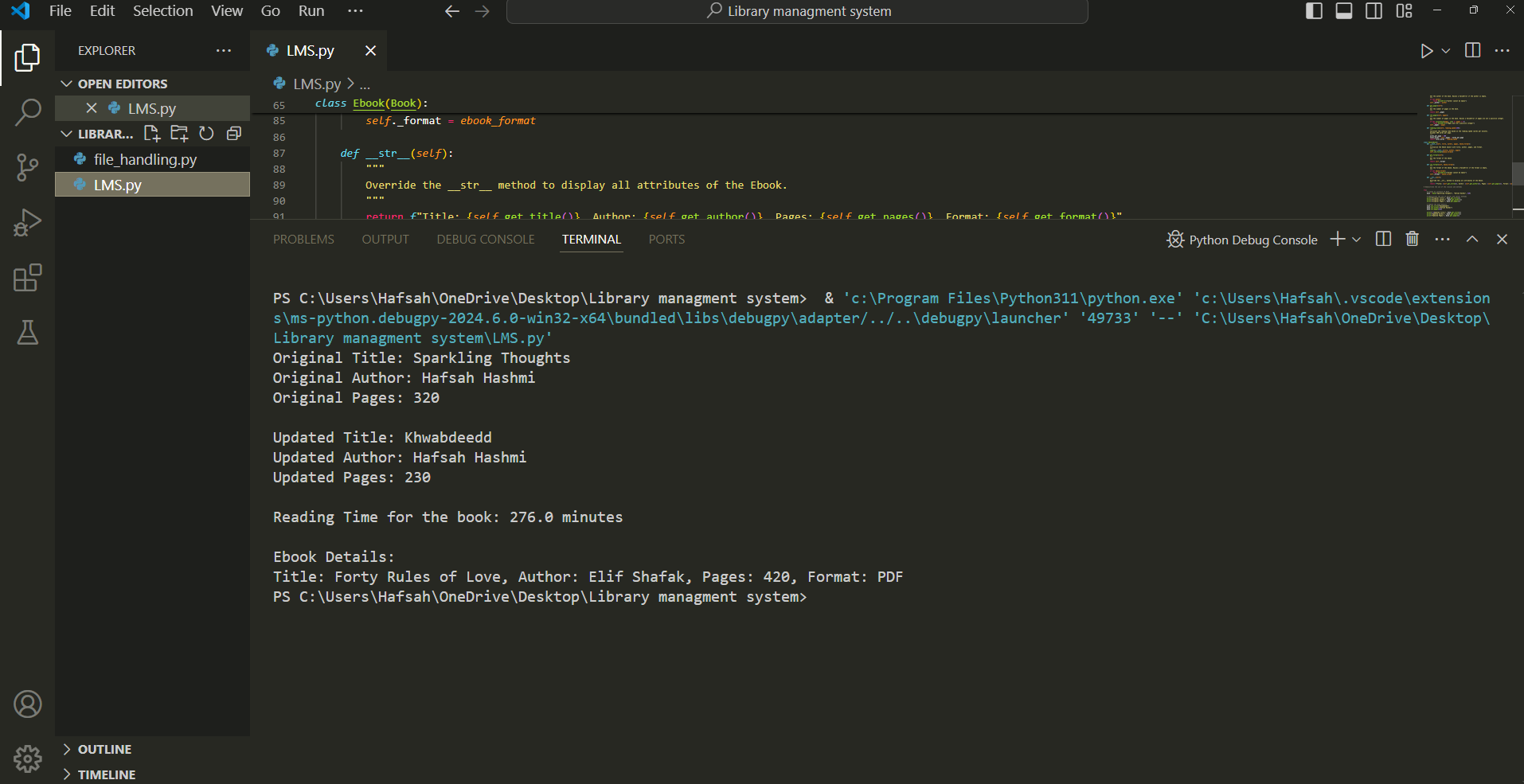
**Week 3**

**Task 1: Output**



Oops concept used in this program:

**1. Classes and Objects:**

* **Class**: A blueprint for creating objects. It defines a set of attributes (variables) and methods (functions) that the objects created from the class can use.
* **Object**: An instance of a class. When a class is defined, objects can be created from it, each with its own set of attributes and methods.

