AIAssistedCoding

Assignment - 02

Name : B.Sushma

RollNo:2303A54040

Batch: 48

Lab2:ExploringAdditionalAICodingToolsbeyondCopilot –Gemini(Colab)and Cursor AI

Task1:StatisticalSummaryforSurveyData

- **Scenario:Youareadataanalystinternworking withsurveyresponsesstoredas numerical lists.**
  - **Promptused:**
    "Write a Python function that takes a list of numbers and returns the mean, minimum, and maximum values."
  - **Generatedcode:**

```
importstatistics
defanalyze_numbers(numbers):
"""
Calculatesthemean,minimum,andmaximumofalistofnumbers. Args:
numbers(list):Alistofnumericalvalues.
Returns:
tuple:Atuplecontaining(mean,minimum,maximum). Returns
(None, None, None) if the list is empty.
"""
ifnot numbers:
return None, None,None
mean_value=statistics.mean(numbers)
min_value = min(numbers)
max_value= max(numbers)
returnmean_value,min_value, max_value
```

Example Usage

```
data=[10,20, 30,40, 50]
mean,minimum,maximum=analyze_numbers(data)
print(f"List: {data}")
print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum:{maximum}")
```

```
data_empty = []
mean_empty,min_empty,max_empty=analyze_numbers(data_empty)
print(f"\nList: {data_empty}")
print(f"Mean: {mean_empty}")
print(f"Minimum: {min_empty}")
print(f"Maximum:{max_empty}")
```

- **SampleOutput:**

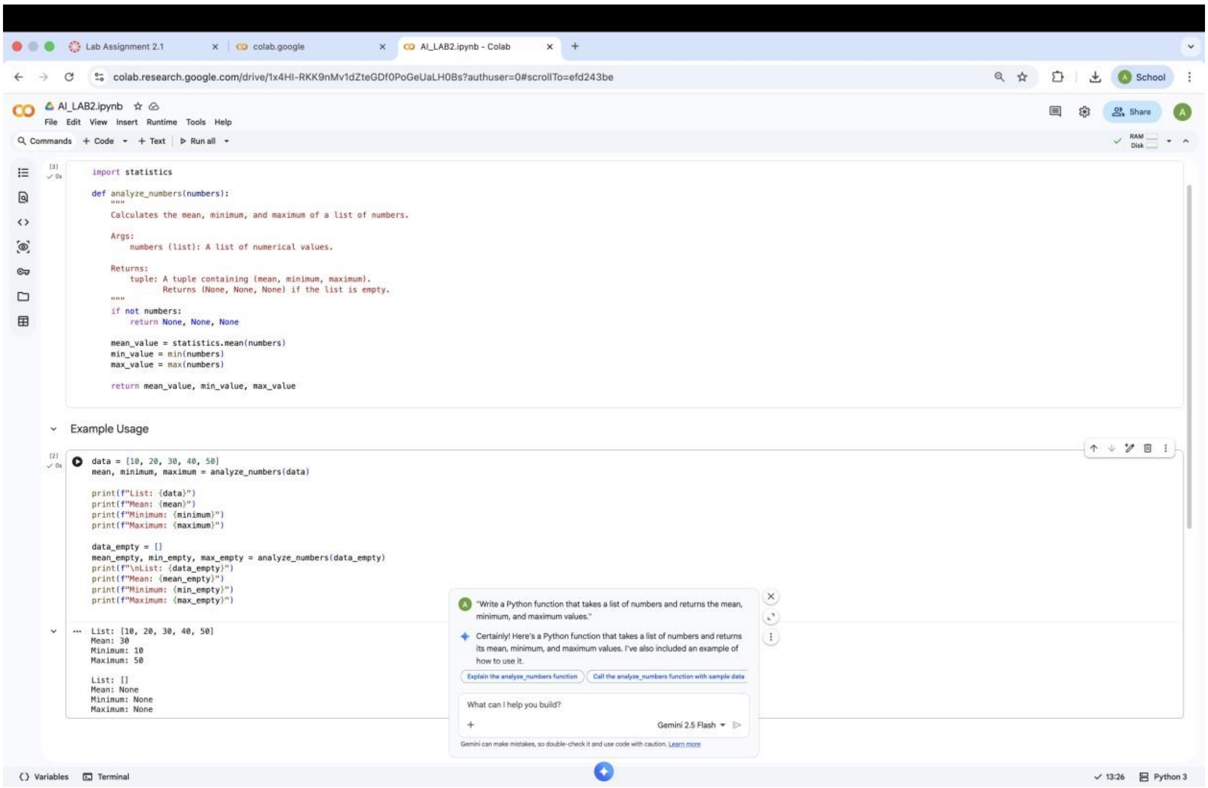
```
List:[10,20,30,40,50]Mean:30Minimum:10Maximum:50 List: []
Mean: None Minimum: None Maximum: None
```

- **ShortExplanationofLogic:**

The **mean** represents the average of a set of numbers. It is obtained by adding all the values in the list and dividing the sum by the total number of elements.

