AI ASSISTED CODING

Hall Ticket No: 2303A51851

Batch:13

Assignment-2.4

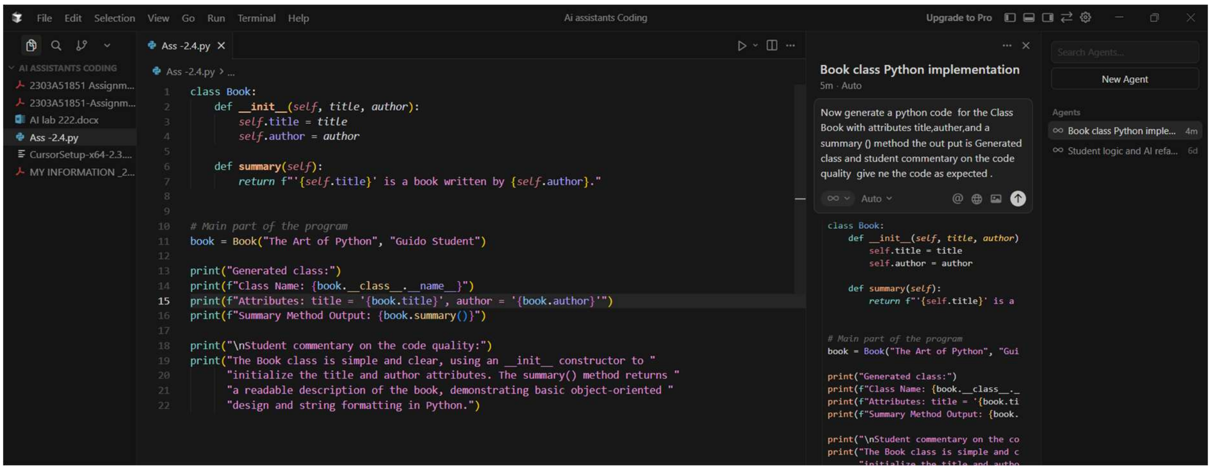
Task-1. Use Cursor AI to generate a Python class Book with attributes title, author,

and a summary() method.

Prompt :

#Now generate a python code for the Class Book with attributes title,auther,and a summary () method the out put is Generated class and student commentary on the code quality give ne the code as expected .

Code



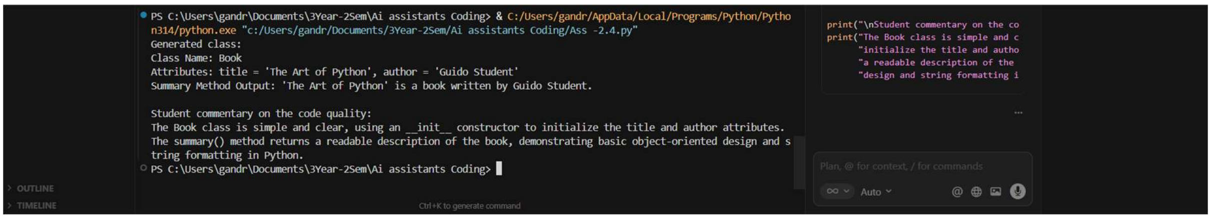
Book class Python implementation

5m · Auto

Now generate a python code for the Class Book with attributes title,auther,and a summary () method the out put is Generated class and student commentary on the code quality give ne the code as expected .

```python
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def summary(self):
        return f"'{self.title}' is a book written by {self.author}."


# Main part of the program
book = Book("The Art of Python", "Guido Student")

print("Generated class:")
print(f"Class Name: {book.__class__.__name__}")
print(f"Attributes: title = '{book.title}', author = '{book.author}'")
print(f"Summary Method Output: {book.summary()}")

print("\nStudent commentary on the code quality:")
print("The Book class is simple and clear, using an __init__ constructor to "
      "initialize the title and author attributes. The summary() method returns "
      "a readable description of the book, demonstrating basic object-oriented "
      "design and string formatting in Python.")
```

Output:


OUTLINE
TIMELINE
Ctrl+K to generate command
Plan, @ for context, / for commands
Auto

```
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Pytho
n314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Ass -2.4.py"
Generated class:
Class Name: Book
Attributes: title = 'The Art of Python', author = 'Guido Student'
Summary Method Output: 'The Art of Python' is a book written by Guido Student.

Student commentary on the code quality:
The Book class is simple and clear, using an __init__ constructor to initialize the title and author attributes.
The summary() method returns a readable description of the book, demonstrating basic object-oriented design and s
tring formatting in Python.
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding> 
```

Justification:

In this task, Cursor AI was used to generate a Python class for a book. The generated code followed proper object-oriented programming principles using a constructor and instance variables. The summary method clearly described the book details in a readable format. The code was simple, clean, and easy to understand for beginners. This shows that Cursor AI is effective in generating well-structured class-based code.

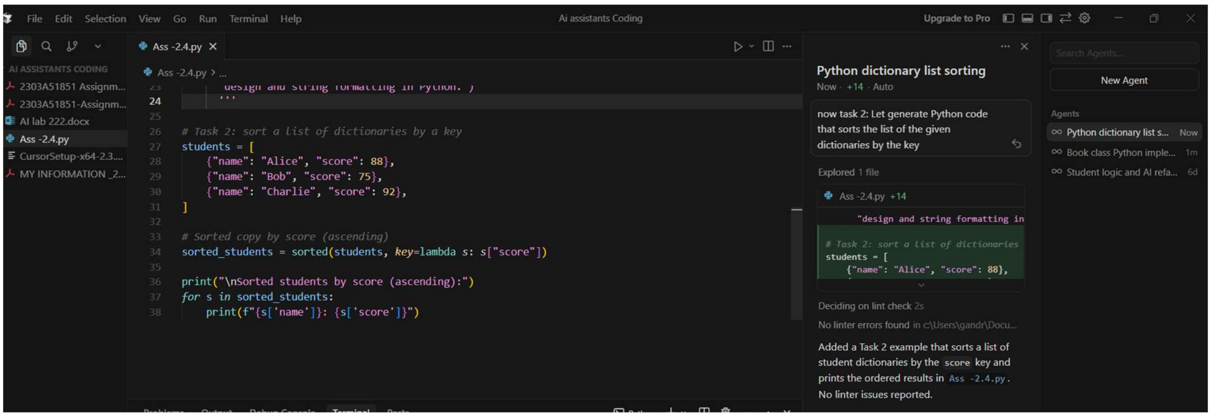
Task-2. Use Gemini and Cursor AI to generate code that sorts a list of dictionaries

by a key.

Prompt:

#Let generate Python code that sorts the list of the given dictionaries by the key

Code: In Curser AI



```
    '''

# Task 2: sort a list of dictionaries by a key
students = [
    {"name": "Alice", "score": 88},
    {"name": "Bob", "score": 75},
    {"name": "Charlie", "score": 92},
]

# Sorted copy by score (ascending)
sorted_students = sorted(students, key=lambda s: s["score"])

print("\nSorted students by score (ascending):")
for s in sorted_students:
    print(f"{s['name']}: {s['score']}")
```

Python dictionary list sorting

now task 2: Let generate Python code that sorts the list of the given dictionaries by the key

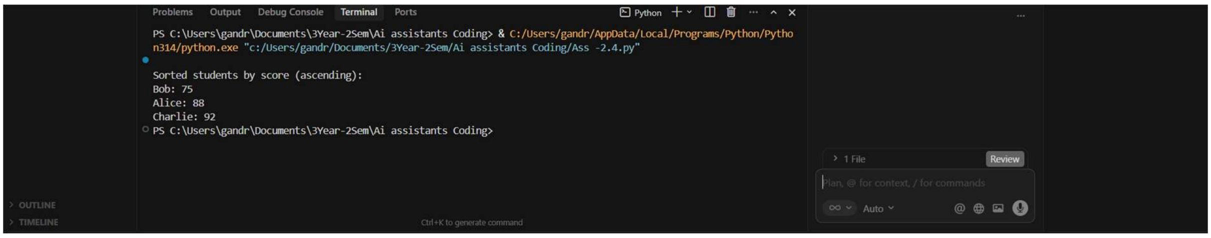
Explored 1 file

Deciding on lint check 2s

No linter errors found in c:\Users\gandr\Docu...