**Example in the Code**:

* Book is a class that defines attributes like title, author, and pages, and methods to get and set these attributes.
* Ebook is another class that inherits from Book and adds an additional attribute called format.

**2. Attributes:**

* Attributes are variables that belong to a class.

**Example in the Code**:

* title, author, and pages are attributes of the Book class.
* format is an additional attribute in the Ebook class.

**3. Methods:**

* Methods are functions that belong to a class. They define the behaviors of the objects created from the class.

**Example in the Code**:

* get\_title, set\_title, get\_author, set\_author, get\_pages, set\_pages, and reading\_time are methods of the Book class.
* get\_format, set\_format, and \_\_str\_\_ are methods of the Ebook class.

**4. Inheritance:**

* Inheritance is a way to create a new class that is based on an existing class. The new class, called a subclass, inherits attributes and methods from the existing class, called a superclass.

**Example in the Code**:

* Ebook is a subclass that inherits from the Book class. This means Ebook has all the attributes and methods of Book, and it can also have additional attributes and methods.

**5. Encapsulation:**

* Encapsulation is the concept of restricting access to certain attributes and methods, making them private, and providing public methods to get and set their values.

**Example in the Code**:

* The attributes \_title, \_author, and \_pages in the Book class are prefixed with an underscore to indicate they are private. Public methods (getters and setters) are provided to access and modify these attributes.

**6. Exception Handling:**

* Exception handling is a way to manage errors that may occur during the execution of a program. It allows the program to continue running or gracefully handle the error.

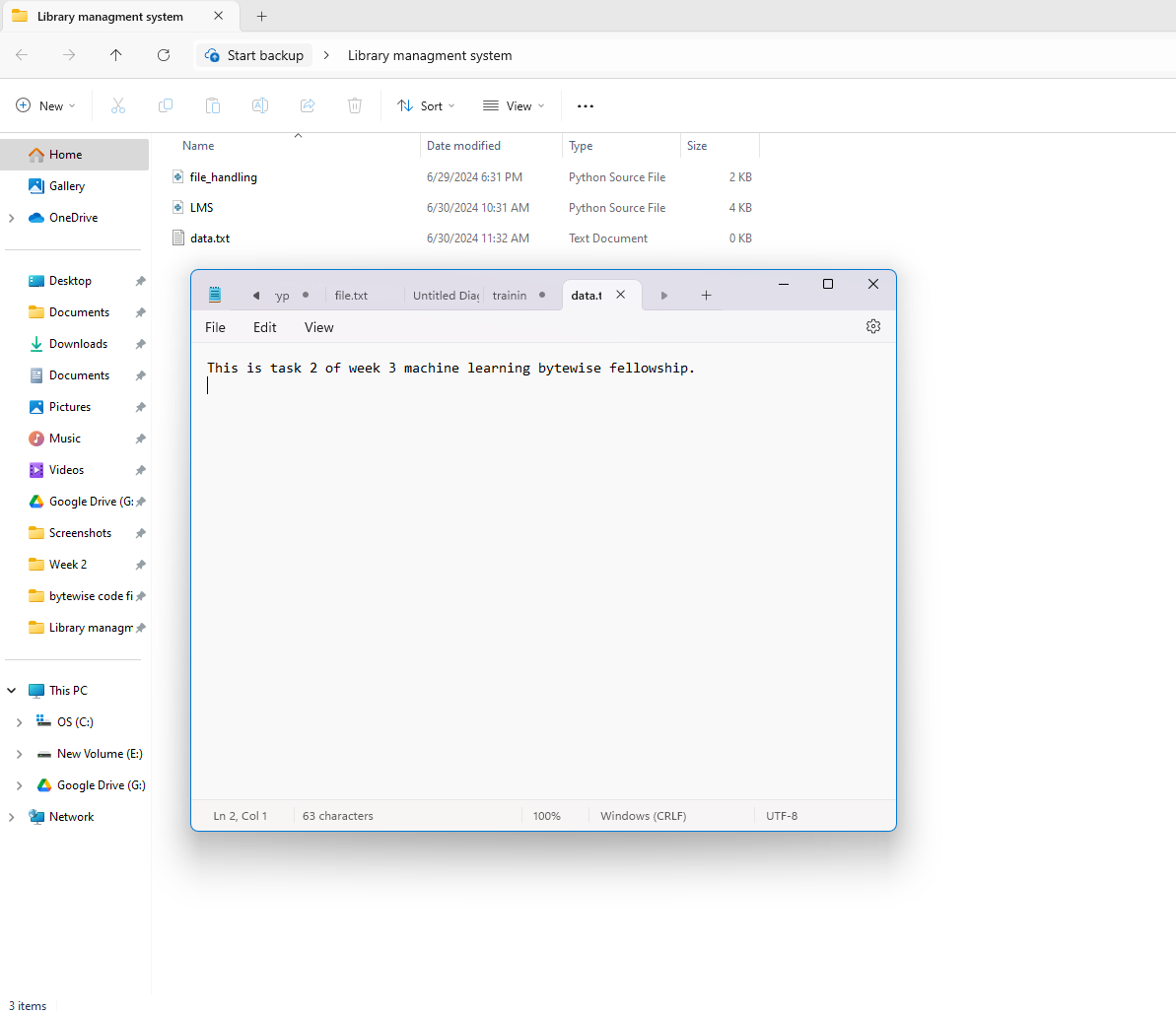
**Example in the Code**:

* The code uses try, except blocks to handle potential errors when setting attributes. For example, if the title is empty or pages is not a positive integer, a ValueError is raised and handled.

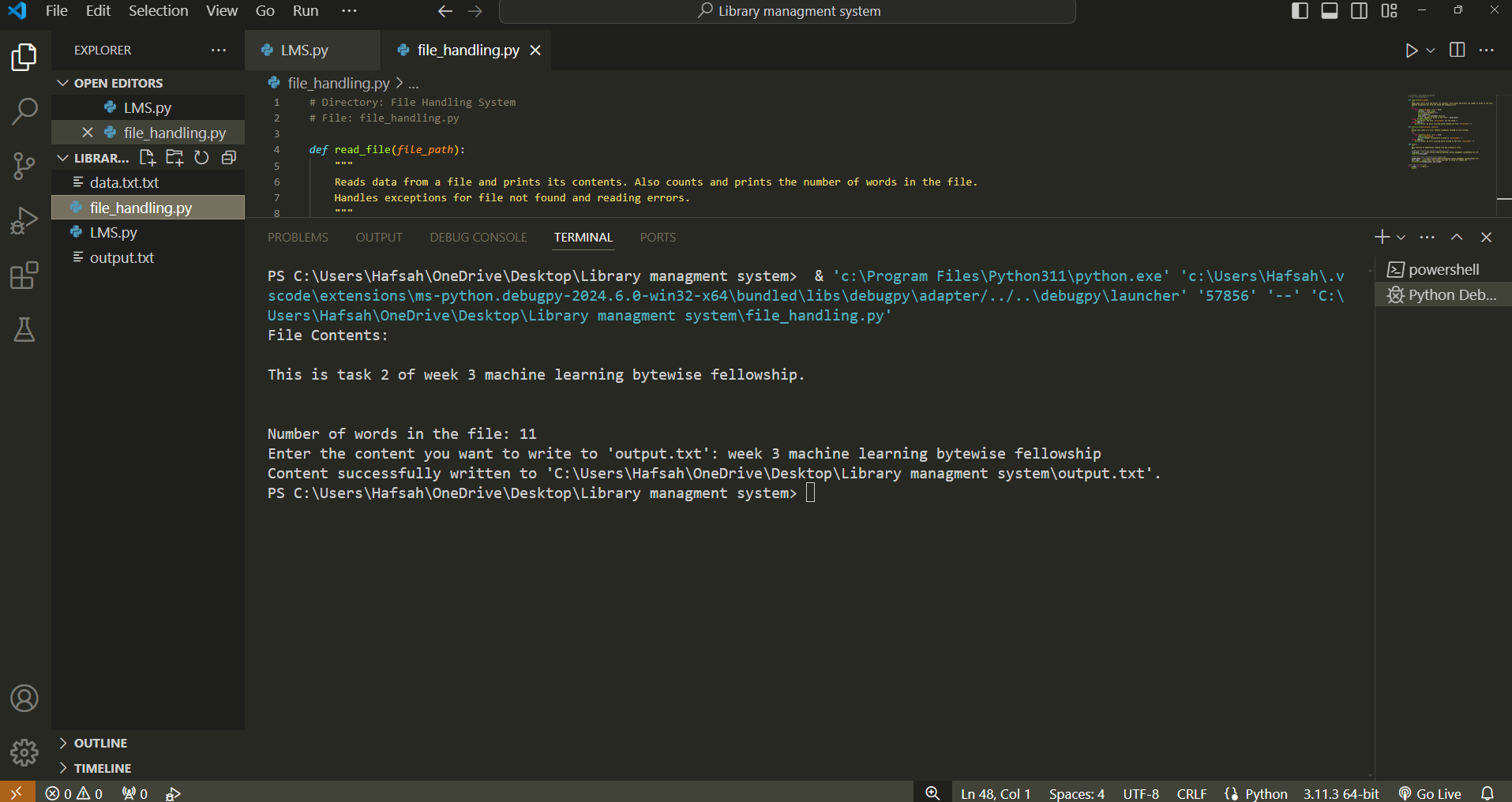
In summary, the code defines two classes, Book and Ebook. The Book class has attributes for title, author, and pages, with methods to get and set these attributes. It also includes a method to calculate the reading time. The Ebook class inherits from the Book class and adds an attribute for the format of the ebook. The code also demonstrates how to create instances of these classes and use their methods, with exception handling to manage potential errors.

