LAB_ASSIGNMENT-2.4

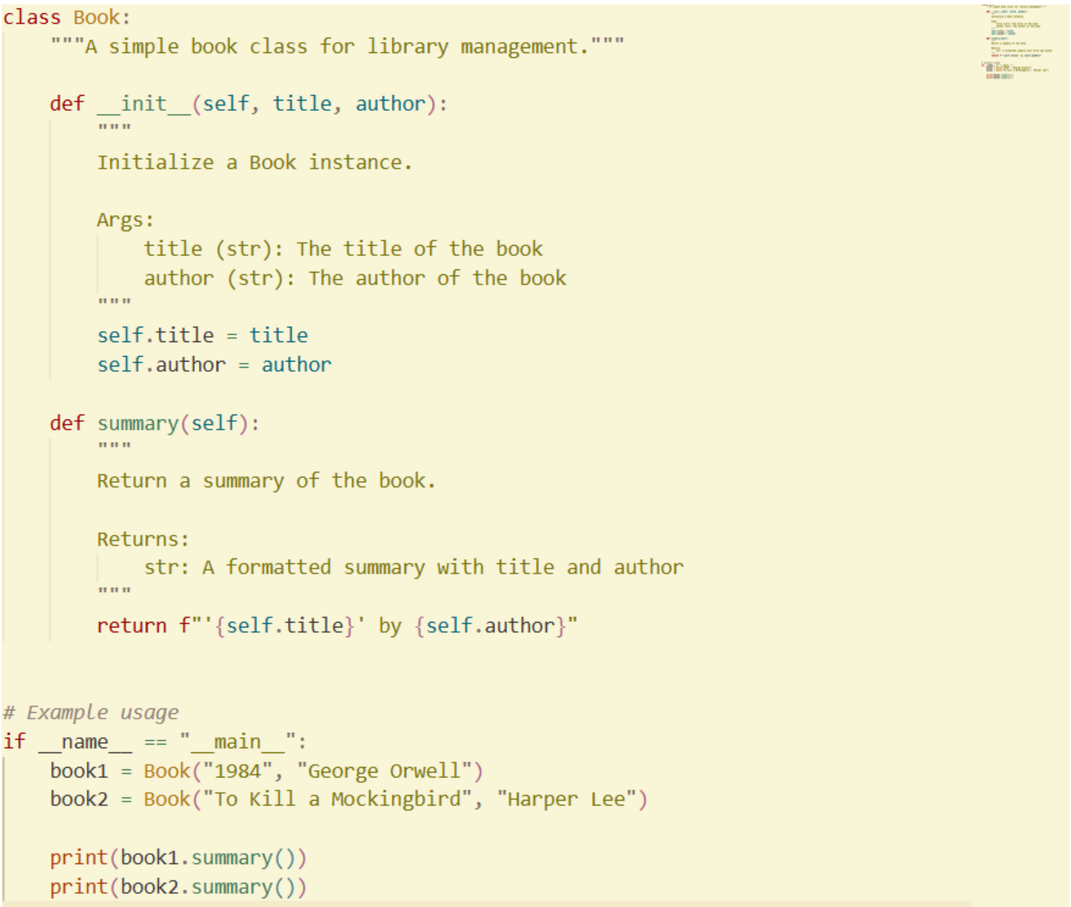
NAME: Vanga Manikanta Varaprasad

ROLL NO: 2303A52130

BATCH: 44

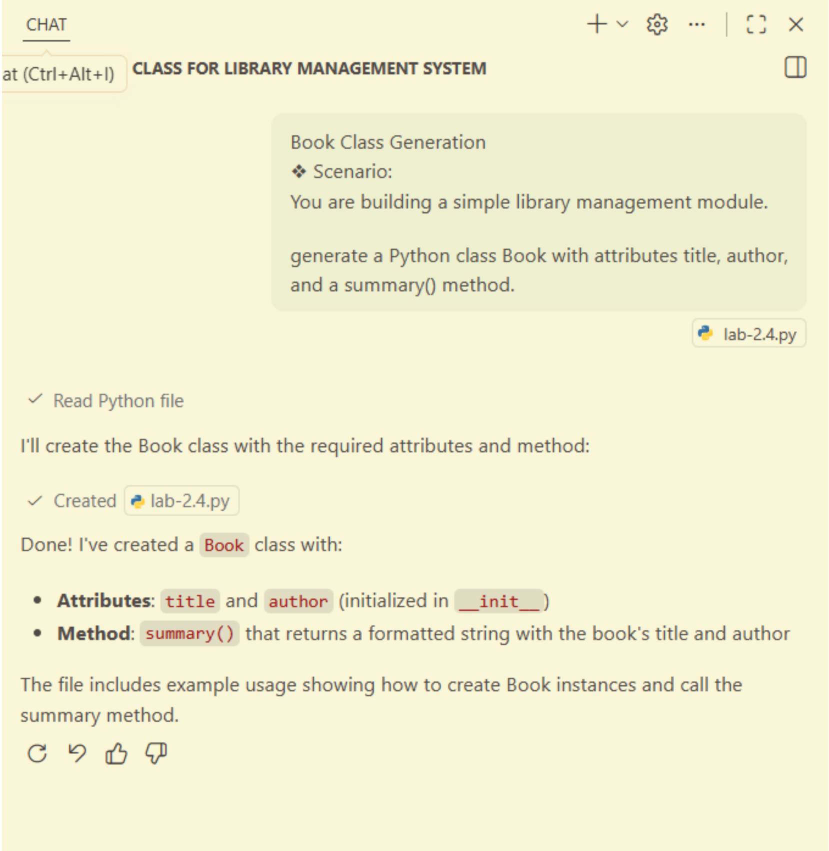
TASK-1

CODE:



```
class Book:
    """A simple book class for library management."""

    def __init__(self, title, author):
        """
        Initialize a Book instance.

        Args:
            title (str): The title of the book
            author (str): The author of the book
        """
        self.title = title
        self.author = author

    def summary(self):
        """
        Return a summary of the book.

        Returns:
            str: A formatted summary with title and author
        """
        return f"'{self.title}' by {self.author}"


# Example usage
if __name__ == "__main__":
    book1 = Book("1984", "George Orwell")
    book2 = Book("To Kill a Mockingbird", "Harper Lee")

    print(book1.summary())
    print(book2.summary())
```

CHAT:



CLASS FOR LIBRARY MANAGEMENT SYSTEM