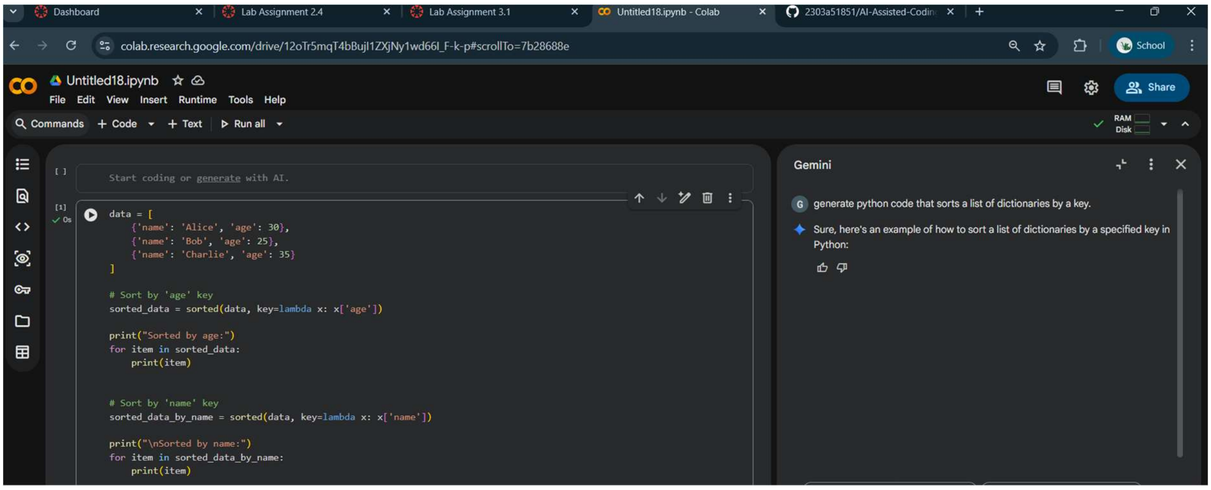
Added a Task 2 example that sorts a list of student dictionaries by the `score` key and prints the ordered results in `Ass -2.4.py`. No linter issues reported.

Output:



```
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Pytho
n314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Ass -2.4.py"

Sorted students by score (ascending):
Bob: 75
Alice: 88
Charlie: 92
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding>
```

Colab:

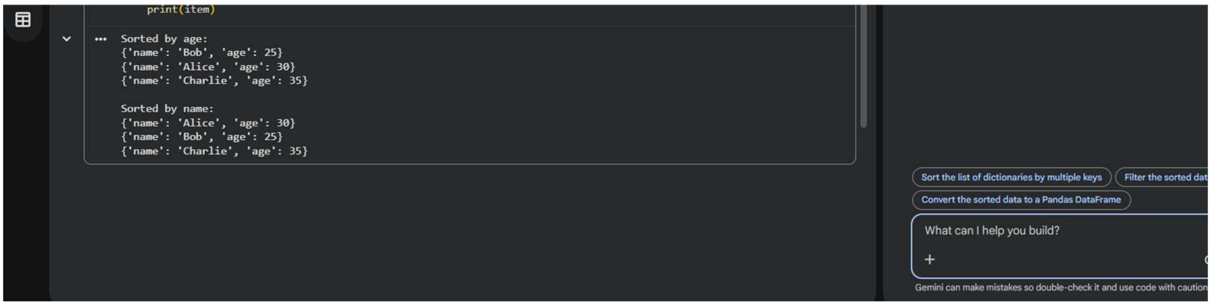


```
data = [
    {'name': 'Alice', 'age': 30},
    {'name': 'Bob', 'age': 25},
    {'name': 'Charlie', 'age': 35}
]

# Sort by 'age' key
sorted_data = sorted(data, key=lambda x: x['age'])

print("Sorted by age:")
for item in sorted_data:
    print(item)


# Sort by 'name' key
sorted_data_by_name = sorted(data, key=lambda x: x['name'])

print("\nSorted by name:")
for item in sorted_data_by_name:
    print(item)
```

generate python code that sorts a list of dictionaries by a key.