**Take 2: output**

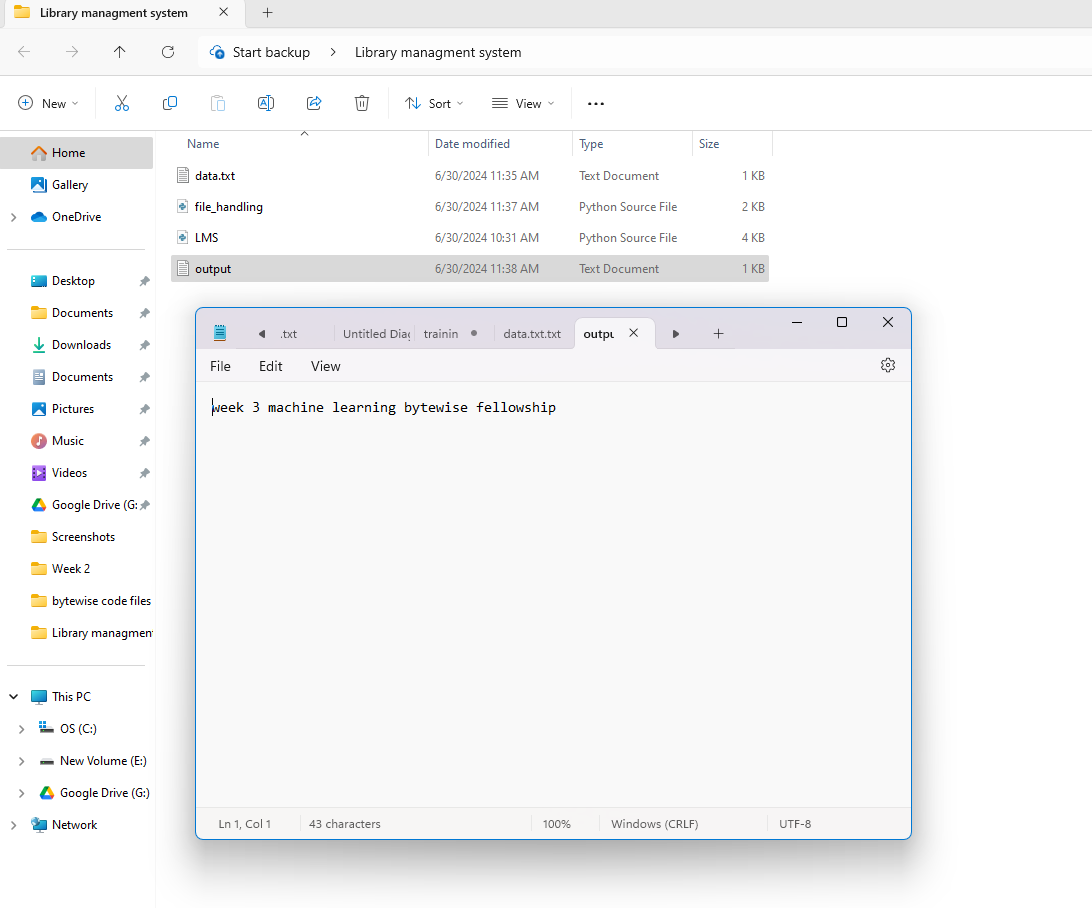
Input.txt:



Output:



Output.txt:



**Key Concepts in the Code**

**1. File Handling:**

* **Reading a File**: The script opens a file to read its contents and then processes or displays the data.
* **Writing to a File**: The script opens a file to write data into it, usually based on user input or some processed information.

**Example in the Code**:

* The read\_file function reads from a file.
* The write\_to\_file function writes user input to a file.

**2. Exception Handling:**

* **Try-Except Blocks**: These are used to catch and handle errors that might occur during the execution of the code. This ensures the program can handle unexpected situations gracefully without crashing.

**Example in the Code**:

* The read\_file function uses try-except blocks to handle cases where the file might not be found (FileNotFoundError) or other I/O errors (IOError).
* The write\_to\_file function also uses try-except blocks to handle potential errors during file writing.

**3. Functions:**

* **Defining Functions**: Functions are defined to perform specific tasks. This helps in organizing the code better and making it reusable.

**Example in the Code**:

* The read\_file function is defined to read and print the contents of a file and count the number of words.
* The write\_to\_file function is defined to write content to a file.
* The main function is defined to control the flow of the script, demonstrating the reading and writing functionalities.

**4. String Manipulation:**

* **String Methods**: Methods are used to manipulate and process strings, such as splitting a string into words.

**Example in the Code**:

* In the read\_file function, contents.split() is used to split the file content into words and count them.

**5. User Input:**

* **Getting Input from the User**: The script prompts the user to enter some text, which is then written to a file.

**Example in the Code**:

* The input function is used in the main function to get user input for writing to the file.

**6. File Paths:**

* **Handling File Paths**: The script uses raw string literals (r"...") to handle file paths on the Windows operating system correctly.