Book Class Generation
❖ Scenario:
You are building a simple library management module.

generate a Python class Book with attributes title, author, and a summary() method.

✓ Read Python file

I'll create the Book class with the required attributes and method:

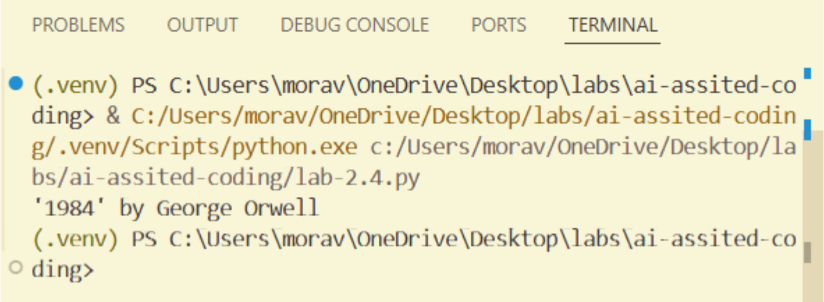
✓ Created lab-2.4.py

Done! I've created a `Book` class with:

- **Attributes**: `title` and `author` (initialized in `__init__`)
- **Method**: `summary()` that returns a formatted string with the book's title and author

The file includes example usage showing how to create Book instances and call the summary method.

OUTPUT:

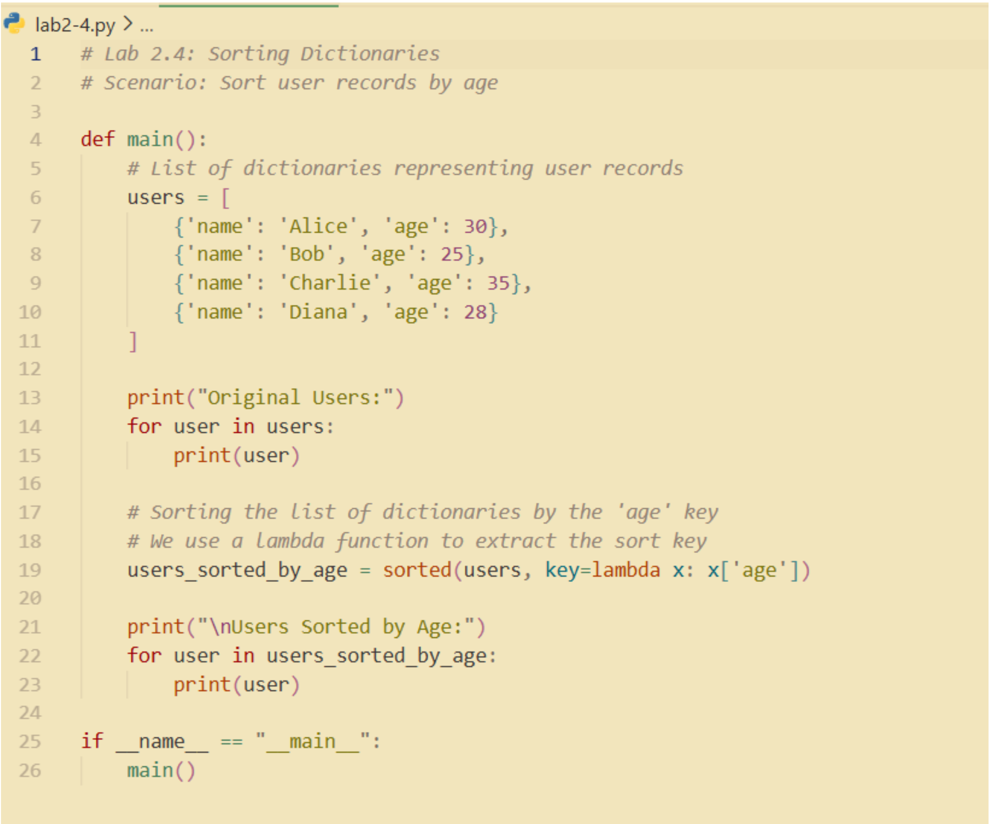


```
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:/Users/morav/OneDrive/Desktop/labs/ai-assited-coding/.venv/Scripts/python.exe c:/Users/morav/OneDrive/Desktop/labs/ai-assited-coding/lab-2.4.py
'1984' by George Orwell
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding>
```