The**minimumvalue**isthesmallestnumberpresentinthelist,andthe**maximumvalue** is the largest number present in the list. These values are calculated using Python's built-in min() and max() functions.

Thefunctionacceptsalistofnumericalvaluesasinput,computesthemean,minimum, and maximum values, and displays them as a statistical summary of the given data.



```
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:
        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 = statistics.mean(numbers)
    min_value = min(numbers)
    max_value = max(numbers)

    return mean_value, min_value, max_value
```

Example Usage

```
data = [10, 20, 30, 40, 50]
mean, minimum, maximum = analyze_numbers(data)

print(f"List: {data}")
print(f"Mean: {mean}")
print(f"Minimum: {minimum}")
print(f"Maximum: {maximum}")

data_empty = []
mean_empty, min_empty, max_empty = analyze_numbers(data_empty)
print(f"\nList: {data_empty}")
print(f"Mean: {mean_empty}")
print(f"Minimum: {min_empty}")
print(f"Maximum: {max_empty}")
```

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

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

"Write a Python function that takes a list of numbers and returns the mean, minimum, and maximum values."

Certainly! Here's a Python function that takes a list of numbers and returns its mean, minimum, and maximum values. I've also included an example of how to use it.

Explain the analyze_numbers function
Call the analyze_numbers function with sample data
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

Task2:ArmstrongNumber–AI Comparison

❖ Scenario:**YouareevaluatingAItoolsfornumericvalidation logic.**

- **Promptused(Gemini):**

"WriteaPythonprogramtocheckwhetheragivennumberis anArmstrongnumber."

- **Promptused(Copilot):**
  "WriteaPythonprogramtocheckwhetheragivennumberis anArmstrongnumber."
- **Generatedcode(Gemini):**

```
defis_armstrong_number(number):
"""
Checksifa givennumber isanArmstrongnumber.
AnArmstrongnumber(alsoknownasanarcissisticnumber,
pluperfect digital invariant, or plus perfect number)
isanumberthatisthesumofitsowndigitseachraised to the
power of the number of digits.
rgs:
number(int):Theintegertocheck. Returns:
bool:TrueifthenumberisanArmstrongnumber,Falseotherwise. """
ifnotisinstance(number,int)ornumber<0: return
False
num_str=str(number) n
= len(num_str)
sum_of_powers = 0
for digit_char in num_str:
digit = int(digit_char)
sum_of_powers+=digit**n
returnsum_of_powers== number
```

- **Generatedcode(Copilot):**

```
#!/usr/bin/envpython3"""Armstrong
numberchecker(task2).

Usage:
python3task2.py153
python3task2.py#promptsforinput"""
importargparsei
mportsys



defis_armstrong(n: int)->bool:
if n<0:
returnFalse
digits=[int(d)fordinstr(n)]p =
len(digits)
return sum(d**pfor d indigits) ==n



defmain() ->None:
parser=argparse.ArgumentParser(description='CheckArmstrongnumbers')par
ser.add_argument('number', nargs='?', type=int, help='integer to check')args =
parser.parse_args()
```

```
ifargs.numberisNone:
try:
s=input('Enteranon-negativeinteger:').strip()num =
int(s)
except (EOFError,
ValueError):print('Invalidinput',file=s
ys.stderr)sys.exit(1)
else:
num= args.number

if is_armstrong(num):
print(f"{num}isanArmstrongnumber.")else:
print(f"{num}is notanArmstrongnumber.")



ifname== 'main':main()
  _      _    _     _
```

- **SampleInput(Gemini):**

```
#Testwithsomenumbers
print(f"Is153anArmstrongnumber?{is_armstrong_number(153)}")#Expected:True (1^3
+ 5^3 + 3^3 = 1 + 125 + 27 = 153)
print(f"Is 9474 an Armstrong number? {is_armstrong_number(9474)}") # Expected:
True (9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 + 2401 + 256 = 9474)
print(f"Is9anArmstrongnumber?{is_armstrong_number(9)}")#Expected:True(9^1
= 9)
print(f"Is 10 an Armstrong number? {is_armstrong_number(10)}") # Expected: False
(1^2 + 0^2 = 1 != 10)
print(f"Is123anArmstrongnumber?{is_armstrong_number(123)}")#Expected: False
```