**Example in the Code**:

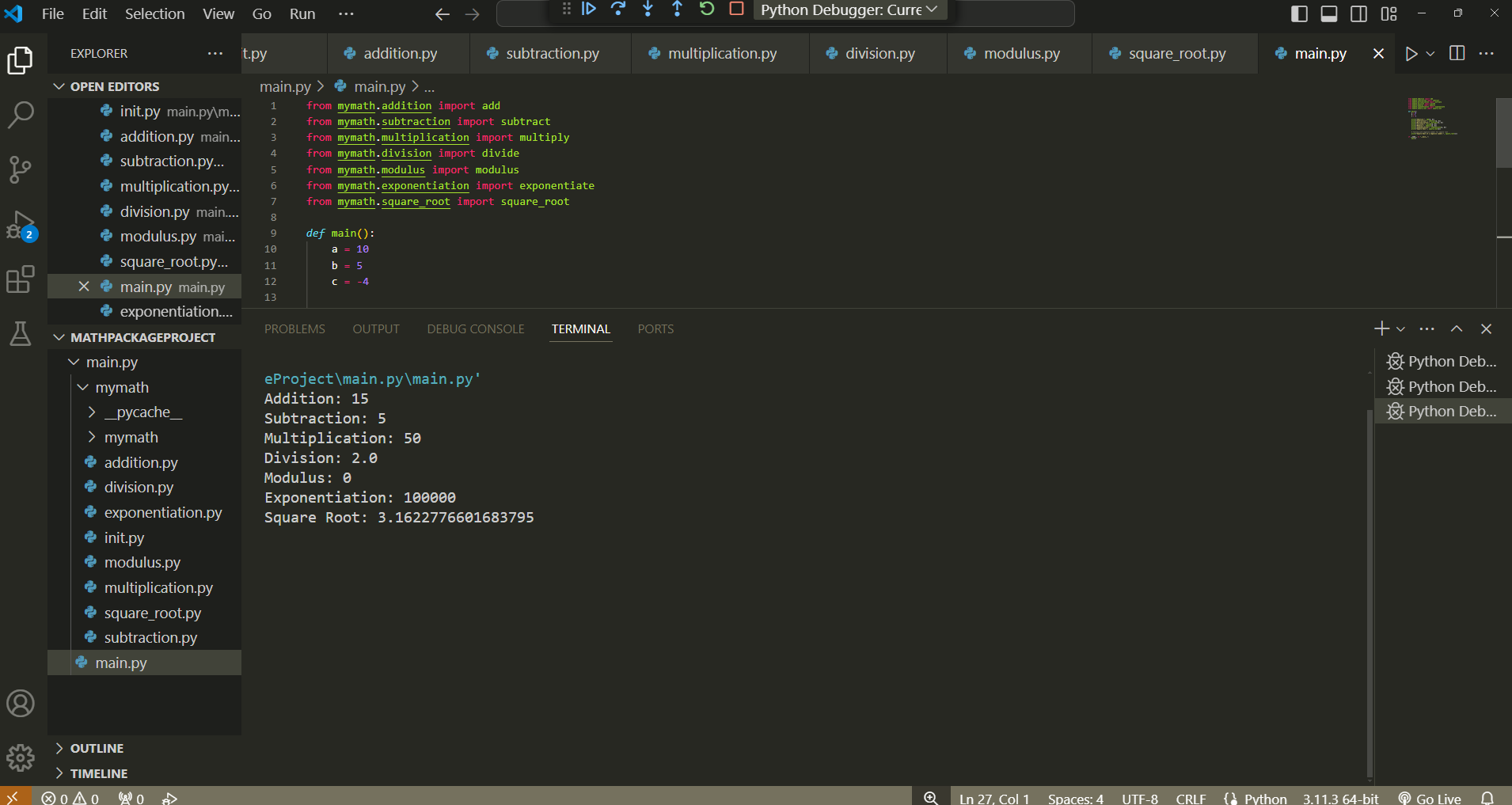
* The file paths for reading (data.txt.txt) and writing (output.txt) are specified as raw string literals to avoid issues with backslashes in the paths.

