

AI Assistant Coding Assignment-2

Name of Student : V. Abhinav Batch : 41
Enrollment No. : 2303A52486

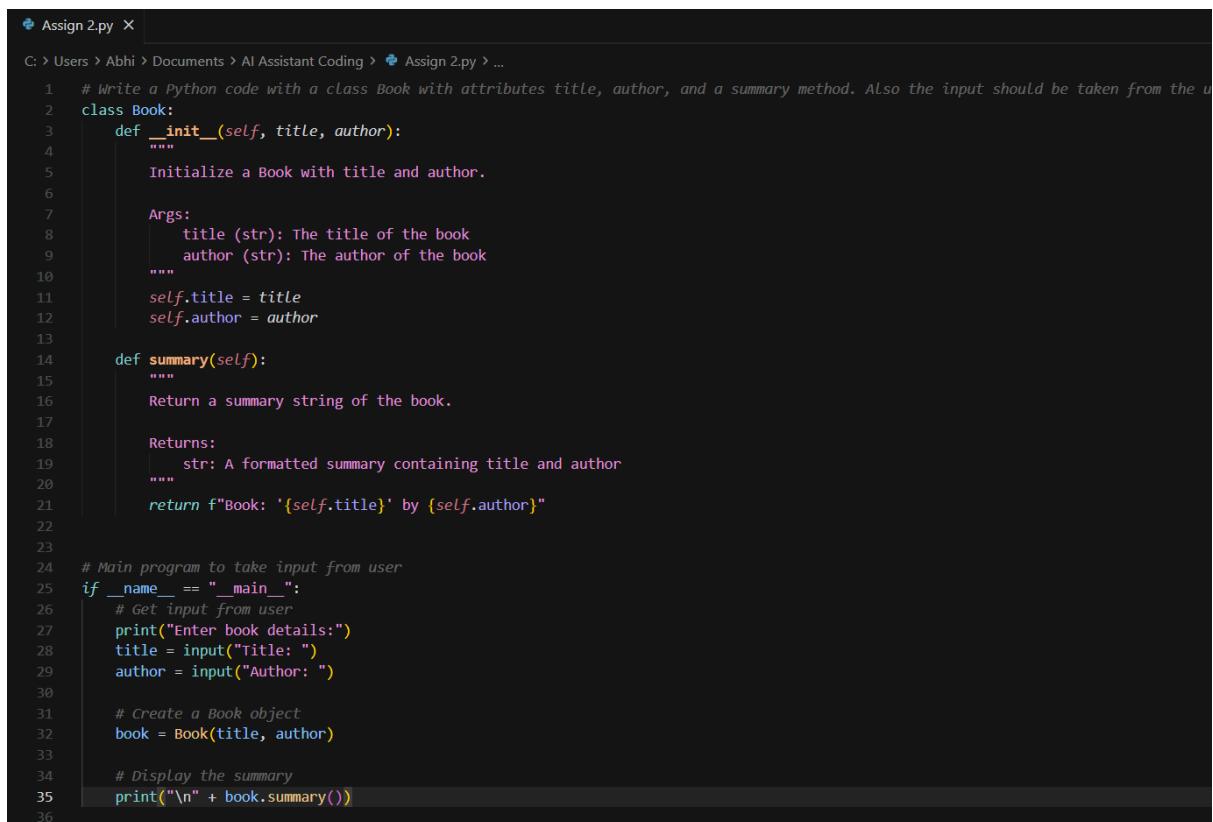
Task 1: Book Class Generation

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

Prompt:

Write a Python code with a class Book with attributes title, author, and a summary method. Also the input should be taken from the user.