- **SampleInput(Copilot):**

```
Enteranon-negativeinteger:343
```

- **SampleOutput (Gemini):**

```
Is 153 an Armstrong number? True
Is9474anArmstrongnumber?True Is
9 an Armstrong number? True
Is 10 an Armstrong number? False
Is123anArmstrongnumber?False
```

- **SampleOutput(Copilot):**

```
343is notanArmstrongnumber.
```

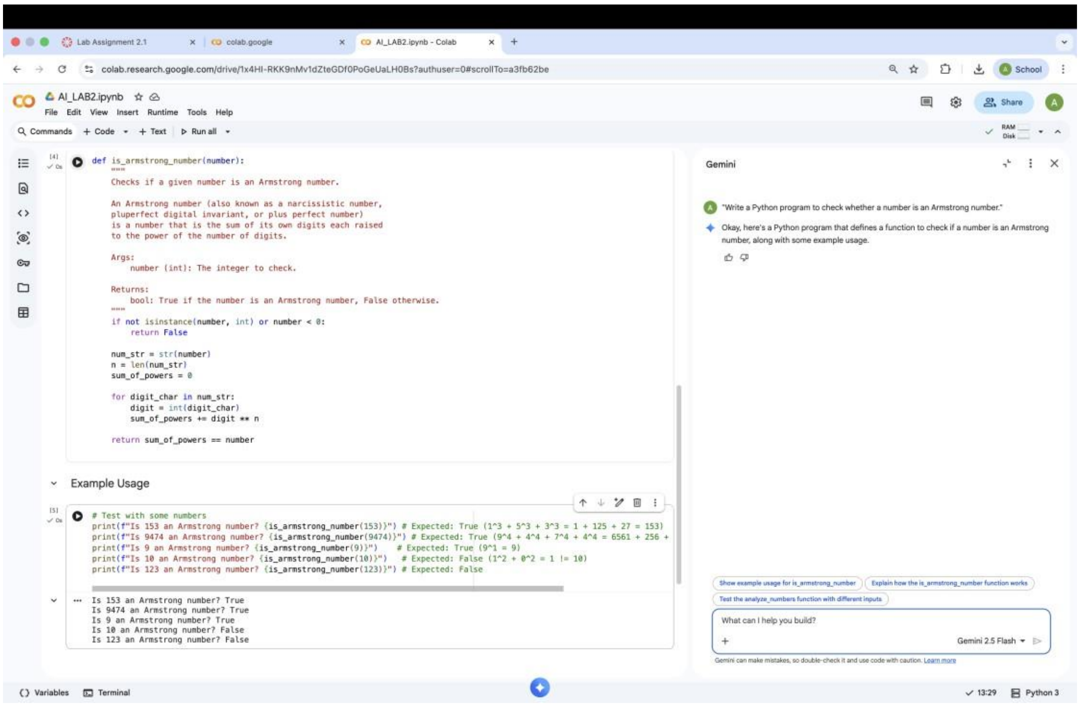
- **Side-by-SideComparisonTable:**

Feature	Google Gemini	GitHub Copilot
Code structure	Uses a separate function	Written directly in main logic
Readability	Very clear and modular	Simple and straightforward
Logic clarity	Easy to understand with function	Easy but less modular
Beginner friendly	High	Medium
Explanation support	Provides explanation along with code	No explanation, only code
Best use case	Learning and documentation	Fast coding inside editor

- **Side-by-SideComparisonTable:**

Analysis : GoogleGeminigenerated awell-structured and modular solution using afunction, making it suitable for learning and reuse. GitHub Copilot produced a concise and direct implementation, which is efficient for quick development within an IDE. Both tools correctly implementedtheArmstrongnumberlogic,butGeminiprovidedbetterclarityandexplanation, while Copilot focused on speed and simplicity.

Conclusion : Both AI tools effectively generated correct Armstrong number checking programs. Google Gemini is more helpful for understanding and learning due to its explanations, while GitHub Copilot is ideal for rapid code generation during development.