Sure, here's an example of how to sort a list of dictionaries by a specified key in Python:

Out Put:



```
        print(item)
```

```
Sorted by age:
{'name': 'Bob', 'age': 25}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}

Sorted by name:
{'name': 'Alice', 'age': 30}
{'name': 'Bob', 'age': 25}
{'name': 'Charlie', 'age': 35}
```

Sort the list of dictionaries by multiple keys
Filter the sorted dat
Convert the sorted data to a Pandas DataFrame
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution

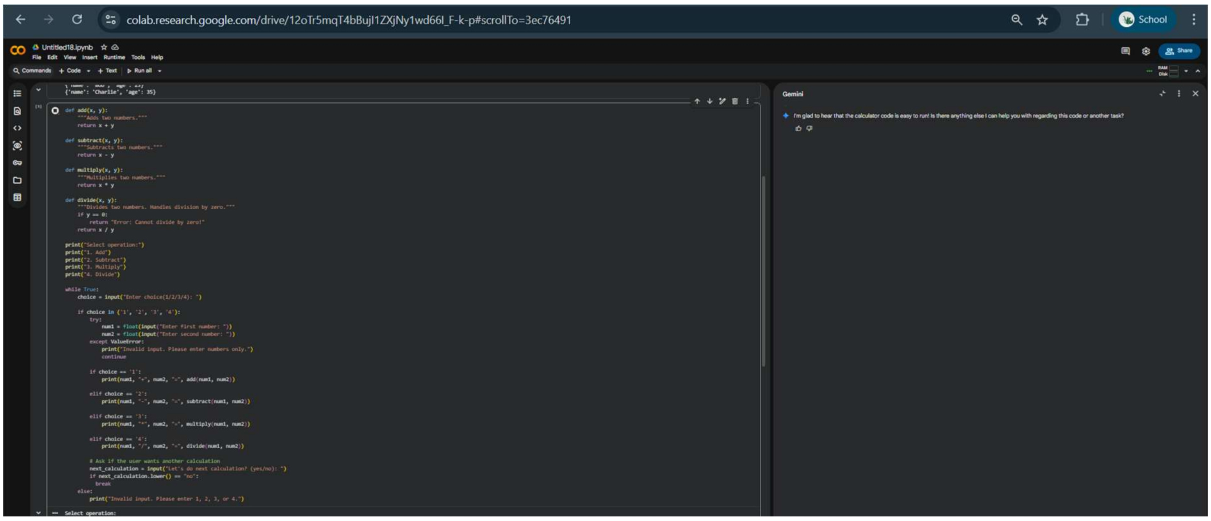
Justification:

In this task, both Gemini and Cursor AI were used to sort a list of dictionaries by age. Gemini generated clear and easy-to-understand code with better readability for beginners. Cursor AI provided a more optimized solution by performing in-place sorting. Both outputs produced correct results with good performance. This comparison highlights differences in clarity and optimization between AI tools

Task-3. Ask Gemini to generate a calculator using functions and explain how it works.

Prompt:

#Generate a Python program that calculator using functions and explain how it works