TASK-2

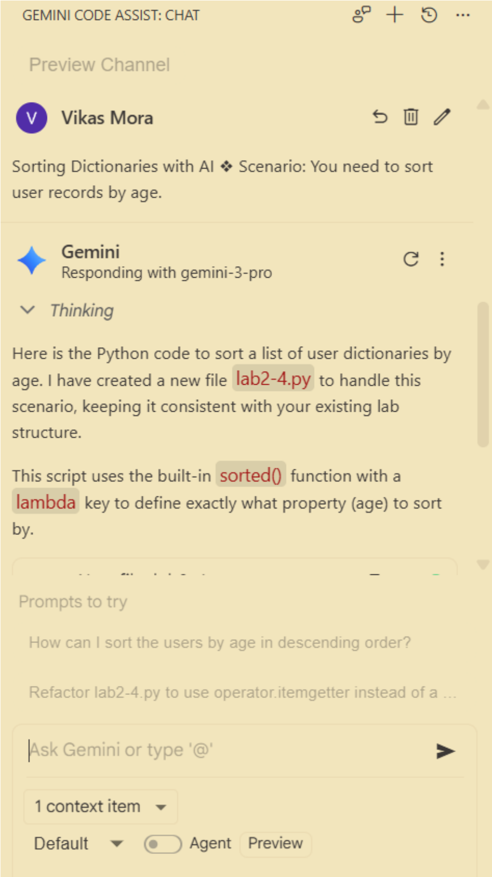
→Using Gemini

CODE:



```python
# Lab 2.4: Sorting Dictionaries
# Scenario: Sort user records by age

def main():
    # List of dictionaries representing user records
    users = [
        {'name': 'Alice', 'age': 30},
        {'name': 'Bob', 'age': 25},
        {'name': 'Charlie', 'age': 35},
        {'name': 'Diana', 'age': 28}
    ]

    print("Original Users:")
    for user in users:
        print(user)

    # Sorting the list of dictionaries by the 'age' key
    # We use a lambda function to extract the sort key
    users_sorted_by_age = sorted(users, key=lambda x: x['age'])

    print("\nUsers Sorted by Age:")
    for user in users_sorted_by_age:
        print(user)

if __name__ == "__main__":
    main()
```

CHAT &OUTPUT:



Vikas Mora

Sorting Dictionaries with AI ❖ Scenario: You need to sort user records by age.

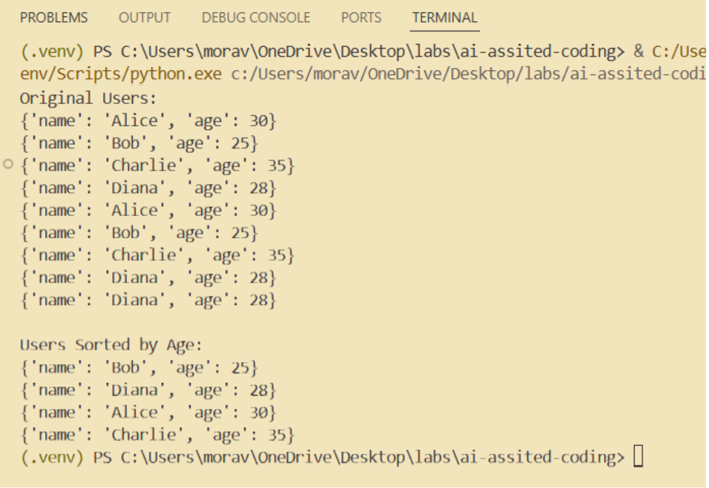
Gemini
Responding with gemini-3-pro

Thinking

Here is the Python code to sort a list of user dictionaries by age. I have created a new file `lab2-4.py` to handle this scenario, keeping it consistent with your existing lab structure.

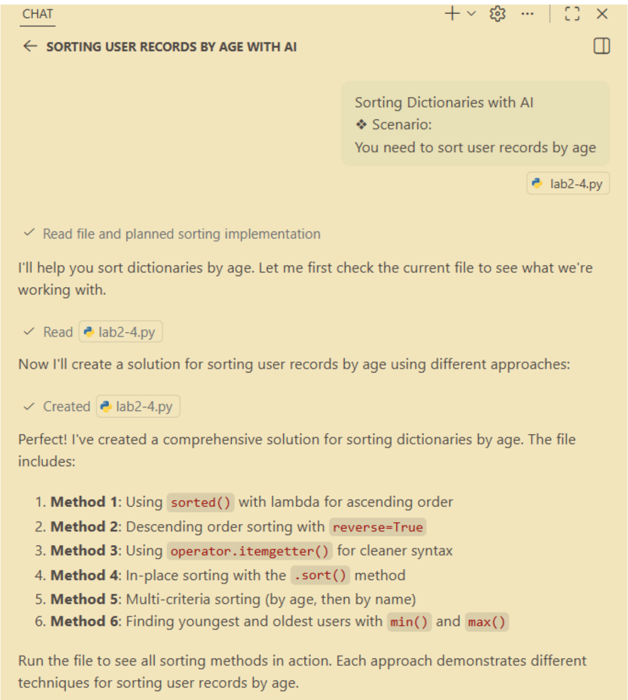
This script uses the built-in `sorted()` function with a `lambda` key to define exactly what property (age) to sort by.