Lab Assignment 2.1
colab.google
AI_LAB2.ipynb - Colab
colab.research.google.com/drive/1x4HI-RKK9nMv1dZteGDf0PoGeUaLH0Bs?authuser=0#scrollTo=a3fb62be
School
AI_LAB2.ipynb
File
Edit
View
Insert
Runtime
Tools
Help
Commands
+ Code
+ Text
Run all
Share
RAM
Disk
Example Usage
Variables
Terminal
13:29
Python 3
Show example usage for is_armstrong_number
Explain how the is_armstrong_number function works
Test the analyze_numbers function with different inputs
What can I help you build?
Gemini 2.5 Flash

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

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

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

    Returns:
        bool: True if the number is an Armstrong number, False otherwise.
    """
    if not isinstance(number, int) or number < 0:
        return False

    num_str = str(number)
    n = len(num_str)
    sum_of_powers = 0

    for digit_char in num_str:
        digit = int(digit_char)
        sum_of_powers += digit ** n

    return sum_of_powers == number
```

```
# Test with some numbers
print(f"Is 153 an Armstrong number? {is_armstrong_number(153)}") # Expected: True (1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153)
print(f"Is 9474 an Armstrong number? {is_armstrong_number(9474)}") # Expected: True (9^4 + 4^4 + 7^4 + 4^4 = 6561 + 256 +
print(f"Is 9 an Armstrong number? {is_armstrong_number(9)}")    # Expected: True (9^1 = 9)
print(f"Is 10 an Armstrong number? {is_armstrong_number(10)}")   # Expected: False (1^2 + 0^2 = 1 != 10)
print(f"Is 123 an Armstrong number? {is_armstrong_number(123)}") # Expected: False
```

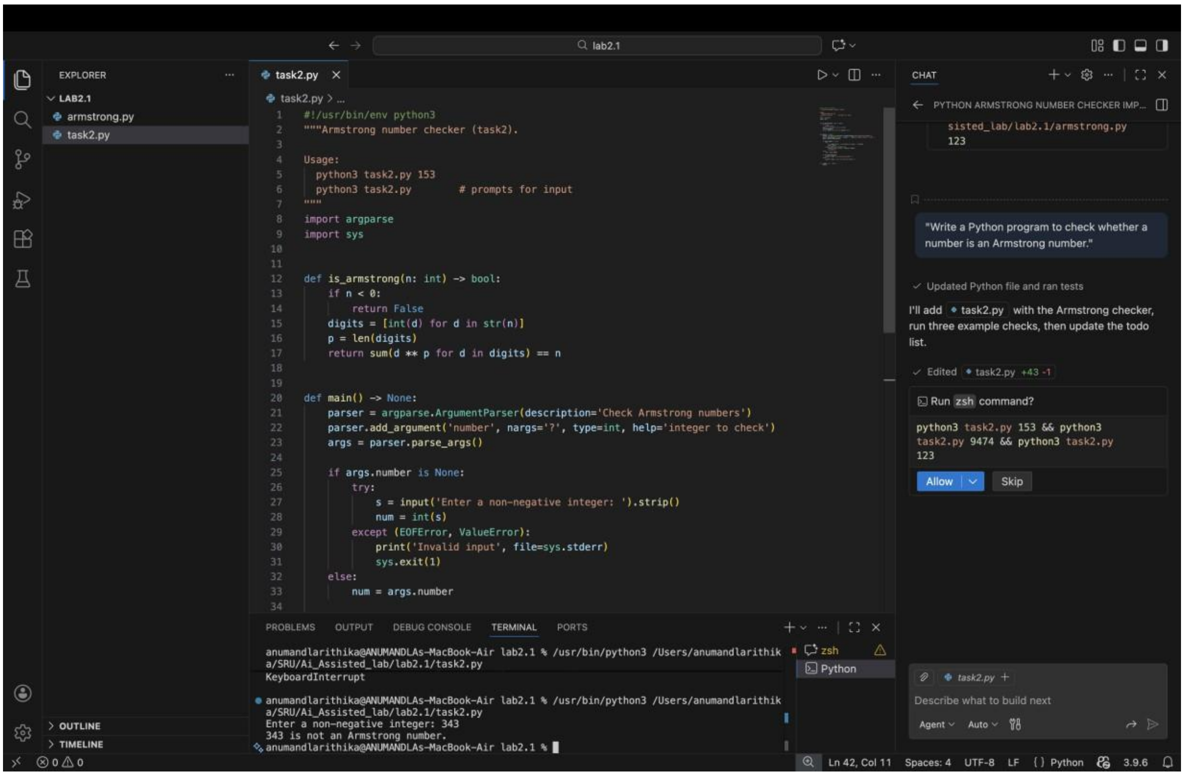
```
Is 153 an Armstrong number? True
Is 9474 an Armstrong number? True
Is 9 an Armstrong number? True
Is 10 an Armstrong number? False
Is 123 an Armstrong number? False
```

Gemini

"Write a Python program to check whether a number is an Armstrong number."

Okay, here's a Python program that defines a function to check if a number is an Armstrong number, along with some example usage.

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



```python
#!/usr/bin/env python3
"""Armstrong number checker (task2).

