AI ASSISTANT CODING - ASSIGNMENT-02

Name: Valaboju Harshavardhanu

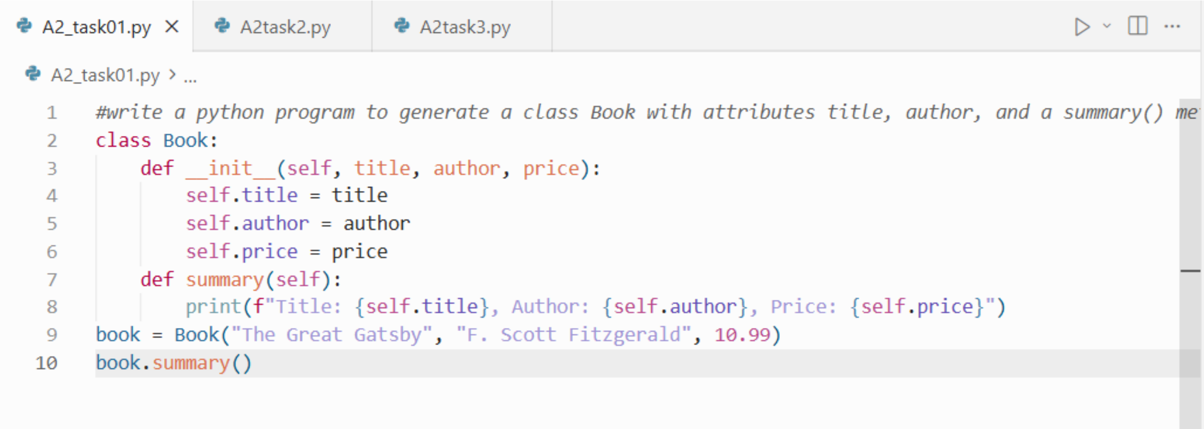
Roll No: 2303A52354 **Batch:** 41

Task 1: Book Class Generation

Prompt:

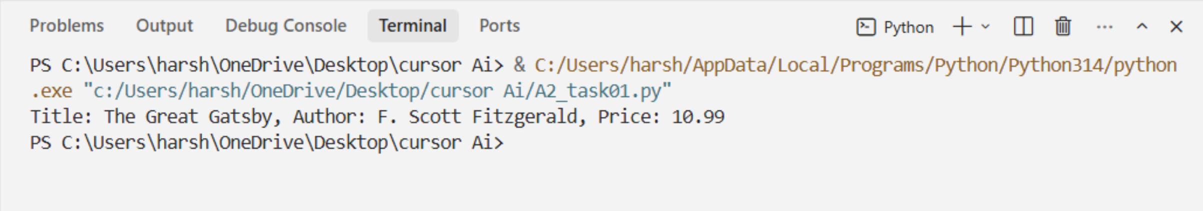
#write a python program to generate a class Book with attributes title, author, and a summary() method that prints the title, author, and price of the book.

Code:



```python
#write a python program to generate a class Book with attributes title, author, and a summary() me
class Book:
    def __init__(self, title, author, price):
        self.title = title
        self.author = author
        self.price = price
    def summary(self):
        print(f"Title: {self.title}, Author: {self.author}, Price: {self.price}")
book = Book("The Great Gatsby", "F. Scott Fitzgerald", 10.99)
book.summary()
```

Output:



```
PS C:\Users\harsh\OneDrive\Desktop\cursor Ai> & C:/Users/harsh/AppData/Local/Programs/Python/Python314/python
.exe "c:/Users/harsh/OneDrive/Desktop/cursor Ai/A2_task01.py"
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Price: 10.99
PS C:\Users\harsh\OneDrive\Desktop\cursor Ai>
```

Justification:

In this task, I used Cursor AI to generate Python code for printing summary method from a Class Book with the attributes of a class. I understood that how the Cursor AI is suggesting the prompt by the prior knowledge. I executed the code, and verified that the output matched the correct to given scenario or not.