Prompts to try
How can I sort the users by age in descending order?
Refactor lab2-4.py to use operator.itemgetter instead of a …
Ask Gemini or type '@'
1 context item
Default
Agent
Preview


PROBLEMS
OUTPUT
DEBUG CONSOLE
PORTS
TERMINAL

```
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:/Use
env/Scripts/python.exe c:/Users/morav/OneDrive/Desktop/labs/ai-assited-codi
Original Users:
{'name': 'Alice', 'age': 30}
{'name': 'Bob', 'age': 25}
{'name': 'Charlie', 'age': 35}
{'name': 'Diana', 'age': 28}
{'name': 'Alice', 'age': 30}
{'name': 'Bob', 'age': 25}
{'name': 'Charlie', 'age': 35}
{'name': 'Diana', 'age': 28}
{'name': 'Diana', 'age': 28}

Users Sorted by Age:
{'name': 'Bob', 'age': 25}
{'name': 'Diana', 'age': 28}
{'name': 'Alice', 'age': 30}
{'name': 'Charlie', 'age': 35}
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> 
```

→USING CURSOR AI



Sorting Dictionaries with AI
❖ Scenario:
You need to sort user records by age

✓ Read file and planned sorting implementation

I'll help you sort dictionaries by age. Let me first check the current file to see what we're working with.

✓ Read lab2-4.py

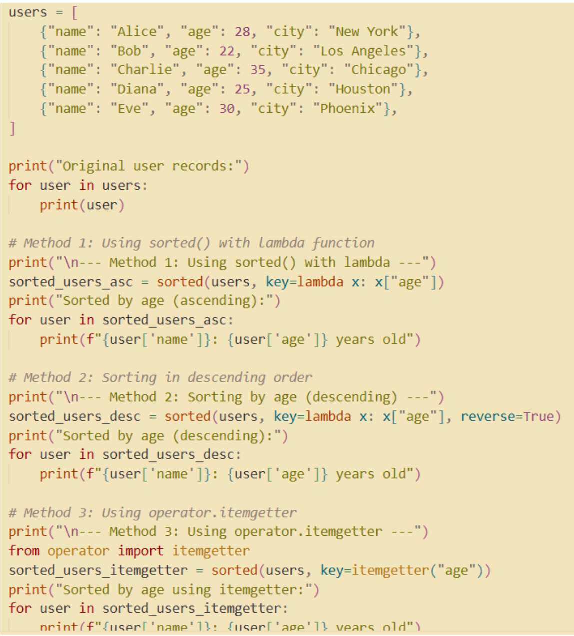
Now I'll create a solution for sorting user records by age using different approaches:

✓ Created lab2-4.py

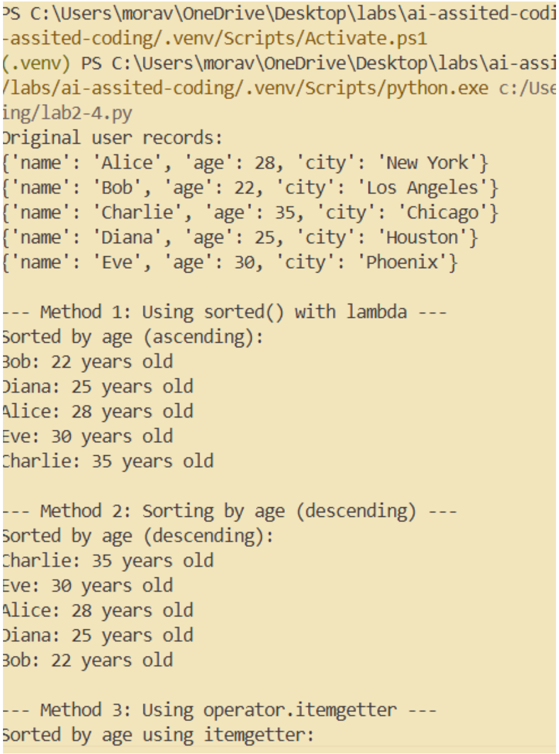
Perfect! I've created a comprehensive solution for sorting dictionaries by age. The file includes:

1. **Method 1**: Using `sorted()` with lambda for ascending order
2. **Method 2**: Descending order sorting with `reverse=True`
3. **Method 3**: Using `operator.itemgetter()` for cleaner syntax
4. **Method 4**: In-place sorting with the `.sort()` method
5. **Method 5**: Multi-criteria sorting (by age, then by name)
6. **Method 6**: Finding youngest and oldest users with `min()` and `max()`