Usage:
  python3 task2.py 153
  python3 task2.py        # prompts for input
"""
import argparse
import sys


def is_armstrong(n: int) -> bool:
    if n < 0:
        return False
    digits = [int(d) for d in str(n)]
    p = len(digits)
    return sum(d ** p for d in digits) == n


def main() -> None:
    parser = argparse.ArgumentParser(description='Check Armstrong numbers')
    parser.add_argument('number', nargs='?', type=int, help='integer to check')
    args = parser.parse_args()

    if args.number is None:
        try:
            s = input('Enter a non-negative integer: ').strip()
            num = int(s)
        except (EOFError, ValueError):
            print('Invalid input', file=sys.stderr)
            sys.exit(1)
    else:
        num = args.number

```

```
anumandlarithika@ANUMANDLAs-MacBook-Air lab2.1 % /usr/bin/python3 /Users/anumandlarithik
a/SRU/Ai_Assisted_lab/lab2.1/task2.py
KeyboardInterrupt

anumandlarithika@ANUMANDLAs-MacBook-Air lab2.1 % /usr/bin/python3 /Users/anumandlarithik
a/SRU/Ai_Assisted_lab/lab2.1/task2.py
Enter a non-negative integer: 343
343 is not an Armstrong number.
anumandlarithika@ANUMANDLAs-MacBook-Air lab2.1 % 
```

PYTHON ARMSTRONG NUMBER CHECKER IMP...

```
sisted_lab/lab2.1/armstrong.py
123
```

"Write a Python program to check whether a number is an Armstrong number."

✓ Updated Python file and ran tests

I'll add task2.py with the Armstrong checker, run three example checks, then update the todo list.

✓ Edited task2.py +43 -1

Run zsh command?

```
python3 task2.py 153 && python3
task2.py 9474 && python3 task2.py
123
```

Task3:LeapYearValidationUsingCursorAI

❖ **Scenario:You arevalidating acalendarmoduleforabackend system.**

- **Prompt 1:**
  "WriteaPythonprogramtocheckwhether agivenyearis aleap year."
- **Generatedcode:**

```
year=int(input("Enterayear:")) #
A year is a leap year if:
#-Itis divisibleby4 AND
#-notdivisibleby100,UNLESS #- it
is also divisible by 400
if(year%4==0andyear%100!=0)or(year%400==0): print(f"{year}
   is a leap year.")
else:
   print(f"{year}isnotaleapyear.")
```

- **Prompt2:**
  "WriteanoptimizedPythonprogramwithproper conditionsandcommentstocheck whether a year is a leap year."
- **Generatedcode:**

```
year=int(input("Enterayear: "))


#Ayear is aleapyearif:
#1.Itis divisibleby400→leap year
#2.Elseifitisdivisibleby100→notaleapyear# 3.
Else if it is divisible by 4 → leap year
#4.Otherwise→notaleapyear
```