Task 2: Sorting Dictionaries with AI

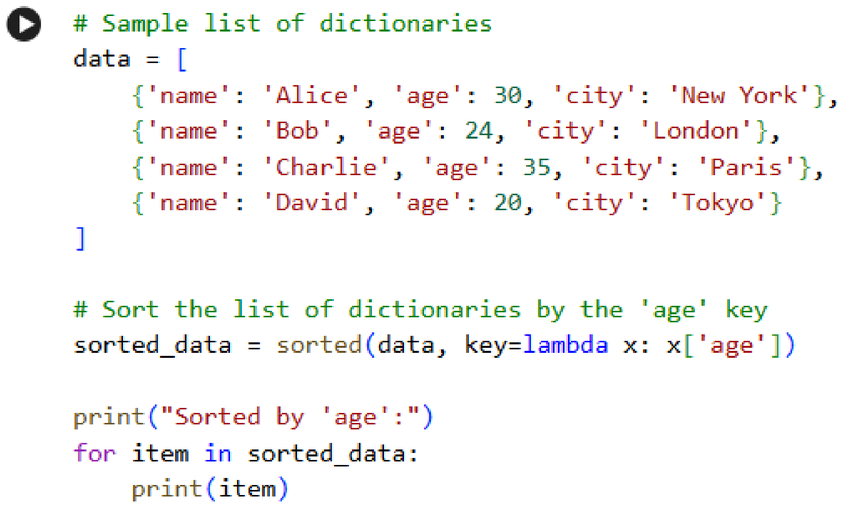
Prompt:

Gemini: #generate a code that sorts a list of dictionaries by a key value age

Cursor AI: #generate a code that sorts a list of dictionaries by a key value age

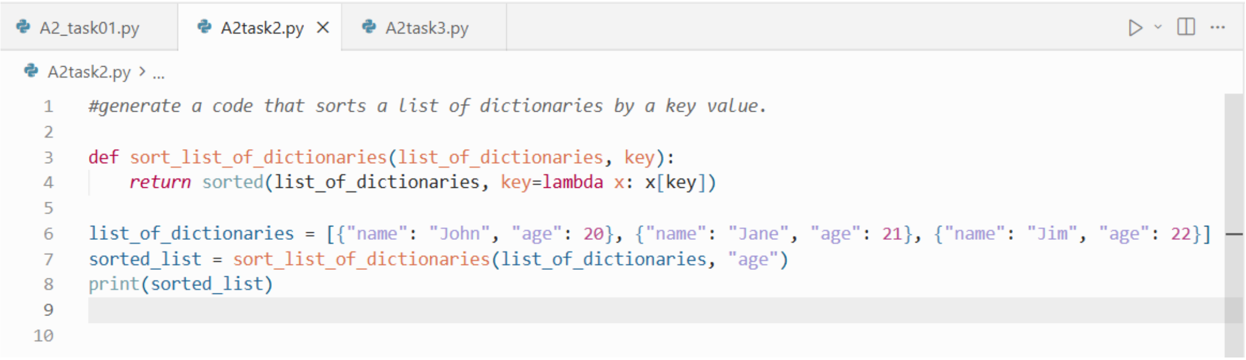
Code:

Gemini



```
# Sample list of dictionaries
data = [
    {'name': 'Alice', 'age': 30, 'city': 'New York'},
    {'name': 'Bob', 'age': 24, 'city': 'London'},
    {'name': 'Charlie', 'age': 35, 'city': 'Paris'},
    {'name': 'David', 'age': 20, 'city': 'Tokyo'}
]

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

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

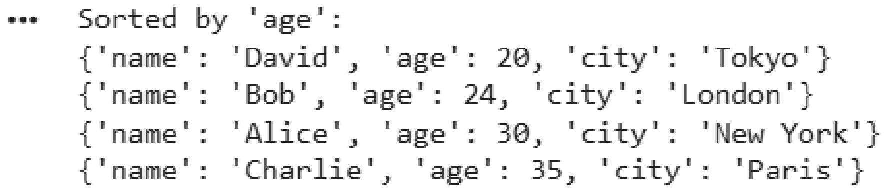
Cursor AI



```
#generate a code that sorts a list of dictionaries by a key value.

def sort_list_of_dictionaries(list_of_dictionaries, key):
    return sorted(list_of_dictionaries, key=lambda x: x[key])

list_of_dictionaries = [{"name": "John", "age": 20}, {"name": "Jane", "age": 21}, {"name": "Jim", "age": 22}]
sorted_list = sort_list_of_dictionaries(list_of_dictionaries, "age")
print(sorted_list)
```

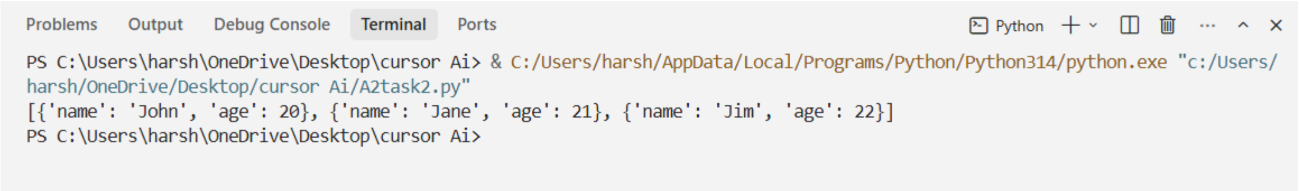
Output:

Gemini