Run the file to see all sorting methods in action. Each approach demonstrates different techniques for sorting user records by age.



```python
users = [
    {"name": "Alice", "age": 28, "city": "New York"},
    {"name": "Bob", "age": 22, "city": "Los Angeles"},
    {"name": "Charlie", "age": 35, "city": "Chicago"},
    {"name": "Diana", "age": 25, "city": "Houston"},
    {"name": "Eve", "age": 30, "city": "Phoenix"},
]

print("Original user records:")
for user in users:
    print(user)

# Method 1: Using sorted() with lambda function
print("\n--- Method 1: Using sorted() with lambda ---")
sorted_users_asc = sorted(users, key=lambda x: x["age"])
print("Sorted by age (ascending):")
for user in sorted_users_asc:
    print(f"{user['name']}: {user['age']} years old")

# Method 2: Sorting in descending order
print("\n--- Method 2: Sorting by age (descending) ---")
sorted_users_desc = sorted(users, key=lambda x: x["age"], reverse=True)
print("Sorted by age (descending):")
for user in sorted_users_desc:
    print(f"{user['name']}: {user['age']} years old")

# Method 3: Using operator.itemgetter
print("\n--- Method 3: Using operator.itemgetter ---")
from operator import itemgetter
sorted_users_itemgetter = sorted(users, key=itemgetter("age"))
print("Sorted by age using itemgetter:")
for user in sorted_users_itemgetter:
    print(f"{user['name']}: {user['age']} years old")
```



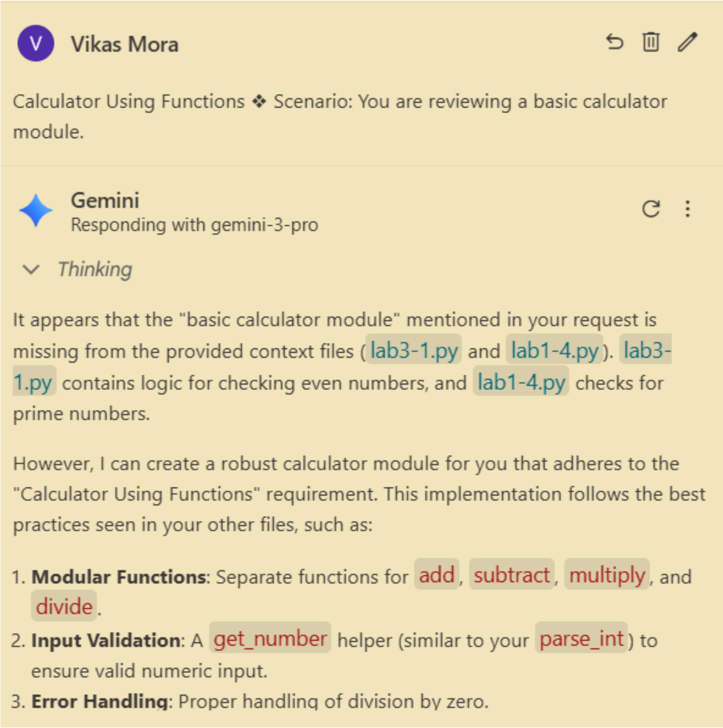
```
PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-codi
-assited-coding/.venv/Scripts/Activate.ps1
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assi
/labs/ai-assited-coding/.venv/Scripts/python.exe c:/Use
ing/lab2-4.py
Original user records:
{'name': 'Alice', 'age': 28, 'city': 'New York'}
{'name': 'Bob', 'age': 22, 'city': 'Los Angeles'}
{'name': 'Charlie', 'age': 35, 'city': 'Chicago'}
{'name': 'Diana', 'age': 25, 'city': 'Houston'}
{'name': 'Eve', 'age': 30, 'city': 'Phoenix'}

--- Method 1: Using sorted() with lambda ---
Sorted by age (ascending):
Bob: 22 years old
Diana: 25 years old
Alice: 28 years old
Eve: 30 years old
Charlie: 35 years old

--- Method 2: Sorting by age (descending) ---
Sorted by age (descending):
Charlie: 35 years old
Eve: 30 years old
Alice: 28 years old
Diana: 25 years old
Bob: 22 years old

--- Method 3: Using operator.itemgetter ---
Sorted by age using itemgetter:
```

COMPARISON:

On comparing both the code I can say that Cursor AI is very advanced and give a optimal code rather than Gemini even though we give same prompt. And the performance of the Cursor is greater than Gemini

TASK-3

CHAT,CODE,AND OUTPUT:



Vikas Mora

Calculator Using Functions ❖ Scenario: You are reviewing a basic calculator module.