**Summary**

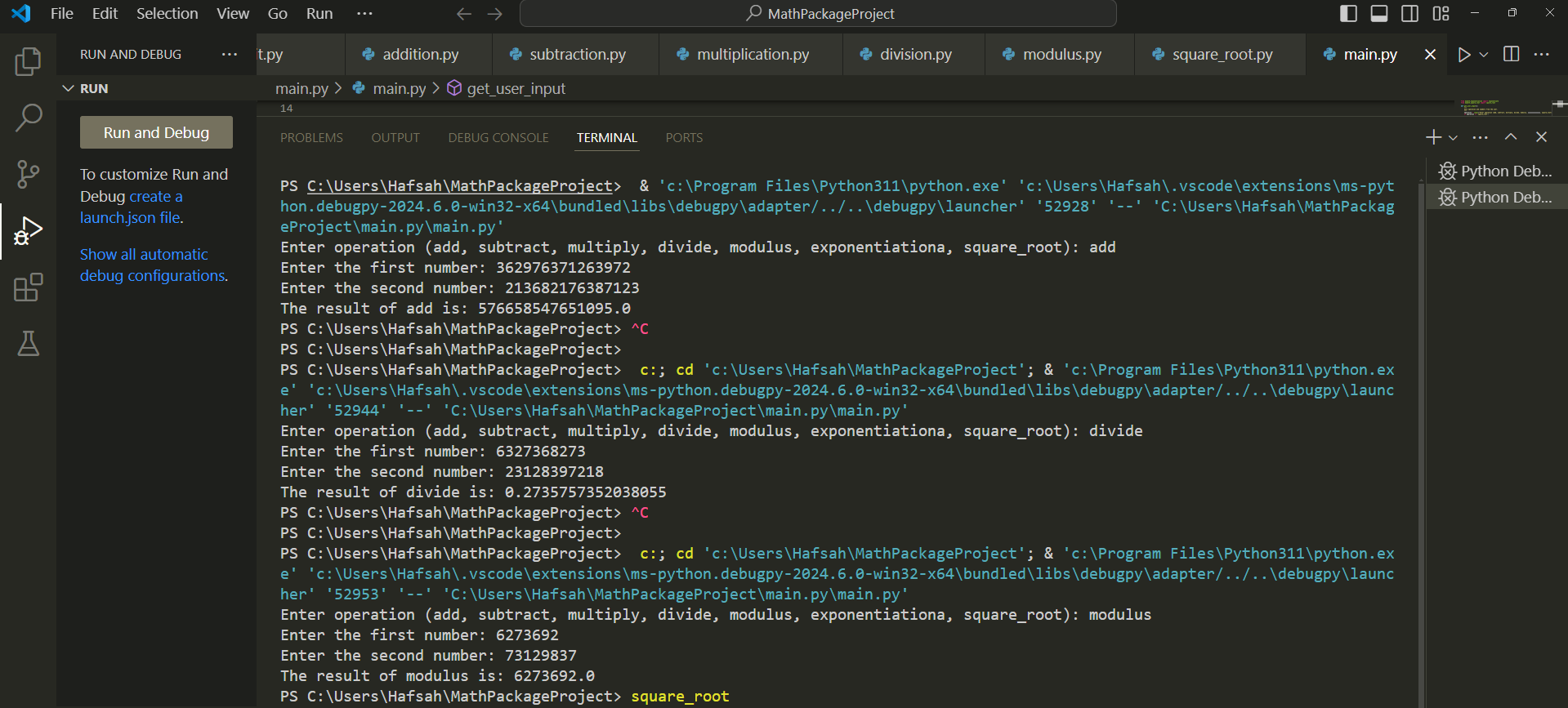
The script demonstrates file handling by reading from and writing to files, with appropriate exception handling to manage potential errors. Functions are defined to encapsulate specific tasks, making the code more organized and reusable. User input is handled to write data to a file, and string manipulation is used to count words in the file content. Raw string literals are used for file paths to ensure compatibility with the Windows operating system.