Code:



The screenshot shows a code editor window with a dark theme. The file is named 'Assign 2.py'. The code defines a 'Book' class with an __init__ method that initializes the book's title and author. It also defines a summary method that returns a string containing the title and author. The code includes docstrings and comments explaining the purpose and usage of each part. The code editor interface includes a status bar at the bottom showing file paths and line numbers.

```
C:\ > Users > Abhi > Documents > AI Assistant Coding > Assign 2.py > ...
1 # Write a Python code with a class Book with attributes title, author, and a summary method. Also the input should be taken from the user
2 class Book:
3     def __init__(self, title, author):
4         """
5             Initialize a Book with title and author.
6
7             Args:
8                 title (str): The title of the book
9                 author (str): The author of the book
10            """
11            self.title = title
12            self.author = author
13
14     def summary(self):
15         """
16             Return a summary string of the book.
17
18             Returns:
19                 str: A formatted summary containing title and author
20            """
21            return f"Book: '{self.title}' by {self.author}"
22
23
24 # Main program to take input from user
25 if __name__ == "__main__":
26     # Get input from user
27     print("Enter book details:")
28     title = input("Title: ")
29     author = input("Author: ")
30
31     # Create a Book object
32     book = Book(title, author)
33
34     # Display the summary
35     print("\n" + book.summary())
36
```

Output:

```
Problems Output Debug Console Terminal Ports
PS C:\Users\Abhi> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 2.py"
Enter book details:
Title: The Great Gatsby
Author: F. Scott
Book: 'The Great Gatsby' by F. Scott
PS C:\Users\Abhi>
```

Explanation:

In this program a class named *Book* is used to store the book details like title and author. The data is taken from the user and A method called *summary()* displays the book information in a readable format.

Task 2: Sorting Dictionaries with AI

Use Gemini or Cursor AI to generate code that sorts a list of dictionaries by a key.

Prompt:

Write a code that sorts a list of dictionaries with additional useful attributes, sort user records by age.

Code:

```
{
    "name": "Diana",
    "age": 28,
    "city": "Tokyo",
    "email": "diana@example.com",
    "phone": "+81-3-1234-5678",
    "occupation": "Developer",
    "country": "Japan",
    "zip_code": "100-0001",
    "registration_date": "2022-05-08",
    "status": "inactive"
}
]

# Sort by age using sorted() function with a key parameter
sorted_users = sorted(users, key=lambda x: x['age'])

# Display sorted results with all attributes
print("Users sorted by age:")
print("-" * 80)
for user in sorted_users:
    print(f"Name: {user['name']}")
    print(f"Age: {user['age']} years old")
    print(f"Email: {user['email']}")
    print(f"Phone: {user['phone']}")
    print(f"Occupation: {user['occupation']}")
    print(f"Location: {user['city']}, {user['country']} ({user['zip_code']})")
    print(f"Registration Date: {user['registration_date']}")
    print(f"Status: {user['status']}")
    print("-" * 80)

# Example: Sort by other attributes
print("\nUsers sorted by registration date:")
sorted_by_registration = sorted(users, key=lambda x: x['registration_date'])
for user in sorted_by_registration:
    print(f"{user['name']} - Registered: {user['registration_date']}
```

```
users = [
    {
        "name": "Alice",
        "age": 30,
        "city": "New York",
        "email": "alice@example.com",
        "phone": "+1-555-0101",
        "occupation": "Engineer",
        "country": "USA",
        "zip_code": "10001",
        "registration_date": "2020-01-15",
        "status": "active"
    },
    {
        "name": "Bob",
        "age": 25,
        "city": "London",
        "email": "bob@example.com",
        "phone": "+44-20-7946-0958",
        "occupation": "Designer",
        "country": "UK",
        "zip_code": "SW1A 1AA",
        "registration_date": "2021-03-22",
        "status": "active"
    },
    {
        "name": "Charlie",
        "age": 35,
        "city": "Paris",
        "email": "charlie@example.com",
        "phone": "+33-1-42-86-83-26",
        "occupation": "Manager",
        "country": "France",
        "zip_code": "75001",
        "registration_date": "2019-11-10",
        "status": "active"
    }
]
```