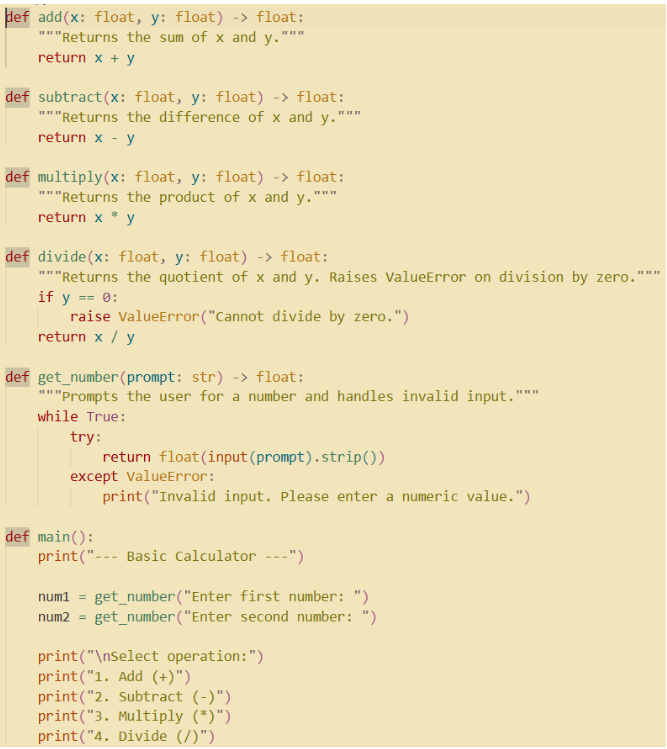
Gemini
Responding with gemini-3-pro

Thinking

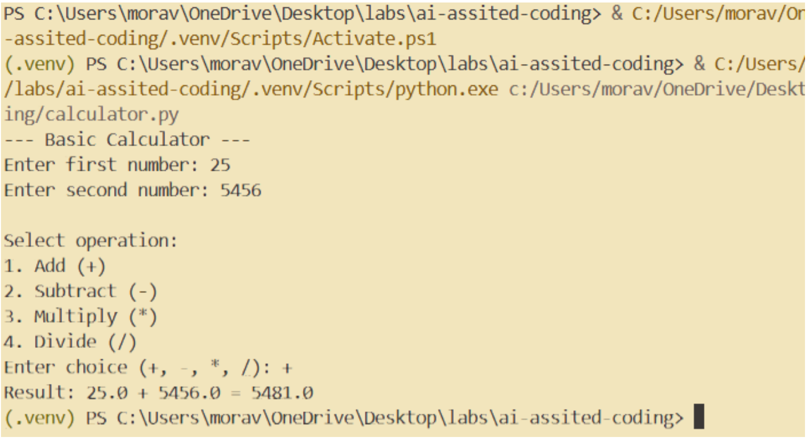
It appears that the "basic calculator module" mentioned in your request is missing from the provided context files (`lab3-1.py` and `lab1-4.py`). `lab3-1.py` contains logic for checking even numbers, and `lab1-4.py` checks for prime numbers.

However, I can create a robust calculator module for you that adheres to the "Calculator Using Functions" requirement. This implementation follows the best practices seen in your other files, such as:

1. **Modular Functions**: Separate functions for `add`, `subtract`, `multiply`, and `divide`.
2. **Input Validation**: A `get_number` helper (similar to your `parse_int`) to ensure valid numeric input.
3. **Error Handling**: Proper handling of division by zero.



```python
def add(x: float, y: float) -> float:
    """Returns the sum of x and y."""
    return x + y

def subtract(x: float, y: float) -> float:
    """Returns the difference of x and y."""
    return x - y

def multiply(x: float, y: float) -> float:
    """Returns the product of x and y."""
    return x * y

def divide(x: float, y: float) -> float:
    """Returns the quotient of x and y. Raises ValueError on division by zero."""
    if y == 0:
        raise ValueError("Cannot divide by zero.")
    return x / y

def get_number(prompt: str) -> float:
    """Prompts the user for a number and handles invalid input."""
    while True:
        try:
            return float(input(prompt).strip())
        except ValueError:
            print("Invalid input. Please enter a numeric value.")

def main():
    print("--- Basic Calculator ---")

    num1 = get_number("Enter first number: ")
    num2 = get_number("Enter second number: ")

    print("\nSelect operation:")
    print("1. Add (+)")
    print("2. Subtract (-)")
    print("3. Multiply (*)")
    print("4. Divide (/)")
```



```
PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:/Users/morav/Or
-assited-coding/.venv/Scripts/Activate.ps1
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:/Users/
/labs/ai-assited-coding/.venv/Scripts/python.exe c:/Users/morav/OneDrive/Deskt
ing/calculator.py
--- Basic Calculator ---
Enter first number: 25
Enter second number: 5456

Select operation:
1. Add (+)
2. Subtract (-)
3. Multiply (*)
4. Divide (/)
Enter choice (+, -, *, /): +
Result: 25.0 + 5456.0 = 5481.0
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> ▌
```

AI Explanation:

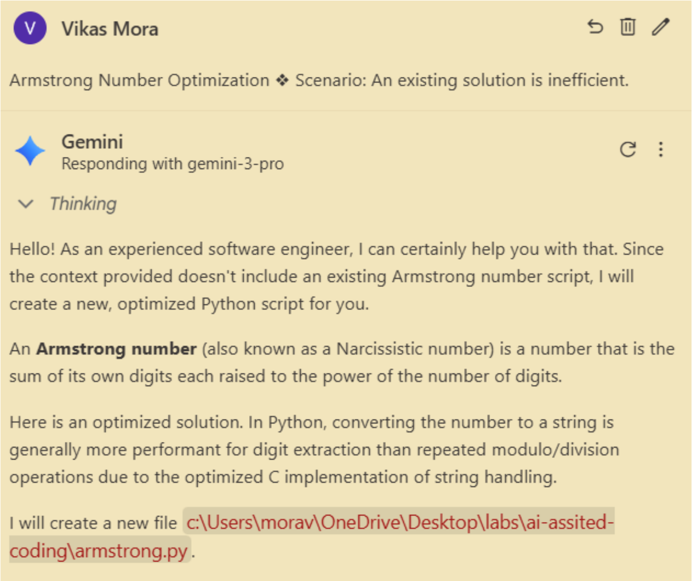
It appears that the "basic calculator module" mentioned in your request is missing from the provided context files contains logic for checking even numbers, and lab1-4.py checks for prime numbers.

