

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

Google Colab Workspace Screenshots:





Code:

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

```
    """
```

Checks if a 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, $153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$.

```
    """
```

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

```
    return False
```

```
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
```

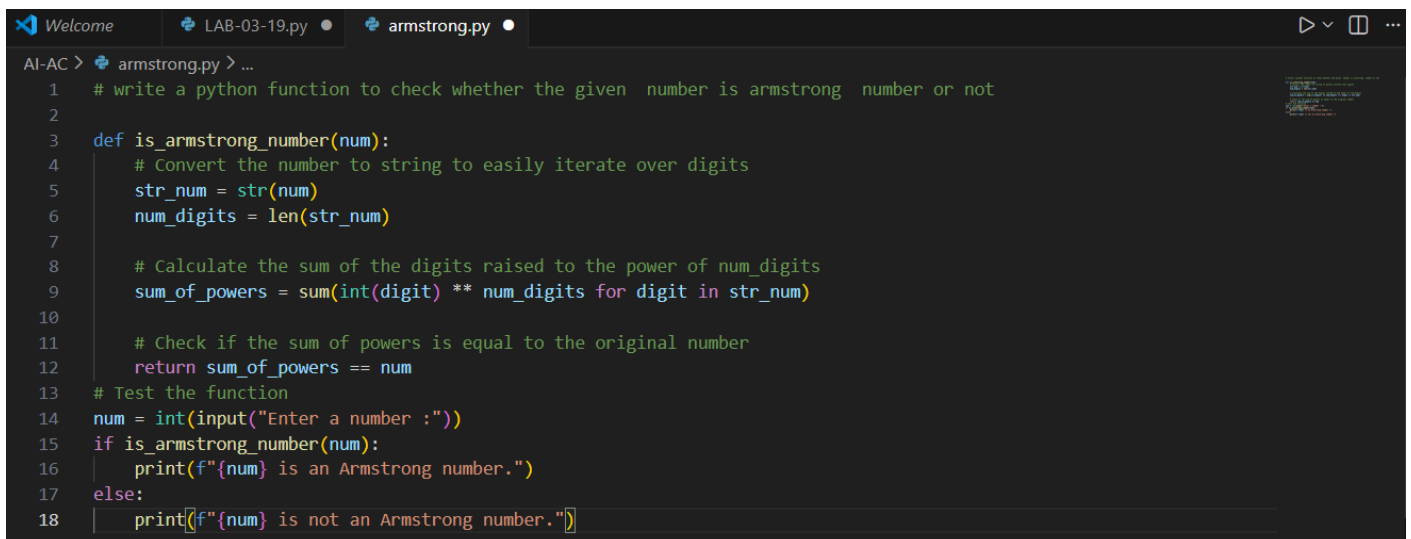
```
return sum_of_powers == number
```

Test cases

```
print(f'Is 153 an Armstrong number? {is_armstrong(153)}') # Expected: True
print(f'Is 9 an Armstrong number? {is_armstrong(9)}') # Expected: True
print(f'Is 370 an Armstrong number? {is_armstrong(370)}') # Expected: True
print(f'Is 123 an Armstrong number? {is_armstrong(123)}') # Expected: False
print(f'Is 1634 an Armstrong number? {is_armstrong(1634)}') # Expected: True
print(f'Is 0 an Armstrong number? {is_armstrong(0)}') # Expected: True (by some definitions)
print(f'Is -153 an Armstrong number? {is_armstrong(-153)}') # Expected: False
```

OUTPUT:

```
... Is 153 an Armstrong number? True
     Is 9474 an Armstrong number? True
     Is 9 an Armstrong number? True
     Is 10 an Armstrong number? False
     Is 407 an Armstrong number? True
     Is 123 an Armstrong number? False
```



The screenshot shows a code editor with a dark theme. The top bar has tabs for 'Welcome', 'LAB-03-19.py', and 'armstrong.py'. The main editor area shows the following Python code:

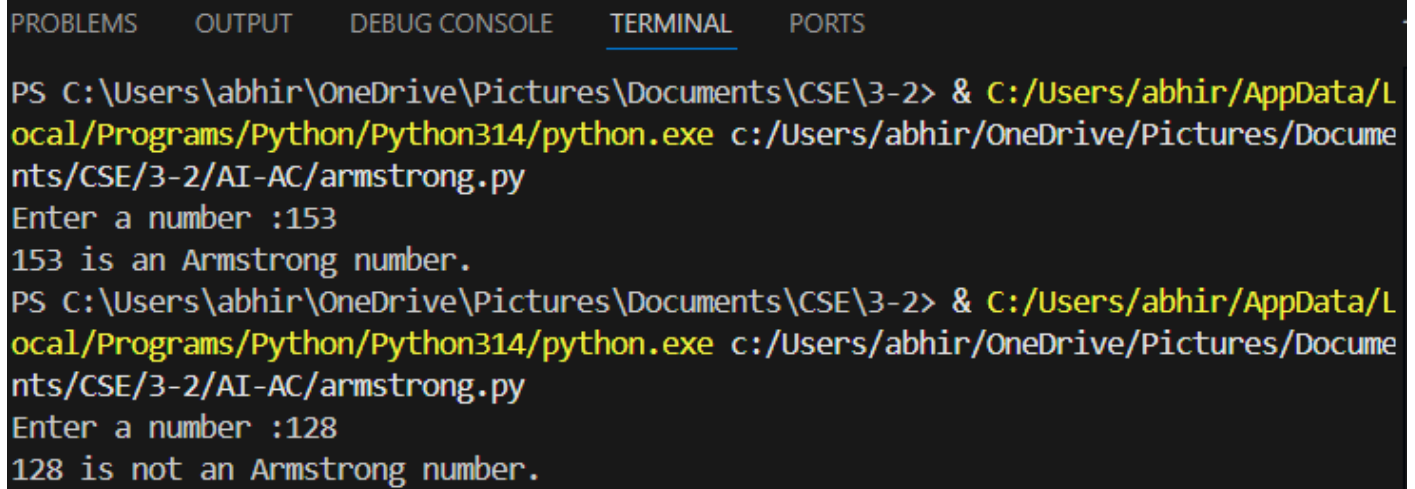
```
1 # write a python function to check whether the given number is armstrong number or not
2
3 def is_armstrong_number(num):
4     # Convert the number to string to easily iterate over digits
5     str_num = str(num)
6     num_digits = len(str_num)
7
8     # Calculate the sum of the digits raised to the power of num_digits
9     sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
10
11     # Check if the sum of powers is equal to the original number
12     return sum_of_powers == num
13
14 # Test the function
15 num = int(input("Enter a number :"))
16 if is_armstrong_number(num):
17     print(f"{num} is an Armstrong number.")
18 else:
19     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:



The screenshot shows a terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active. The command prompt shows the execution of the program armstrong.py. The first run uses the input 153, which is correctly identified as an Armstrong number. The second run uses the input 128, which is correctly identified as not an Armstrong number.

```
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local/Programs/Python/Python314/python.exe c:/Users/abhir/OneDrive/Pictures/Documents/CSE/3-2/AI-AC/armstrong.py  
Enter a number :153  
153 is an Armstrong number.  
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local/Programs/Python/Python314/python.exe c:/Users/abhir/OneDrive/Pictures/Documents/CSE/3-2/AI-AC/armstrong.py  
Enter a number :128  
128 is not an Armstrong number.
```

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.

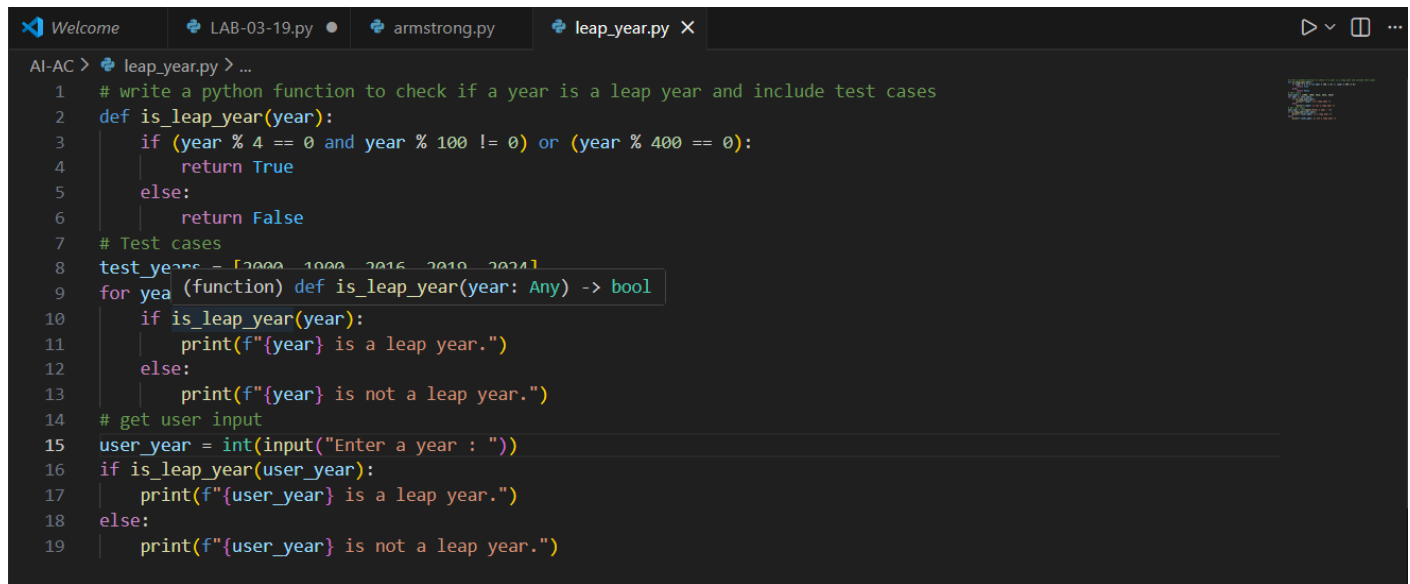
Expected Output:

- 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.

A screenshot of a code editor with a dark theme. The editor has several tabs at the top: 'Welcome', 'LAB-03-19.py', 'armstrong.py', and 'leap_year.py' (which is active). The code in the active tab is a Python script. It starts with a comment '# write a python function to check if a year is a leap year and include test cases'. Then it defines a function 'def is_leap_year(year):'. Inside the function, there is an 'if' statement: 'if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):'. If the condition is true, it returns 'True'; otherwise, it returns 'False'. After the function definition, there is a comment '# Test cases'. Then it defines a list 'test_years = [2000, 1900, 2004, 2001, 2100, 2400]'. It then enters a 'for' loop: 'for year in test_years:'. Inside the loop, it calls 'if is_leap_year(year):'. If true, it prints 'print(f"{year} is a leap year.")'; otherwise, it prints 'print(f"{year} is not a leap year.")'. After the loop, there is a comment '# get user input'. Then it takes user input: 'user_year = int(input("Enter a year : "))'. Finally, it checks 'if is_leap_year(user_year):' and prints the result accordingly. The code is numbered from 1 to 19 on the left margin.