```
•••  Sorted by 'age':
     {'name': 'David', 'age': 20, 'city': 'Tokyo'}
     {'name': 'Bob', 'age': 24, 'city': 'London'}
     {'name': 'Alice', 'age': 30, 'city': 'New York'}
     {'name': 'Charlie', 'age': 35, 'city': 'Paris'}
```

Cursor AI



```
PS C:\Users\harsh\OneDrive\Desktop\cursor Ai> & C:/Users/harsh/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/
harsh/OneDrive/Desktop/cursor Ai/A2task2.py"
[{'name': 'John', 'age': 20}, {'name': 'Jane', 'age': 21}, {'name': 'Jim', 'age': 22}]
PS C:\Users\harsh\OneDrive\Desktop\cursor Ai>
```

Justification:

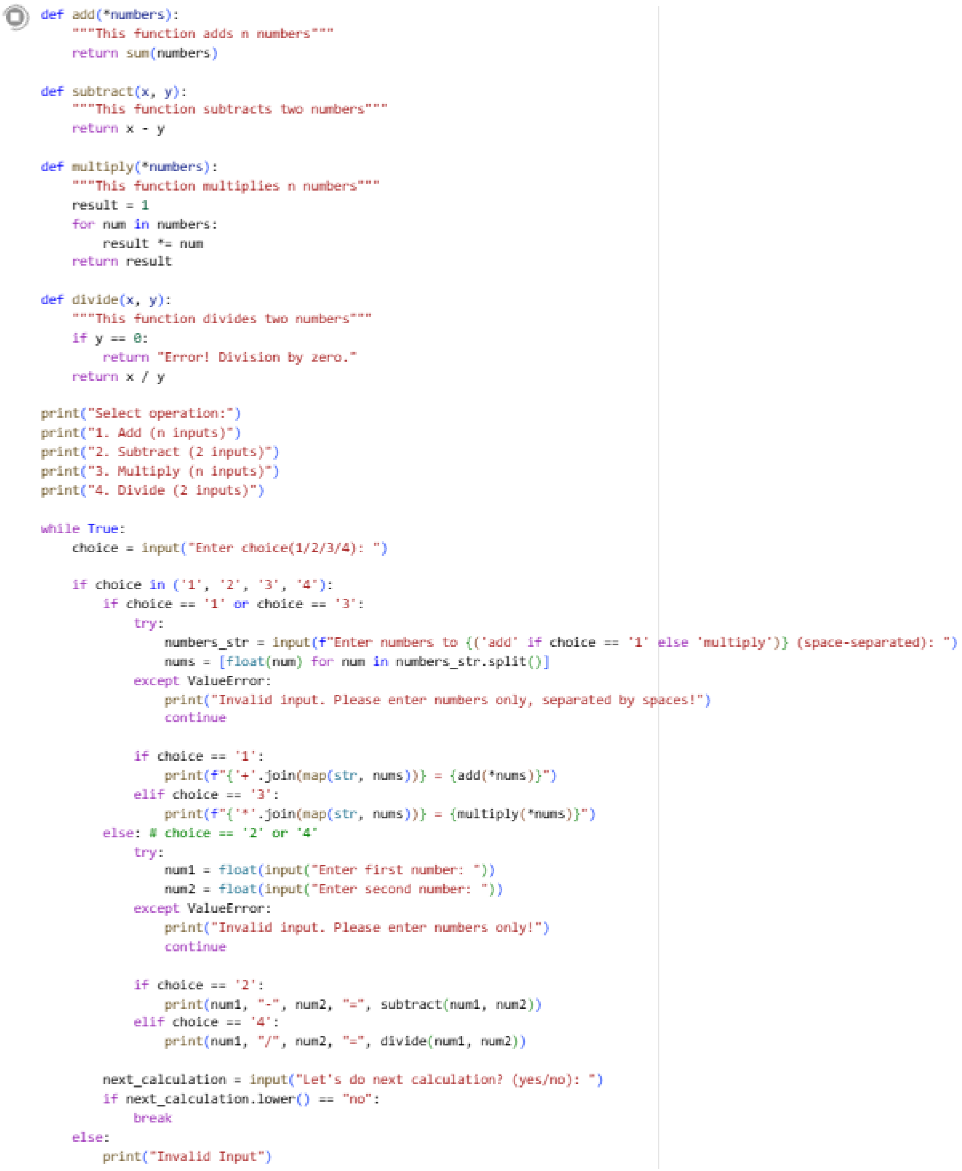
In this task, we have compared the Gemini and Cursor AI for generating the code that sorts the dictionaries by the key value age. And we have compared the clarity and performance of both. As I observed that the Gemini gave good clarity and performance with the code and output which is clear and easy to understand while the cursor AI also good but Gemini is clear comparing with the Cursor AI.

Task 3: Calculator Using Functions

Prompt:

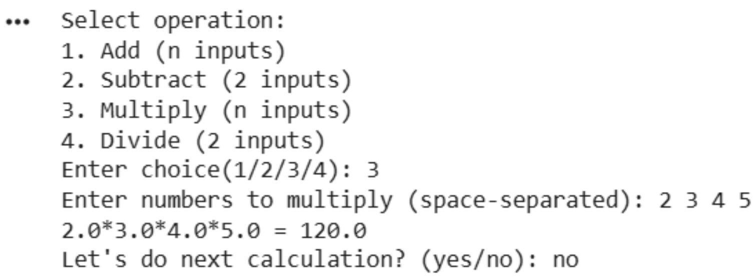
#write a python program to create a basic calculator module with functions for addition, subtraction, multiplication, division. input for Addition and Multiplication are n inputs.

Code:



```python
def add(*numbers):
    """This function adds n numbers"""
    return sum(numbers)

def subtract(x, y):
    """This function subtracts two numbers"""
    return x - y

def multiply(*numbers):
    """This function multiplies n numbers"""
    result = 1
    for num in numbers:
        result *= num
    return result

def divide(x, y):
    """This function divides two numbers"""
    if y == 0:
        return "Error! Division by zero."
    return x / y

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

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

    if choice in ('1', '2', '3', '4'):
        if choice == '1' or choice == '3':
            try:
                numbers_str = input(f"Enter numbers to {('add' if choice == '1' else 'multiply')} (space-separated): ")
                nums = [float(num) for num in numbers_str.split()]
            except ValueError:
                print("Invalid input. Please enter numbers only, separated by spaces!")
                continue

            if choice == '1':
                print(f"{'+'.join(map(str, nums))} = {add(*nums)}")
            elif choice == '3':
                print(f"{'*'.join(map(str, nums))} = {multiply(*nums)}")
        else: # choice == '2' or '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 == '2':
                print(num1, "-", num2, "=", subtract(num1, num2))
            elif choice == '4':
                print(num1, "/", num2, "=", divide(num1, num2))

        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation.lower() == "no":
            break
    else:
        print("Invalid Input")
```

Output:



```
Select operation:
1. Add (n inputs)
2. Subtract (2 inputs)
3. Multiply (n inputs)
4. Divide (2 inputs)
Enter choice(1/2/3/4): 3
Enter numbers to multiply (space-separated): 2 3 4 5
2.0*3.0*4.0*5.0 = 120.0
Let's do next calculation? (yes/no): no
```

Justification:

In this task, using the Gemini we have generated the basic calculator module which is included with the four basic methods of a calculator addition, subtraction, multiplication, division. The user have to choose the operation to perform based on the users input the operation will be performed. This shows how the AI is performing the task efficiently with the single user give prompt.

Task 4: Armstrong Number Optimization

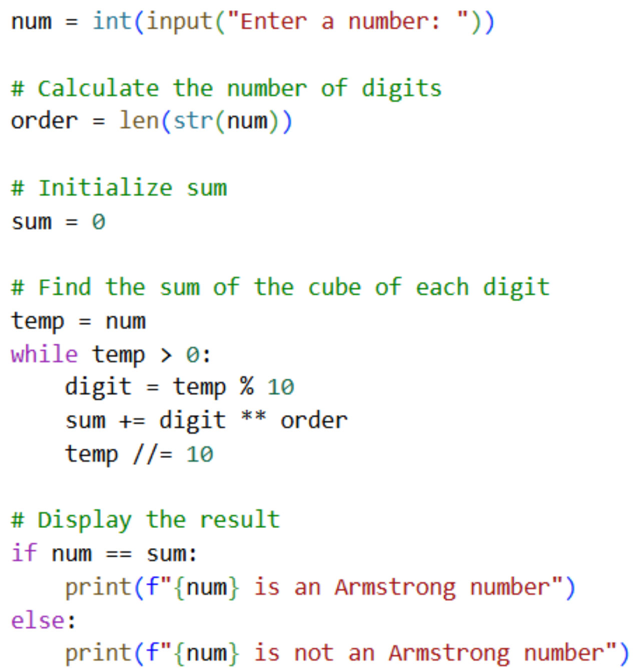
Prompt:

Gemini: #write a code for printing a Fibonacci series up to n terms without using a functions.

Cursor AI: # optimized version of Fibonacci series up to n terms using functions

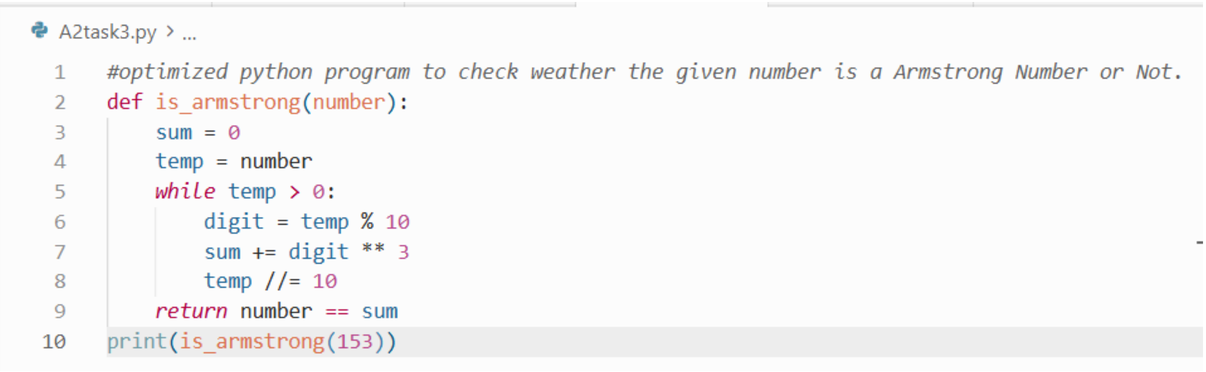
Code:

Gemini



```
num = int(input("Enter a number: "))

# Calculate the number of digits
order = len(str(num))

# Initialize sum
sum = 0

# Find the sum of the cube of each digit
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** order
    temp //= 10

# Display the result
if num == sum:
    print(f"{num} is an Armstrong number")
else:
    print(f"{num} is not an Armstrong number")
```

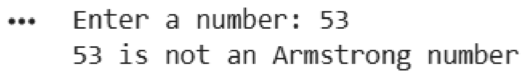
Cursor AI



```
#optimized python program to check weather the given number is a Armstrong Number or Not.
def is_armstrong(number):
    sum = 0
    temp = number
    while temp > 0:
        digit = temp % 10
        sum += digit ** 3
        temp //= 10
    return number == sum
print(is_armstrong(153))
```

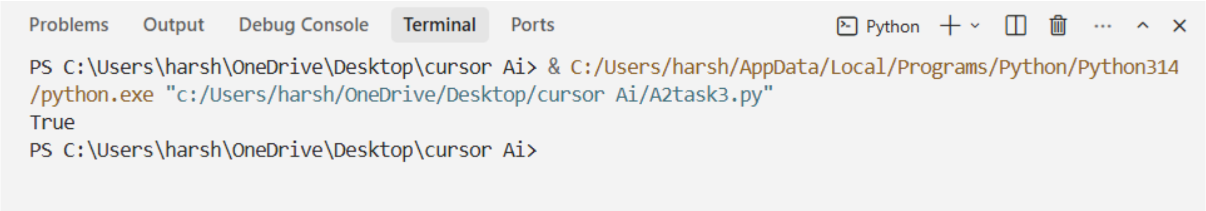
Output:

Gemini



```
Enter a number: 53
53 is not an Armstrong number
```

Cursor AI



```
PS C:\Users\harsh\OneDrive\Desktop\cursor Ai> & C:/Users/harsh/AppData/Local/Programs/Python/Python314
/python.exe "c:/Users/harsh/OneDrive/Desktop/cursor Ai/A2task3.py"
True
PS C:\Users\harsh\OneDrive\Desktop\cursor Ai>
```

Justification:

In this task, we have generated the code to check weather the given number is a Armstrong number or not. First we have generated the code in Gemini and After we improved in Cursor AI. Comparing both the Gemini is a bit more time complexity and in a cursor AI as we have used the prompt which makes the code optimized version and it is understanded by anyone. Even though the outputs are same but the Cursor AI code is more Efficient.