However, I can create a robust calculator module for you that adheres to the "Calculator Using Functions" requirement. This implementation follows the best practices seen in your other files, such as:

1. **Modular Functions**: Separate functions for add, subtract, multiply, and divide.
2. **Input Validation**: A get_number helper (similar to your parse_int) to ensure valid numeric input.
3. **Error Handling**: Proper handling of division by zero.
4. **Type Hinting**: For better code clarity and quality.

TASK-4

CHAT,CODE AND OUTPUT(USING GEMINI):



Vikas Mora

Armstrong Number Optimization ❖ Scenario: An existing solution is inefficient.

Gemini
Responding with gemini-3-pro

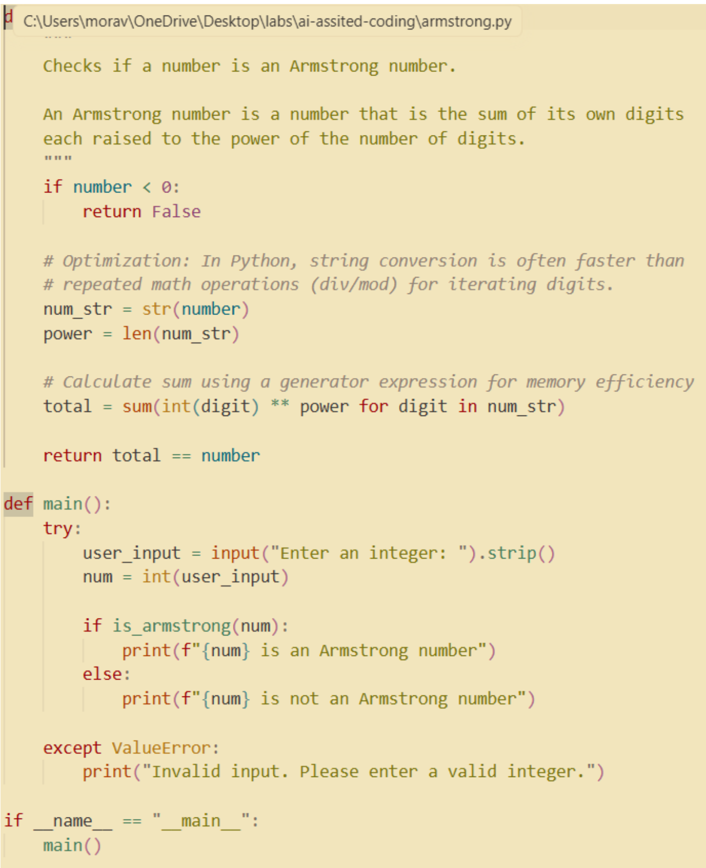
Thinking

Hello! As an experienced software engineer, I can certainly help you with that. Since the context provided doesn't include an existing Armstrong number script, I will create a new, optimized Python script for you.

An **Armstrong number** (also known as a Narcissistic number) is a number that is the sum of its own digits each raised to the power of the number of digits.

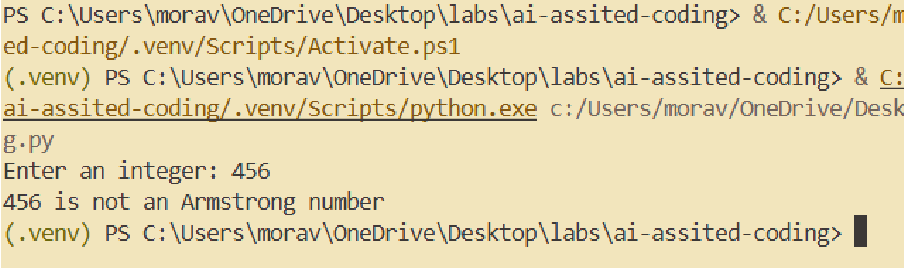
Here is an optimized solution. In Python, converting the number to a string is generally more performant for digit extraction than repeated modulo/division operations due to the optimized C implementation of string handling.

I will create a new file `c:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding\armstrong.py`.



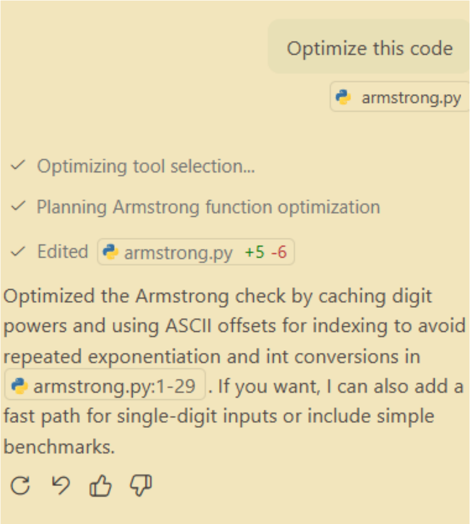
C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding\armstrong.py

```
    Checks if a number is an Armstrong number.

    An Armstrong number is a number that is the sum of its own digits
    each raised to the power of the number of digits.
    """
    if number < 0:
        return False

    # Optimization: In Python, string conversion is often faster than
    # repeated math operations (div/mod) for iterating digits.
    num_str = str(number)
    power = len(num_str)

    # Calculate sum using a generator expression for memory efficiency
    total = sum(int(digit) ** power for digit in num_str)

    return total == number

def main():
    try:
        user_input = input("Enter an integer: ").strip()
        num = int(user_input)

        if is_armstrong(num):
            print(f"{num} is an Armstrong number")
        else:
            print(f"{num} is not an Armstrong number")

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

if __name__ == "__main__":
    main()
```



```
PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:/Users/m
ed-coding/.venv/Scripts/Activate.ps1
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:
ai-assited-coding/.venv/Scripts/python.exe c:/Users/morav/OneDrive/Desk
g.py
Enter an integer: 456
456 is not an Armstrong number
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> 
```