```
1 # write a python function to check if a year is a leap year and include test cases
2 def is_leap_year(year):
3     if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):
4         return True
5     else:
6         return False
7 # Test cases
8 test_years = [2000, 1900, 2004, 2001, 2100, 2400]
9 for year in test_years:
10     if is_leap_year(year):
11         print(f"{year} is a leap year.")
12     else:
13         print(f"{year} is not a leap year.")
14 # get user input
15 user_year = int(input("Enter a year : "))
16 if is_leap_year(user_year):
17     print(f"{user_year} is a leap year.")
18 else:
19     print(f"{user_year} is not a leap year.")
```

Code:

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

def is_leap_year(year):

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):

return True

else:

return False

Test cases

test_years = [2000, 1900, 2004, 2001, 2100, 2400]

for year in test_years:

if is_leap_year(year):

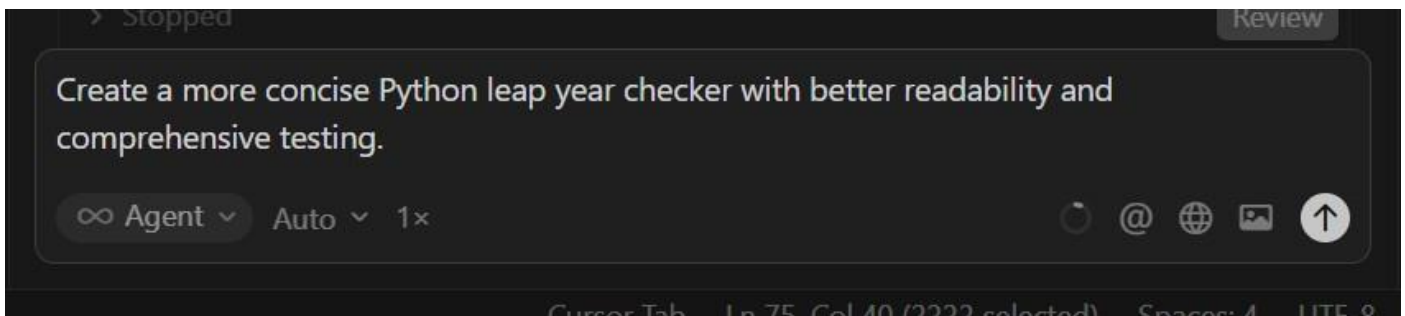
print(f"{year} is a leap year.")