```
if year % 400 ==
0:print(f"{year}isaleapyear.")
elif year % 100 == 0:
print(f"{year}isnotaleap year.")
elif year % 4 ==
0:print(f"{year}isaleapyear.")
else:
print(f"{year}isnotaleap year.")
```

- **SampleInput:**

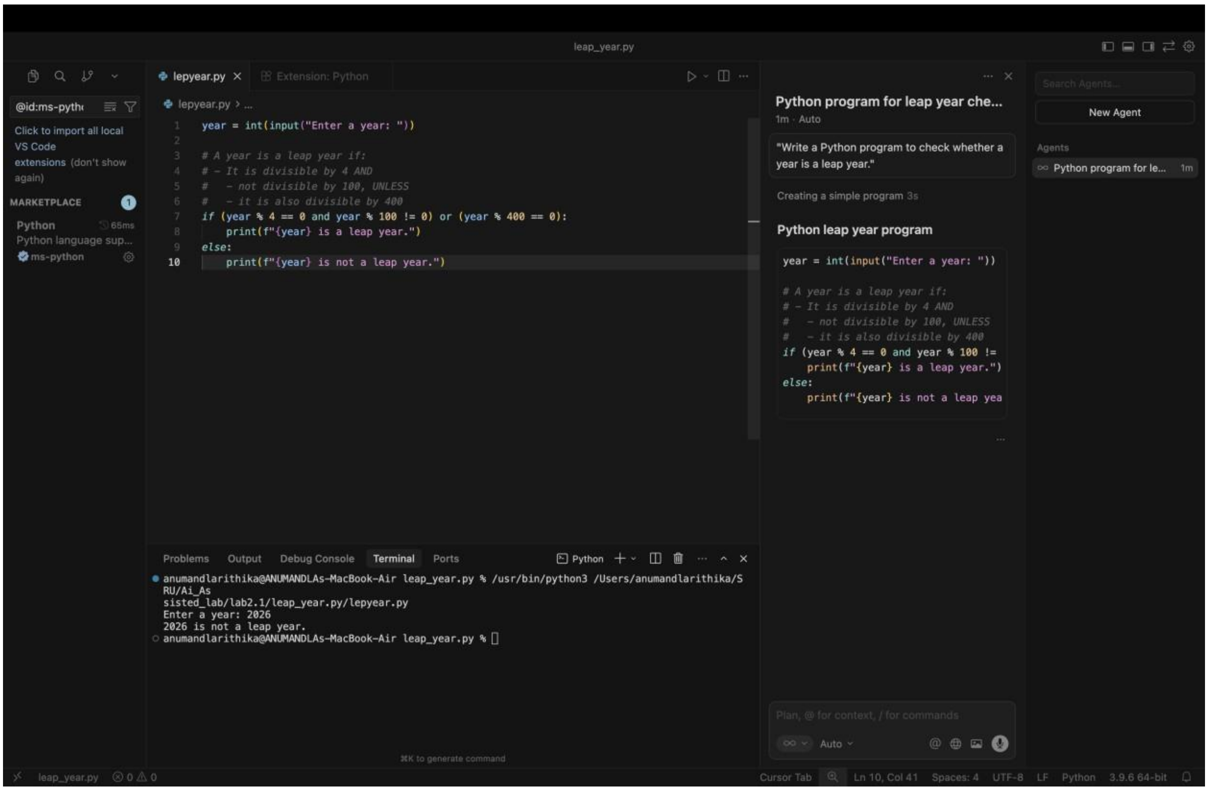
  Enterayear:2026

- **SampleOutput:**

  2026is notaleap year.

- **ShortExplanationofLogic:**

  ThefirstversionofthecodegeneratedbyCursorAIprovidedbasicleapyearvalidation logic. The second prompt resulted in improved code with better readability and clear commentsexplainingtheconditions.Theoptimizedversioniseasiertounderstand and more suitable for real-world applications.



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

# A year is a leap year if:
# - It is divisible by 4 AND
#   - not divisible by 100, UNLESS
#   - it is also divisible by 400
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.")
```

Python program for leap year che...

1m · Auto

"Write a Python program to check whether a year is a leap year."

