ASSIGNMENT 2.4

K. Jashwanth

Batch:42

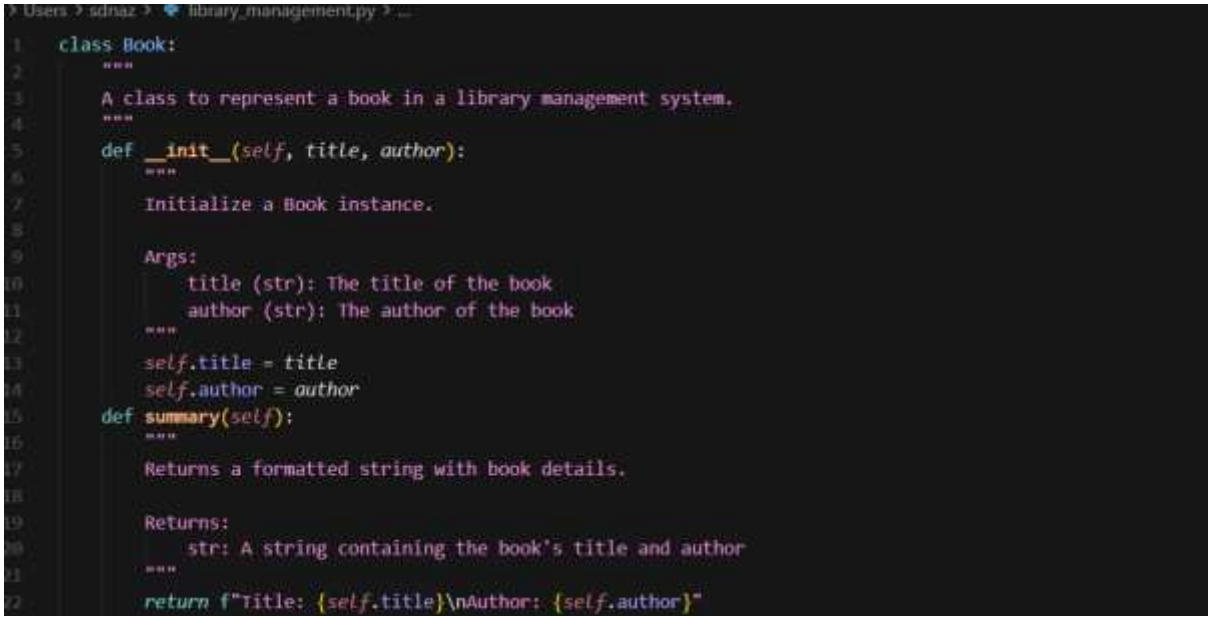
2303A52300

Task 1: Book Class Generation

PROMPT

Generate a Book class with attributes title and author. Include a summary() method that returns book details.

CODE:



```python
class Book:
    """
    A class to represent a book in a library management system.
    """
    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):
        """
        Returns a formatted string with book details.

        Returns:
            str: A string containing the book's title and author
        """
        return f"Title: {self.title}\nAuthor: {self.author}"
```

```python
        """
        return f"Title: {self.title}\nAuthor: {self.author}"
# Example usage
if __name__ == "__main__":
    # Create a book instance
    book1 = Book("The Great Gatsby", "F. Scott Fitzgerald")
    # Display book summary
    print(book1.summary())
    # Create another book
    book2 = Book("1984", "George Orwell")
    print("\n" + book2.summary())
```

OUTPUT:

```
Title: The Great Gatsby
Author: F. Scott Fitzgerald

Title: 1984
Author: George Orwell
PS C:\Program Files\cursor>
```

JUSTIFICATION:

By finishing this task, we can display the book information as required in the question using Cursor AI. The code is easy to understand.

Task 2: Sorting Dictionaries with AI

PROMPT

CURSOR: Create efficient Python code to sort a list of dictionaries by the key "age". Each record contains "name" and "age"

GEMINI: #Generate Python code to sort a list of dictionaries containing user records by the key "age". Each dictionary has the keys: "name" and "age". Find and print names of users with the given age

