The server:
There are two command line arguments that the server takes: server.py and a port number. Peers will connect with that server on that port. While there are no peers connected yet, it will sit there and wait for a connection. When peer connects with the server, a new thread will start.
The following global variables are instantiated:

- fileDict: This dictionary has the filenames and for each file name, there is a dictionary of peers and the list of chunks it has from that file.

When the client sends a message to the server, it will come in an encoded json message (dictionary). The server will get the message number and run the function corresponding to that message:

1. register_request() = message 1
2. file_list_request() = message 2
3. file_locations_request() = message 3
4. chunk_register_request() = message 4

For message 5, the client connects to another client, which is why this function is not in the server file, it's in the client file. Each function will do the actual work and calculations to get the data then encode and send the message to the peer in the same format the server got it in. Then the server will wait for the next request from the client.

The client:
There are at least four command line arguments that the client takes.

- The client.py
- The port that connects to the server
- A port for peers to listen or other peers connections
- The name of that client (ex: client1, client2..)

Any additional arguments are files that the client would like to share to the server. The client also has an option to not share any files with the server.
Then it will initialize the following global variables:

- fileList: This dictionary is the same as fileDict from the server, but is constructed by the requests (messages) that the peer makes to the server.
- Chunkdatalist: This dictionary is used for files that the client doesn't have all the chunks of. The dictionary contains the files and the chunks with the chunk data.

Then it will prompt the client to send a request (message). The client can make the following requests:

1. Register Request
2. File List Request
3. File Locations Request
4. Chunk Register Request

5. File Chunk Request
6. Close out of the program

They only actually request 2,3,5,6. The descriptions of these requests can be found in the original Lab 1 document.

Each message has its own function. They are named message1(),message2()..etc.
Within each message function it will set up a message in the format of an encoded json dictionary. The dictionary contains the message number and the names of the files,chunks etc that the client is requesting. Some of the messages contain other messages in order to get all the necessary data for the client. For example, to carry out message 5, the client will need to run message 2, message 3, and message 4 within message 5 .

Since message 5 requests to a peer, it will connect to a listening port for peers. Message5() will send its data to File_Chunk_Request(). Then in Message5() it will write the file chunk to a new file if it does not have the file, otherwise it will create a new file and append it in there. The chunks are written in the correct order each time a new chunk is added.

After each message is carried out, the socket will close. The client will be prompted if it would like to send another message.