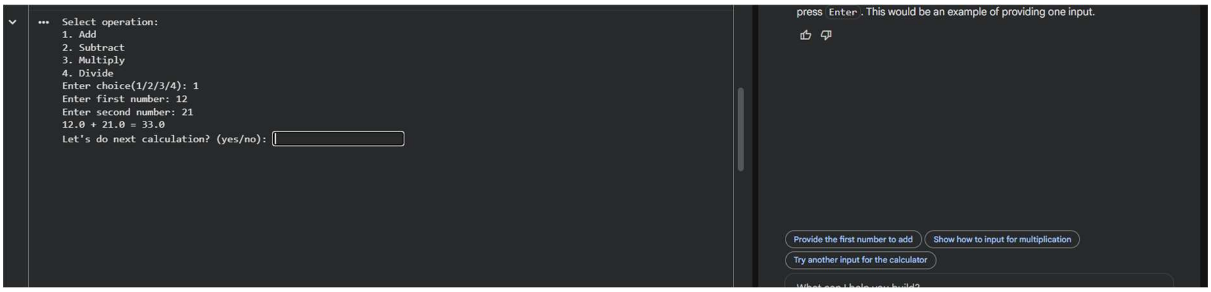
```
{'name': 'Charlie', 'age': 35}
```

```python
def add(x, y):
    """Adds two numbers."""
    return x + y

def subtract(x, y):
    """Subtracts two numbers."""
    return x - y

def multiply(x, y):
    """Multiplies two numbers."""
    return x * y

def divide(x, y):
    """Divides two numbers. Handles division by zero."""
    if y == 0:
        return "Error: Cannot divide by zero!"
    return x / y

print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")

while True:
    choice = input("Enter choice(1/2/3/4): ")

    if choice in ('1', '2', '3', '4'):
        try:
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
            print("Invalid input. Please enter numbers only.")
            continue

        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))

        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))

        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))

        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))

        # Ask if the user wants another calculation
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation.lower() == "no":
          break
    else:
        print("Invalid input. Please enter 1, 2, 3, or 4.")
```

```
Select operation:
```

I'm glad to hear that the calculator code is easy to run! Is there anything else I can help you with regarding this code or another task?

Output:

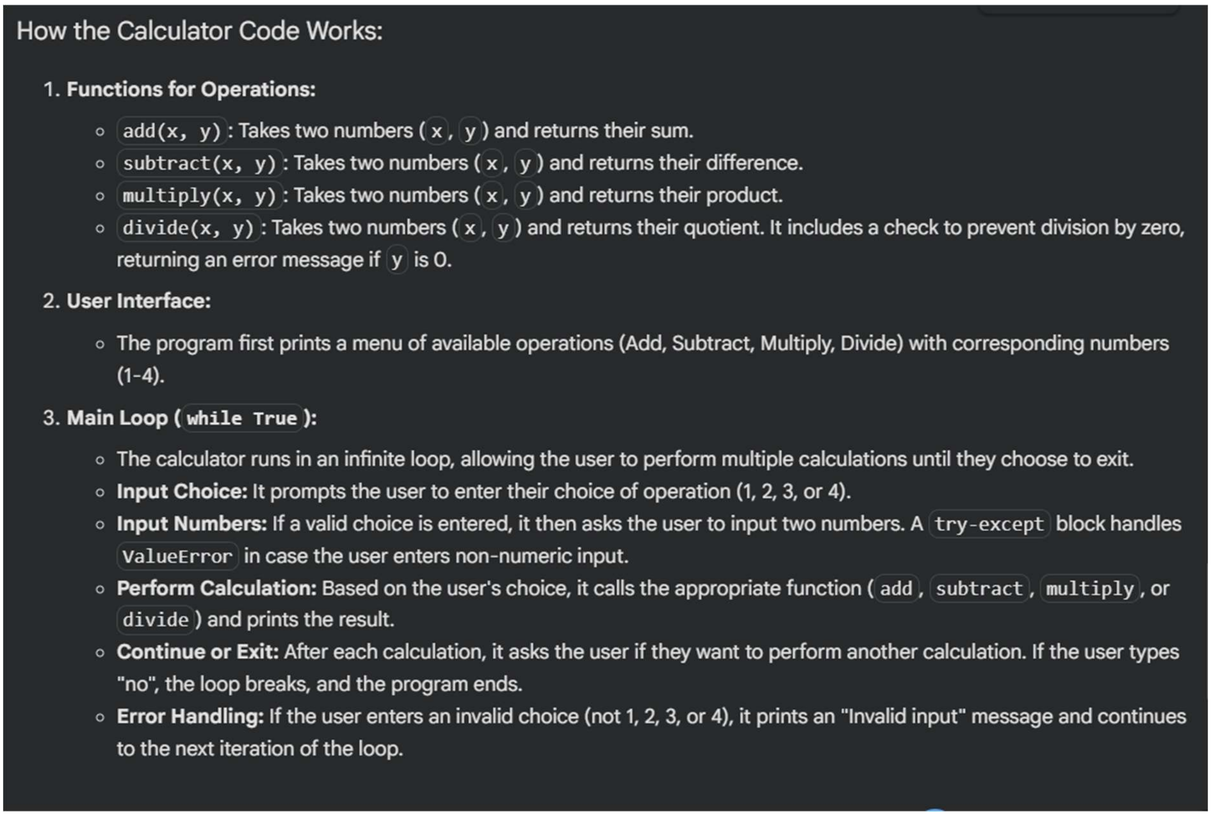


```
Select operation:
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice(1/2/3/4): 1
Enter first number: 12
Enter second number: 21
12.0 + 21.0 = 33.0
Let's do next calculation? (yes/no):
```