Output:

```
Problems Output Debug Console Terminal Ports
PS C:\Users\Abhi> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:\Users\Abhi\PycharmProjects\Python\Day 1\user.py"
Users sorted by age:
-----
Name: Bob
Age: 25 years old
Email: bob@example.com
Phone: +44-20-7946-0958
Occupation: Designer
Location: London, UK (SW1A 1AA)
Registration Date: 2021-03-22
Status: active
-----
Name: Diana
Age: 28 years old
Email: diana@example.com
Phone: +81-3-1234-5678
Occupation: Developer
Location: Tokyo, Japan (100-0001)
Registration Date: 2022-05-08
Status: inactive
-----
Name: Alice
Age: 30 years old
Email: alice@example.com
Phone: +1-555-0101
Occupation: Engineer
Location: New York, USA (10001)
Registration Date: 2020-01-15
Status: active
-----
Name: Charlie
Age: 35 years old
Email: charlie@example.com
Phone: +33-1-42-86-83-26
Occupation: Manager
Location: Paris, France (75001)
Registration Date: 2019-11-10
Status: active
```

Justification:

Here a list of dictionaries stores user information. It sorts users based on age and registration date using the `sorted()` function. Lambda functions are used to specify the sorting key. The sorted user details are then printed.

Task 3: Calculator Using Functions

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

Prompt:

Generate a python code for basic calculator module that takes user input, also asks the user to perform a specific operation like ex: 1: Add; 2: Subtract. Use inline comments for readability or explanation.

Code:

```
def calculator():
    """
    Basic calculator module with user input and operation selection.
    Enhanced with multi-number addition feature.
    """

    # Display operation menu
    print("\n" + "="*40)
    print("    CALCULATOR MENU")
    print("="*40)
    print("1: Add      2: Subtract")
    print("3: Multiply  4: Divide")
    print("5: Modulus   6: Power")
    print("0: Exit")
    print("="*40)

    # Get user's operation choice
    choice = input("Select operation (0-6): ").strip()

    # Exit if user chooses 0
    if choice == '0':
        return False  # Signal to exit

    # Validate choice
    if choice not in ['1', '2', '3', '4', '5', '6']:
        print("X Invalid choice! Please enter 0-6.")
        return True  # Continue loop

    # Special handling for Add operation - check if user wants to add multiple numbers
    if choice == '1':  # Add operation
        add_multiple = input("Do you want to add multiple numbers? (yes/no): ").lower().strip()

        if add_multiple in ['yes', 'y']:
            # Get list of numbers from user
            numbers_input = input("Enter numbers separated by comma or space (e.g., 1,2,3,4 or 1 2 3 4): ").strip()
```

```

def calculator():
    # Dictionary mapping choices to operations and symbols
    operations = {
        '1': (lambda x, y: x + y, '+'),
        '2': (lambda x, y: x - y, '-'),
        '3': (lambda x, y: x * y, '*'),
        '4': (lambda x, y: x / y if y != 0 else None, '/'),
        '5': (lambda x, y: x % y if y != 0 else None, '%'),
        '6': (lambda x, y: x ** y, '^')
    }

    # Get operation function and symbol
    operation_func, symbol = operations[choice]

    # Check for division by zero (for divide and modulus)
    if choice in ['4', '5'] and num2 == 0:
        print("✖ Error: Cannot divide by zero!")
        return True

    # Perform calculation
    result = operation_func(num1, num2)

    # Display result
    print(f"\n✓ Result: {num1} {symbol} {num2} = {result}")
    print("-" * 40)

    return True # Continue Loop

# Main execution
if __name__ == "__main__":
    print("Welcome to the Basic Calculator!")

    # Run calculator in a Loop
    while calculator():
        pass # Continue until calculator() returns False

    print("\n👋 Thank you for using the calculator!")