```

else:
    print(f"{year} is not a leap year.")
# Get input from user
user_year = int(input("Enter a year to check if it's a leap year: "))
if is_leap_year(user_year):
    print(f"{user_year} is a leap year.")
else:
    print(f"{user_year} is not a leap year.")

```

Version—02



Code”:

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

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

```
    """
```

Checks if a given year is a leap year.

A year is a leap year if:

- It is divisible by 4,
- EXCEPT if it is divisible by 100,

- UNLESS it is also divisible by 400.

"""

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):

return True

else:

return False

Test cases

print(f"Is 2000 a leap year? {is_leap_year(2000)}") # Expected: True (divisible by 400)

print(f"Is 1900 a leap year? {is_leap_year(1900)}") # Expected: False (divisible by 100 but not 400)

print(f"Is 2024 a leap year? {is_leap_year(2024)}") # Expected: True (divisible by 4 and not by 100)

print(f"Is 2023 a leap year? {is_leap_year(2023)}") # Expected: False (not divisible by 4)

print(f"Is 1600 a leap year? {is_leap_year(1600)}") # Expected: True

output:

```
Is 2000 a leap year? True
Is 1900 a leap year? False
Is 2024 a leap year? True
Is 2023 a leap year? False
Is 1600 a leap year? True
```

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:

A screenshot of a code editor with a dark theme. The editor has several tabs at the top: 'Welcome', 'LAB-03-19.py', 'armstrong.py', 'leap_year.py', and 'even_odd_sum.py'. The 'even_odd_sum.py' tab is active. The code is written in Python and is as follows:

```
1 def even_odd_sum(num):
2     even_sum = 0
3     odd_sum = 0
4
5     for i in range(1, num + 1):
6         if i % 2 == 0:
7             even_sum += i
8         else:
9             odd_sum += i
10
11     return even_sum, odd_sum
12
13 number = int(input("Enter a number: "))
14 even_sum, odd_sum = even_odd_sum(number)
15 print(f"Sum of even numbers from 1 to {number} is: {even_sum}")
16 print(f"Sum of odd numbers from 1 to {number} is: {odd_sum}")
```

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: "))

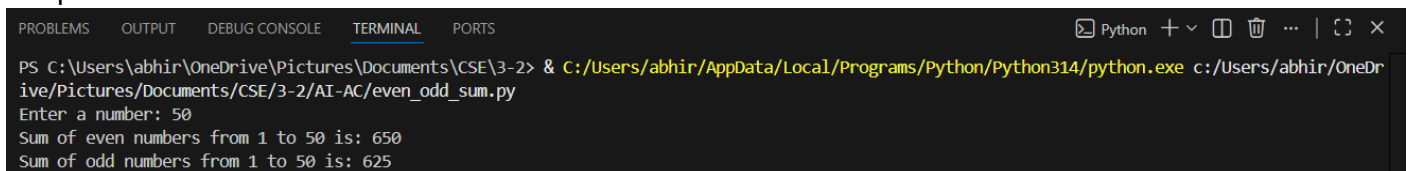
even, odd = even_odd_sum_tuple(user_input)

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

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

```

Output:



```

PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local/Programs/Python/Python314/python.exe c:/Users/abhir/OneDrive/Pictures/Documents/CSE/3-2/AI-AC/even_odd_sum.py
Enter a number: 50
Sum of even numbers from 1 to 50 is: 650
Sum of odd numbers from 1 to 50 is: 625

```

Refactoring the code :



```

1  def even_odd_sum(num):
2
3      if i % 2 == 0:
4          even_sum += i
5      else:
6          odd_sum += i
7
8      return even_sum, odd_sum
9
10
11
12
13 number = int(input("Enter a number: "))
14 even_sum, odd_sum = even_odd_sum(number)
15 print(f"Sum of even numbers from 1 to {number} is: {even_sum}")

```

Generate code ✓ ✕

Add Context... Auto ▾

```

16 print(f"Sum of odd numbers from 1 to {number} is: {odd_sum}")
17 while number <= 0:
18     print("Please enter a positive number.")
19     number = int(input("Enter a number: "))

```

AI-AC > even_odd_sum.py > ...

```
1 def even_odd_sum(num):
2     even_sum = 0
3     odd_sum = 0
4
5     for i in range(1, num + 1):
6         if i % 2 == 0:
7             even_sum += i
8         else:
9             odd_sum += i
10
11     return even_sum, odd_sum
12
13 number = int(input("Enter a number: "))
14 even_sum, odd_sum = even_odd_sum(number)
15 while number <= 0:
16     print("Please enter a positive number.")
17     number = int(input("Enter a number: "))
18 print(f"Sum of odd numbers from 1 to {number} is: {odd_sum}")
19 print(f"Sum of even numbers from 1 to {number} is: {even_sum}")
20
```

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:


```
PS C:\Users\abhir\OneDrive\Pictures\Documents\CSE\3-2> & C:/Users/abhir/AppData/Local/Programs/Python/Python314/python.exe c:/Users/abhir/OneDrive/Pictures/Documents/CSE/3-2/AI-AC/even_odd_sum.py
Enter a number: 99
Sum of odd numbers from 1 to 99 is: 2500
Sum of even numbers from 1 to 99 is: 2450
```

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).