press `Enter`. This would be an example of providing one input.

Provide the first number to add
Show how to input for multiplication
Try another input for the calculator

How It works:



How the Calculator Code Works:

1. **Functions for Operations:**
    - `add(x, y)`: Takes two numbers (`x`, `y`) and returns their sum.
    - `subtract(x, y)`: Takes two numbers (`x`, `y`) and returns their difference.
    - `multiply(x, y)`: Takes two numbers (`x`, `y`) and returns their product.
    - `divide(x, y)`: Takes two numbers (`x`, `y`) and returns their quotient. It includes a check to prevent division by zero, returning an error message if `y` is 0.
2. **User Interface:**
    - The program first prints a menu of available operations (Add, Subtract, Multiply, Divide) with corresponding numbers (1-4).
3. **Main Loop (`while True`):**
    - The calculator runs in an infinite loop, allowing the user to perform multiple calculations until they choose to exit.
    - **Input Choice:** It prompts the user to enter their choice of operation (1, 2, 3, or 4).
    - **Input Numbers:** If a valid choice is entered, it then asks the user to input two numbers. A `try-except` block handles `ValueError` in case the user enters non-numeric input.
    - **Perform Calculation:** Based on the user's choice, it calls the appropriate function (`add`, `subtract`, `multiply`, or `divide`) and prints the result.
    - **Continue or Exit:** After each calculation, it asks the user if they want to perform another calculation. If the user types "no", the loop breaks, and the program ends.
    - **Error Handling:** If the user enters an invalid choice (not 1, 2, 3, or 4), it prints an "Invalid input" message and continues to the next iteration of the loop.

Justification:

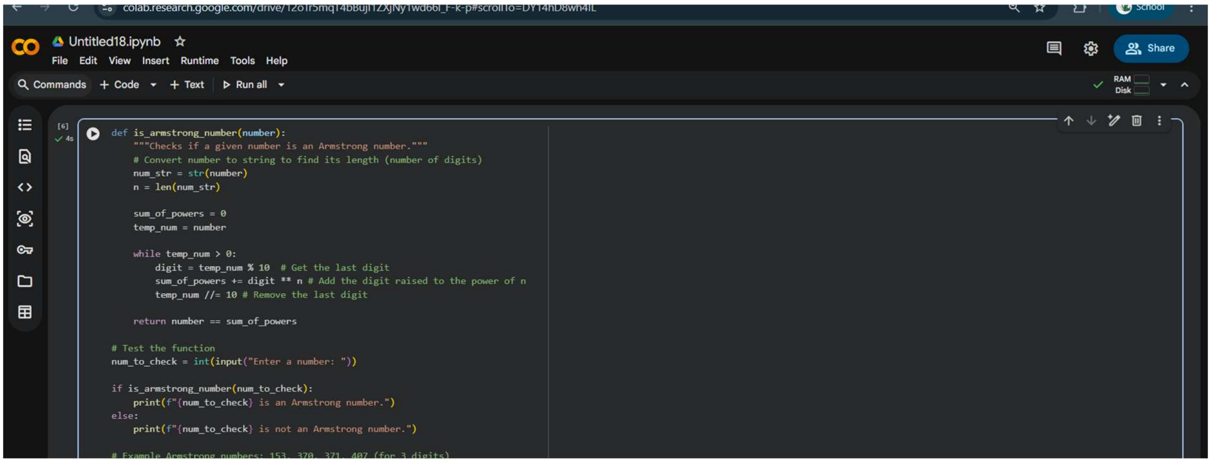
In this task, Gemini was used to generate a calculator program using functions. The calculator included basic operations such as addition, subtraction, multiplication, and division. Gemini also explained how each function works, which helped in understanding the logic. The use of functions improved modularity and code reuse. This task demonstrates Gemini's strength in explanation and learning support.