OPTIMIZING THE CODE USING CURSOR:



Optimize this code

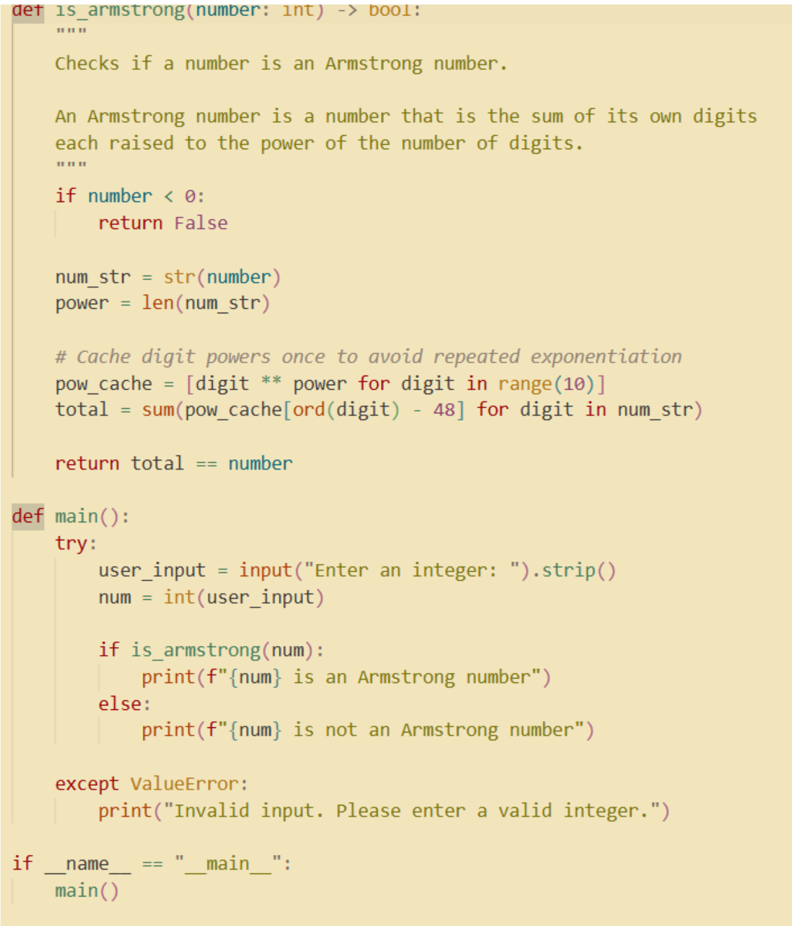
armstrong.py

✓ Optimizing tool selection...

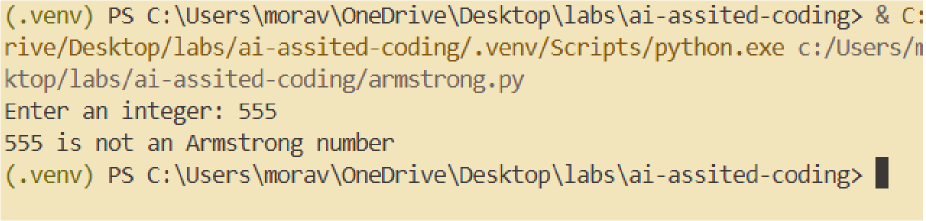
✓ Planning Armstrong function optimization

✓ Edited armstrong.py +5 -6

Optimized the Armstrong check by caching digit powers and using ASCII offsets for indexing to avoid repeated exponentiation and int conversions in armstrong.py:1-29 . If you want, I can also add a fast path for single-digit inputs or include simple benchmarks.



```
def is_armstrong(number: int) -> bool:
    """
    Checks if a number is an Armstrong number.

    An Armstrong number is a number that is the sum of its own digits
    each raised to the power of the number of digits.
    """
    if number < 0:
        return False

    num_str = str(number)
    power = len(num_str)

    # Cache digit powers once to avoid repeated exponentiation
    pow_cache = [digit ** power for digit in range(10)]
    total = sum(pow_cache[ord(digit) - 48] for digit in num_str)

    return total == number

def main():
    try:
        user_input = input("Enter an integer: ").strip()
        num = int(user_input)

        if is_armstrong(num):
            print(f"{num} is an Armstrong number")
        else:
            print(f"{num} is not an Armstrong number")

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

if __name__ == "__main__":
    main()
```



```
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> & C:
rive/Desktop/labs/ai-assited-coding/.venv/Scripts/python.exe c:/Users/m
ktop/labs/ai-assited-coding/armstrong.py
Enter an integer: 555
555 is not an Armstrong number
(.venv) PS C:\Users\morav\OneDrive\Desktop\labs\ai-assited-coding> 
```