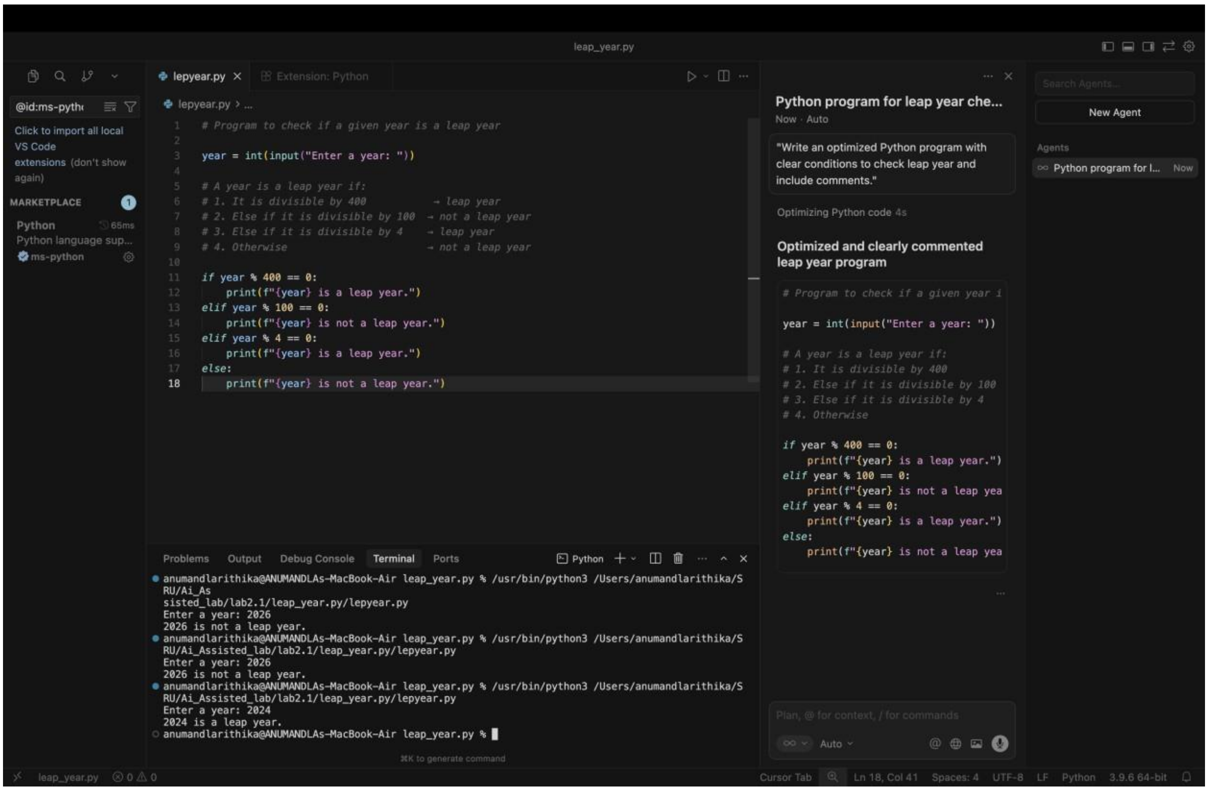
Creating a simple program 3s

Python leap year program

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

# A year is a leap year if:
# - It is divisible by 4 AND
#   - not divisible by 100, UNLESS
#   - it is also divisible by 400
if (year % 4 == 0 and year % 100 !=
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap yea
```

```
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % /usr/bin/python3 /Users/anumandlarithika/S
RU/Ai_As
sisted_lab/lab2.1/leap_year.py/lepyear.py
Enter a year: 2026
2026 is not a leap year.
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % 
```



```python
# Program to check if a given year is a leap year

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

# A year is a leap year if:
# 1. It is divisible by 400          → leap year
# 2. Else if it is divisible by 100  → not a leap year
# 3. Else if it is divisible by 4    → leap year
# 4. Otherwise                       → not a leap year

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

```
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % /usr/bin/python3 /Users/anumandlarithika/S
RU/Ai_As
sisted_lab/lab2.1/leap_year.py/lepyear.py
Enter a year: 2026
2026 is not a leap year.
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % /usr/bin/python3 /Users/anumandlarithika/S
RU/Ai_Assisted_lab/lab2.1/leap_year.py/lepyear.py
Enter a year: 2026
2026 is not a leap year.
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % /usr/bin/python3 /Users/anumandlarithika/S
RU/Ai_Assisted_lab/lab2.1/leap_year.py/lepyear.py
Enter a year: 2024
2024 is a leap year.
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py %
```

Python program for leap year che...

Now · Auto

"Write an optimized Python program with clear conditions to check leap year and include comments."

Optimizing Python code 4s

Optimized and clearly commented leap year program

```python
# Program to check if a given year i

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

# A year is a leap year if:
# 1. It is divisible by 400
# 2. Else if it is divisible by 100
# 3. Else if it is divisible by 4
# 4. Otherwise

if year % 400 == 0:
    print(f"{year} is a leap year.")
elif year % 100 == 0:
    print(f"{year} is not a leap yea
elif year % 4 == 0:
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap yea
```

Task4:StudentLogic+AIRefactoring(Odd/Even Sum)

- **Scenario:Companypolicyrequiresdeveloperstowritelogicbeforeusing AI.**
  - **Promptused:**
    "RefactorthisPythoncodetoimprovereadabilityandefficiency."
  - **StudentCode:**

    ```
    T =(1, 2, 3, 4, 5, 6,7)

    even_sum=0

    odd_sum=0 for

    i in t:

    if i% 2 == 0:

    even_sum=even_sum+i else:

    odd_sum= odd_sum+ i

    print("Sumofevennumbers:",even_sum)