Task-4: Generate an Armstrong number program using Gemini, then improve it

using Cursor AI.

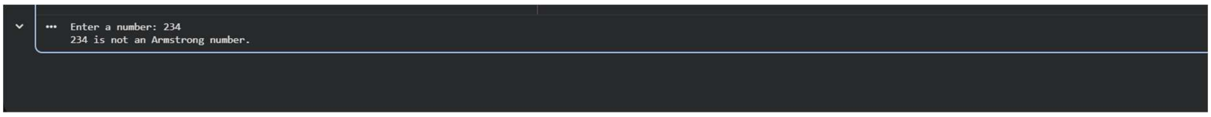
Prompt: Generate the Python code for the Armstrong Number.

Code:



```
def is_armstrong_number(number):
    """Checks if a given number is an Armstrong number."""
    # Convert number to string to find its length (number of digits)
    num_str = str(number)
    n = len(num_str)

    sum_of_powers = 0
    temp_num = number

    while temp_num > 0:
        digit = temp_num % 10  # Get the last digit
        sum_of_powers += digit ** n # Add the digit raised to the power of n
        temp_num //= 10 # Remove the last digit

    return number == sum_of_powers

# Test the function
num_to_check = int(input("Enter a number: "))

if is_armstrong_number(num_to_check):
    print(f"{num_to_check} is an Armstrong number.")
else:
    print(f"{num_to_check} is not an Armstrong number.")

# Example Armstrong numbers: 153, 370, 371, 407 (for 3 digits)
```

Out Put:

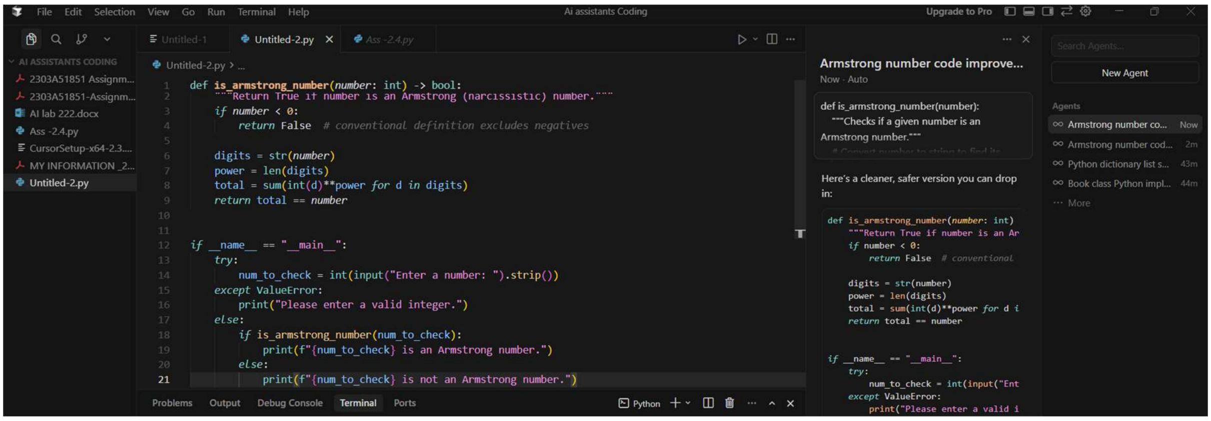


```
Enter a number: 234
234 is not an Armstrong number.
```