```

Output:

```

Welcome to the Basic Calculator!
=====
CALCULATOR MENU
=====
1: Add      2: Subtract
3: Multiply 4: Divide
5: Modulus   6: Power
0: Exit
=====
Select operation (0-6): 1
Do you want to add multiple numbers? (yes/no): no
Enter first number: 6
Enter second number: 7

✓ Result: 6.0 + 7.0 = 13.0
=====

=====
CALCULATOR MENU
=====
1: Add      2: Subtract
3: Multiply 4: Divide
5: Modulus   6: Power
0: Exit
=====
Select operation (0-6): 0

👋 Thank you for using the calculator!

```

Justification:

In the above code a menu-based calculator that performs basic mathematical operations. The user selects an operation(0-6) and enters numbers to calculate the result. It includes input validation and handles errors like division by zero. The program keeps running until the user chooses to exit.

Task 4: Armstrong Number Optimization

Generate an Armstrong number program using Gemini, then improve it using Cursor AI.

Prompt:

Generate python code for checking if the number Armstrong number or not, taking input from the user.

Code:

Original:

```
Assign 2.py X

C: > Users > Abhi > Documents > AI Assistant Coding > Assign 2.py > ...
224
225 # Generate python code for checking if the number Armstrong number or not without, taking input from user
226
227 num_str = input("Enter a number: ")
228 num = int(num_str)
229
230 # Count the number of digits
231 num_digits = len(num_str)
232
233 # Calculate sum of digits raised to the power of number of digits
234 sum_power = 0
235 temp = num
236 while temp > 0:
237     digit = temp % 10
238     # calculate digit raised to power of num_digits
239     power = 1
240     for i in range(num_digits):
241         power = power * digit
242     sum_power = sum_power + power
243     temp = temp // 10
244
245 # Check if it's an Armstrong number
246 if num == sum_power:
247     print(f"{num} is an Armstrong number")
248 else:
249     print(f"{num} is not an Armstrong number")
250
```

Improved:

```
# Improve this code and use function so that it can be called when needed.
def is_armstrong(num):
    num_str = str(num)
    num_digits = len(num_str)
    sum_power = 0
    temp = num
    while temp > 0:
        digit = temp % 10
        power = 1
        for i in range(num_digits):
            power = power * digit
        sum_power = sum_power + power
        temp = temp // 10
    return num == sum_power

# Take input from user
num = int(input("Enter a number: "))

# Check if it's an Armstrong number
if is_armstrong(num):
    print(f"{num} is an Armstrong number")
else:
    print(f"{num} is not an Armstrong number")
```

Output:

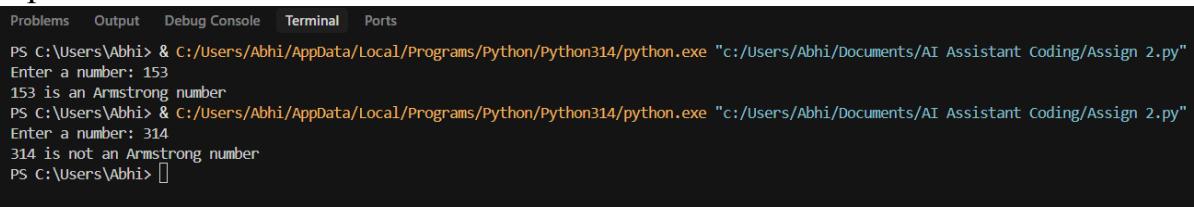
Original:



```
Problems Output Debug Console Terminal Ports

PS C:\Users\Abhi> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 2.py"
Enter a number: 327
327 is not an Armstrong number
PS C:\Users\Abhi> 
```

Improved:



```
Problems Output Debug Console Terminal Ports

PS C:\Users\Abhi> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 2.py"
Enter a number: 153
153 is an Armstrong number
PS C:\Users\Abhi> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 2.py"
Enter a number: 314
314 is not an Armstrong number
PS C:\Users\Abhi> 
```

Justification:

Here both programs check whether a given number is Armstrong or not. The improved code has *is_armstrong* function that can be used over and over again. Both logics are the same, separating each digit of the number and raising it to the power of the total number of digits. All the results are added and compared with the original number. If both are equal, the number is an Armstrong number.