CURSOR

C: > Users > sdnaz > library_management.py > sort_users_by_age

```
"""
User Record Management System - Sorting by Age
Efficiently sorts a list of user dictionaries by their age.
"""
def sort_users_by_age(users, reverse=False):
    """
    Sort a list of user dictionaries by age.
    Args:
        users: List of dictionaries, each containing at least 'age' key
        reverse: If True, sort in descending order (oldest first)
                 If False, sort in ascending order (youngest first)

    Returns:
        Sorted list of user dictionaries
    """
    return sorted(users, key=lambda user: user['age'], reverse=reverse)
def sort_users_by_age_inplace(users, reverse=False):
    """
    Sort a list of user dictionaries by age in-place (modifies original list).
    More memory efficient for large datasets.

    Args:
        users: List of dictionaries, each containing at least 'age' key
```

```python
def sort_users_by_age_inplace(users, reverse=False):
        reverse: If True, sort in descending order (oldest first)
                 If False, sort in ascending order (youngest first)

    Returns:
        None (modifies list in-place)
    """
    users.sort(key=lambda user: user['age'], reverse=reverse)
# Example usage
if __name__ == "__main__":
    # Sample user records
    users = [
        {"name": "Alice", "age": 30},
        {"name": "Bob", "age": 25},
        {"name": "Charlie", "age": 35},
        {"name": "Diana", "age": 28},
        {"name": "Eve", "age": 32}
    ]

    print("Original users:")
    for user in users:
        print(f"  {user['name']}: {user['age']}")
```

C: › Users › sdnaz › library_management.py › sort_users_by_age

```python

    # Sort by age (ascending - youngest first)
    sorted_users = sort_users_by_age(users)
    print("\nSorted by age (ascending):")
    for user in sorted_users:
        print(f"  {user['name']}: {user['age']}")

    # Sort by age (descending - oldest first)
    sorted_users_desc = sort_users_by_age(users, reverse=True)
    print("\nSorted by age (descending):")
    for user in sorted_users_desc:
        print(f"  {user['name']}: {user['age']}")

    # In-place sorting example
    users_copy = users.copy()
    sort_users_by_age_inplace(users_copy)
    print("\nIn-place sorted (ascending):")
    for user in users_copy:
        print(f"  {user['name']}: {user['age']}")
```

OUTPUT:

```
Original users:
  Alice: 30
  Bob: 25
  Charlie: 35
  Diana: 28
  Eve: 32

Sorted by age (ascending):
  Bob: 25
```

```
  Bob: 25
  Diana: 28
  Alice: 30
  Eve: 32
  Charlie: 35

Sorted by age (descending):
  Charlie: 35
  Eve: 32
  Alice: 30
  Diana: 28
```

```
  In-place sorted (ascending):
    Bob: 25
    Diana: 28
    Alice: 30
    Eve: 32
    Charlie: 35
  PS C:\Program Files\cursor>
```

GEMINI:

```
6]
3s
# List of user records (each user has a name and age)
users = [
    {"name": "Amit", "age": 21},
    {"name": "Riya", "age": 19},
    {"name": "Karan", "age": 22},
    {"name": "Sneha", "age": 21}
]

# Sort the list of dictionaries by age
sorted_users = sorted(users, key=lambda user: user["age"])

# Display sorted records
print("Users sorted by age:")
for user in sorted_users:
    print(user)

# Ask the user to enter an age to search
search_age = int(input("\nEnter age to search: "))

# Find and print names of users with the given age
found = False
for user in sorted_users:
    if user["age"] == search_age:
        print("Name:", user["name"])
        found = True

# If no user found with the given age
if not found:
    print("No user found with this age.")
```

OUTPUT

```
Users sorted by age:
{'name': 'Riya', 'age': 19}
{'name': 'Amit', 'age': 21}
{'name': 'Sneha', 'age': 21}
{'name': 'Karan', 'age': 22}

Enter age to search: 21
Name: Amit
Name: Sneha
```

JUSTIFICATION:

In this task, even though we used two different AI tools, both produced the required output. we can clearly notice a difference in the way the code is written. The clarity of both codes is very good, but when it comes to output presentation, the output from Cursor AI is more neat and easier for anyone to understand.

TASK 3: Calculator Using Functions

PROMPT:

Write a python code for simple calculator with functions addition of n numbers, subtraction of n numbers, multiplication of n numbers, division of n numbers and power of n numbers which chooses the operation on the user input.

```python
def addition(numbers):
    """Addition of n numbers"""
    return sum(numbers)


def subtraction(numbers):
    """Subtraction of n numbers: first - all others"""
    if not numbers:
        return 0
    result = numbers[0]
    for num in numbers[1:]:
        result -= num
    return result


def multiplication(numbers):
    """Multiplication of n numbers"""
    if not numbers:
        return 1
    result = 1
    for num in numbers:
        result *= num
```

```
        return result
    def division(numbers):
        """Division of n numbers: first / all others"
        if not numbers:
            return 0
        result = numbers[0]
        for num in numbers[1:]:
            if num == 0:
                return "Error: Division by zero!"
            result /= num
        return result

    def power(numbers):
        """Power of n numbers: power tower (right-ass
        if not numbers:
            return 1
        if len(numbers) == 1:
            return numbers[0]

        # Right-associative power tower: a^(b^(c^d...
        result = numbers[-1]
        for i in range(len(numbers) - 2, -1, -1):
            result = numbers[i] ** result
```

```
        return result
    def get_numbers():
        """Get n numbers from user"""
        try:
            n = int(input("How many numbers do you want to enter? "))
            if n < 1:
                print("Please enter at least 1 number.")
                return None
            numbers = []
            for i in range(n):
                num = float(input(f"Enter number {i+1}: "))
                numbers.append(num)
            return numbers
        except ValueError:
            print("Invalid input! Please enter valid numbers.")
            return None
    def main():
        print("=" * 50)
        print("Simple Calculator with N Numbers")
        print("=" * 50)
        print("Operations available:")
        print("1. Addition (+)")
        print("2. Subtraction (-)")
```

```
def main():
    print("3. Multiplication (*)")
    print("4. Division (/)")
    print("5. Power (^)")
    print("0. Exit")
    print("=" * 50)

    while True:
        try:
            choice = input("\nEnter your choice (0-5): ").strip()

            if choice == '0':
                print("Thank you for using the calculator!")
                break

            elif choice == '1':
                print("\n--- Addition Operation ---")
                numbers = get_numbers()
                if numbers is not None:
                    result = addition(numbers)
                    print(f"Result: {' + '.join(map(str, numbers))} = {result}")
```

```
            elif choice == '2':
                print("\n--- Subtraction Operation ---")
                numbers = get_numbers()
                if numbers is not None:
                    result = subtraction(numbers)
                    print(f"Result: {' - '.join(map(str, numbers))} = {result}")
            elif choice == '3':
                print("\n--- Multiplication Operation ---")
                numbers = get_numbers()
                if numbers is not None:
                    result = multiplication(numbers)
                    print(f"Result: {' * '.join(map(str, numbers))} = {result}")
            elif choice == '4':
                print("\n--- Division Operation ---")
                numbers = get_numbers()
                if numbers is not None:
                    result = division(numbers)
                    if isinstance(result, str):
                        print(result)
                    else:
                        print(f"Result: {' / '.join(map(str, numbers))} = {result}")
```

```
            elif choice == '5':
                print("\n--- Power Operation (Power Tower) ---")
                numbers = get_numbers()
                if numbers is not None:
                    result = power(numbers)
                    print(f"Result: {'^'.join(map(str, numbers))} = {result}")
            else:
                print("Invalid choice! Please enter a number between 0-5.")
        except KeyboardInterrupt:
            print("\n\nProgram interrupted. Goodbye!")
            break
        except Exception as e:
            print(f"An error occurred: {e}")

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

OUTPUT

```
==================================================
Simple Calculator with N Numbers
==================================================
Operations available:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
5. Power (^)
0. Exit
==================================================
```

```
Enter your choice (0-5): 4

--- Division Operation ---
How many numbers do you want to enter? 2
Enter number 1: 45
Enter number 2: 2
Result: 45.0 / 2.0 = 22.5

Enter your choice (0-5): 0
Thank you for using the calculator!
```

JUSTIFICATION:

By finishing this task the simple basic calculator, The program is easy to understand because each operation is written in a separate function. It can perform calculations on more than two numbers, so it is more useful than a normal calculator. The program checks for errors like wrong input and division by zero, so it runs safely.

Task 4: Armstrong Number Optimization

PROMPT:

check whether the given number is Armstrong or not

CODE:

GEMINI:

```python
def is_armstrong(number):
    """Checks if a given number is an Armstrong number."""
    # Convert number to string to find the number of digits
    str_number = str(number)
    num_digits = len(str_number)

    sum_of_powers = 0
    for digit_char in str_number:
        digit = int(digit_char)
        sum_of_powers += digit ** num_digits

    return sum_of_powers == number
try:
    num_to_check = int(input("Enter a number to check if it's an Armstrong number: "))
    if is_armstrong(num_to_check):
        print(f"{num_to_check} is an Armstrong number.")
    else:
        print(f"{num_to_check} is not an Armstrong number.")
except ValueError:
    print("Invalid input. Please enter a valid integer.")
```

CURSOR AI:

```python
n=int(input("Enter the number: "))
l=[]
y=n
while n>0:
    f=n%10
    l.append(f)
    n=n//10
x=len(l)
s=0
for i in l:
    s=s+i**x
if s==y:
    print("The number is amstrong")
else:
    print("The number is not amstrong")
```

OUTPUT:

GEMINI:

```
Enter a number to check if it's an Armstrong number: 153
153 is an Armstrong number.
```

CURSOR AI:

```
Enter the number: 153
The number is amstrong
```

JUSTIFICATION:

In this task, we can clearly observe that, the code given by gemini is a bit complex and has more time complexity which makes code less efficient, as we have optimized it in cursor ai,

the code is optimized version with only basic syntax which makes the code easy to understand for anyone. Even though the outputs are same, both the codes given by both gemini and cursor ai are efficient.