Prompt:

# Now I am giving the code of Armstrong number improve it using functions

Code:



```python
def is_armstrong_number(number: int) -> bool:
    """Return True if number is an Armstrong (narcissistic) number."""
    if number < 0:
        return False  # conventional definition excludes negatives

    digits = str(number)
    power = len(digits)
    total = sum(int(d)**power for d in digits)
    return total == number


if __name__ == "__main__":
    try:
        num_to_check = int(input("Enter a number: ").strip())
    except ValueError:
        print("Please enter a valid integer.")
    else:
        if is_armstrong_number(num_to_check):
            print(f"{num_to_check} is an Armstrong number.")
        else:
            print(f"{num_to_check} is not an Armstrong number.")
```

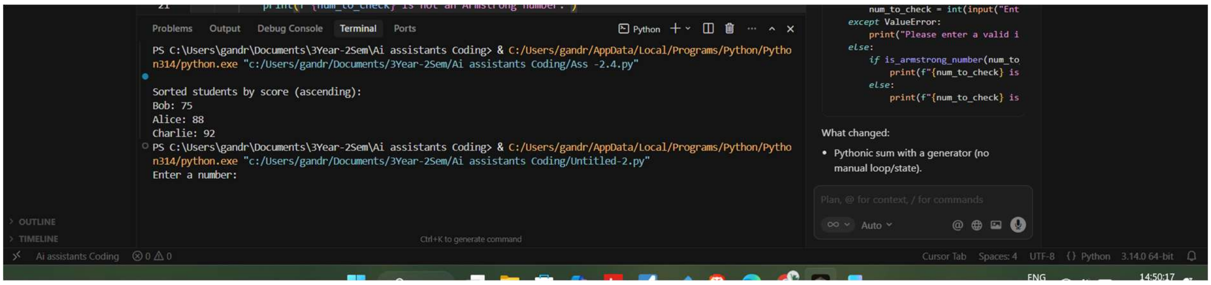
Armstrong number code improve...
Now · Auto

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

Here's a cleaner, safer version you can drop in:

```python
def is_armstrong_number(number: int)
    """Return True if number is an Ar
    if number < 0:
        return False  # conventional

    digits = str(number)
    power = len(digits)
    total = sum(int(d)**power for d i
    return total == number


if __name__ == "__main__":
    try:
        num_to_check = int(input("Ent
    except ValueError:
        print("Please enter a valid i
```

Output:


Problems Output Debug Console Terminal Ports
Python
OUTLINE
TIMELINE
Ctrl+K to generate command
Ai assistants Coding
Cursor Tab Spaces: 4 UTF-8 Python 3.14.0 64-bit
Plan, @ for context, / for commands
Auto
ENG
14:50:17

```
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Pytho
n314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Ass -2.4.py"

Sorted students by score (ascending):
Bob: 75
Alice: 88
Charlie: 92
PS C:\Users\gandr\Documents\3Year-2Sem\Ai assistants Coding> & C:/Users/gandr/AppData/Local/Programs/Python/Pytho
n314/python.exe "c:/Users/gandr/Documents/3Year-2Sem/Ai assistants Coding/Untitled-2.py"
Enter a number:
```

```python
        num_to_check = int(input("Ent
    except ValueError:
        print("Please enter a valid i
    else:
        if is_armstrong_number(num_to
            print(f"{num_to_check} is
        else:
            print(f"{num_to_check} is
```

What changed:

- Pythonic sum with a generator (no manual loop/state).

Justification:

In this task, an Armstrong number program was first generated using Gemini. Although the program worked correctly, it was not fully optimized. Cursor AI was then used to improve the code by introducing functions and list comprehension. The optimized version was more readable and reusable. This shows how Cursor AI helps in refining and optimizing existing code.