    print("Sum of odd numbers:", odd_sum)
    ```

  - **AI Code:**

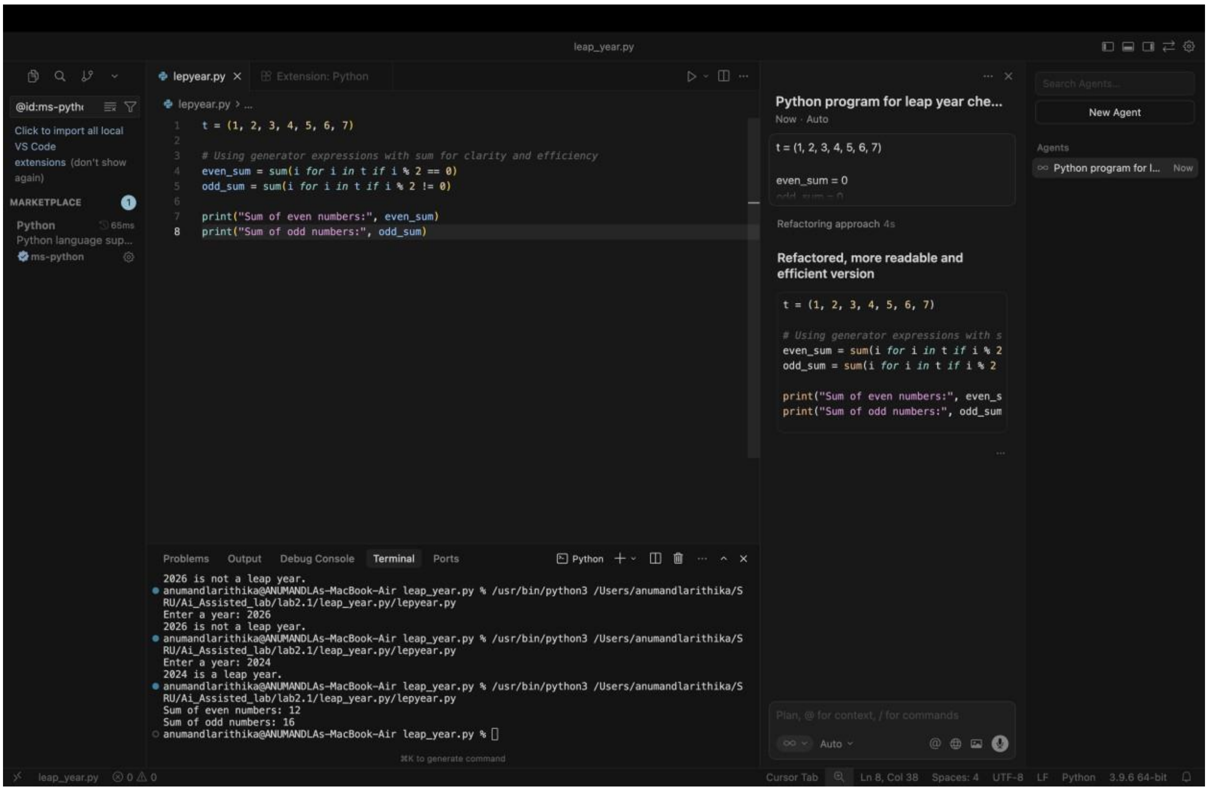
```
t= (1,2,3,4,5,6,7)

#Usinggeneratorexpressionswithsumforclarityand efficiency
even_sum=sum(iforintifi%2==0)odd_sum =
sum(i for i in t if i % 2 != 0)

print("Sumofevennumbers:",even_sum)print("Sum
of odd numbers:", odd_sum)
```

- **SampleOutput:**

  Sumofevennumbers:12
  Sum of odd numbers: 16



```python
t = (1, 2, 3, 4, 5, 6, 7)

# Using generator expressions with sum for clarity and efficiency
even_sum = sum(i for i in t if i % 2 == 0)
odd_sum = sum(i for i in t if i % 2 != 0)

print("Sum of even numbers:", even_sum)
print("Sum of odd numbers:", odd_sum)
```

Python program for leap year che...
Now · Auto

```
t = (1, 2, 3, 4, 5, 6, 7)

even_sum = 0
```

Refactoring approach 4s

Refactored, more readable and efficient version

```python
t = (1, 2, 3, 4, 5, 6, 7)

# Using generator expressions with s
even_sum = sum(i for i in t if i % 2
odd_sum = sum(i for i in t if i % 2

print("Sum of even numbers:", even_s
print("Sum of odd numbers:", odd_sum
```

```
2026 is not a leap year.
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % /usr/bin/python3 /Users/anumandlarithika/S
RU/Ai_Assisted_lab/lab2.1/leap_year.py/lepyear.py
Enter a year: 2026
2026 is not a leap year.
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % /usr/bin/python3 /Users/anumandlarithika/S
RU/Ai_Assisted_lab/lab2.1/leap_year.py/lepyear.py
Enter a year: 2024
2024 is a leap year.
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py % /usr/bin/python3 /Users/anumandlarithika/S
RU/Ai_Assisted_lab/lab2.1/leap_year.py/lepyear.py
Sum of even numbers: 12
Sum of odd numbers: 16
anumandlarithika@ANUMANDLAs-MacBook-Air leap_year.py %
```