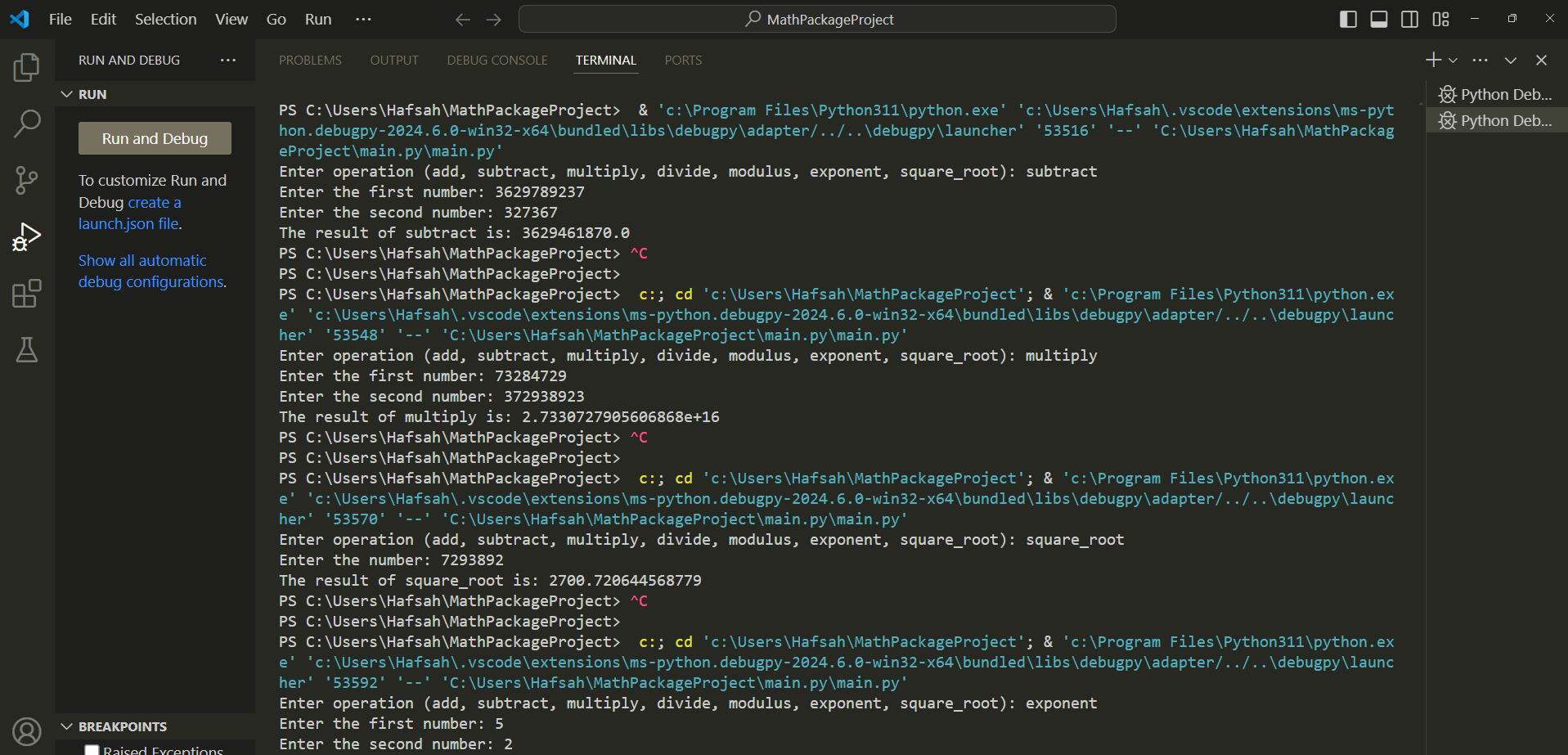
**Task 3:**

Output fixed values:



Output run time:





**Task 4: output**

