CS 371 Final Project: Socket Programming

By: Kyle Graves

The overall goal of this project was to learn about socket programming. How it can be used to save files in a remote location(cloud storage). In this project we had to create a way to connect to and communicate with a server in order to do basic commands like upload, download, delete, and see the contents of the file.

For my Project I have a client and server as well as folders used by the client and server to hold their data and files. My server folder is called “server\_data” and my client folder is simply called “client”. In my server\_data folder I have a text file called “DownloadFile.txt” and in my client folder I have 2 files, “data.txt” and “Upload.txt”. The overall design of my project followed closely with the design of the code we were given by the TA to help get us started. For the client\_1.py I only used 1 main function and used if elif statements to get user commands that were sent to the client. In server.py I used two functions. A main function that was used to establish a connection with the client and a handle\_client function that handled all of the commands of the client.

Starting with my client.py the first thing I did was create an input command to connect to the server, and if the command was incorrect they are asked to type the command again until it is typed correctly. I did this with a simple while loop.

While command is not “CONNECT”

Print a you were not connected to the screen

Print a type connect to connect display to the screen

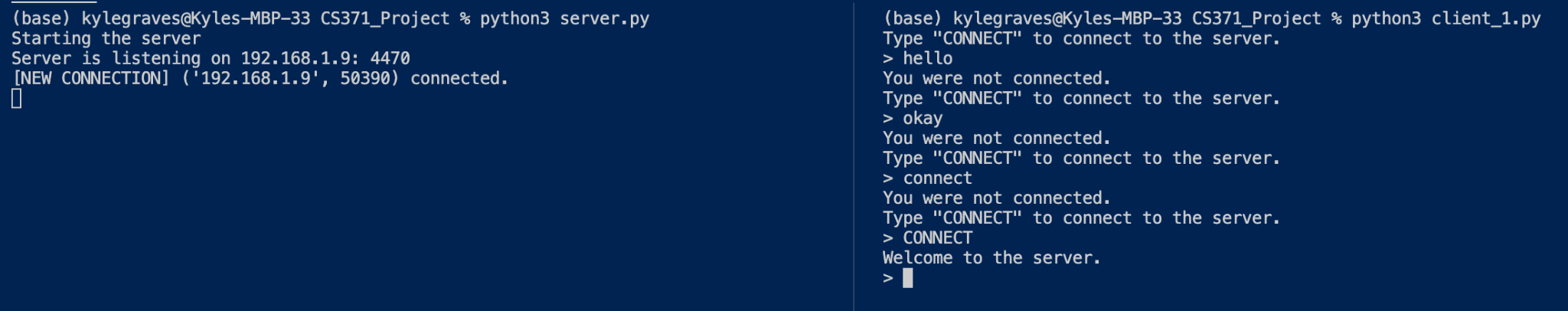
Get another user input.

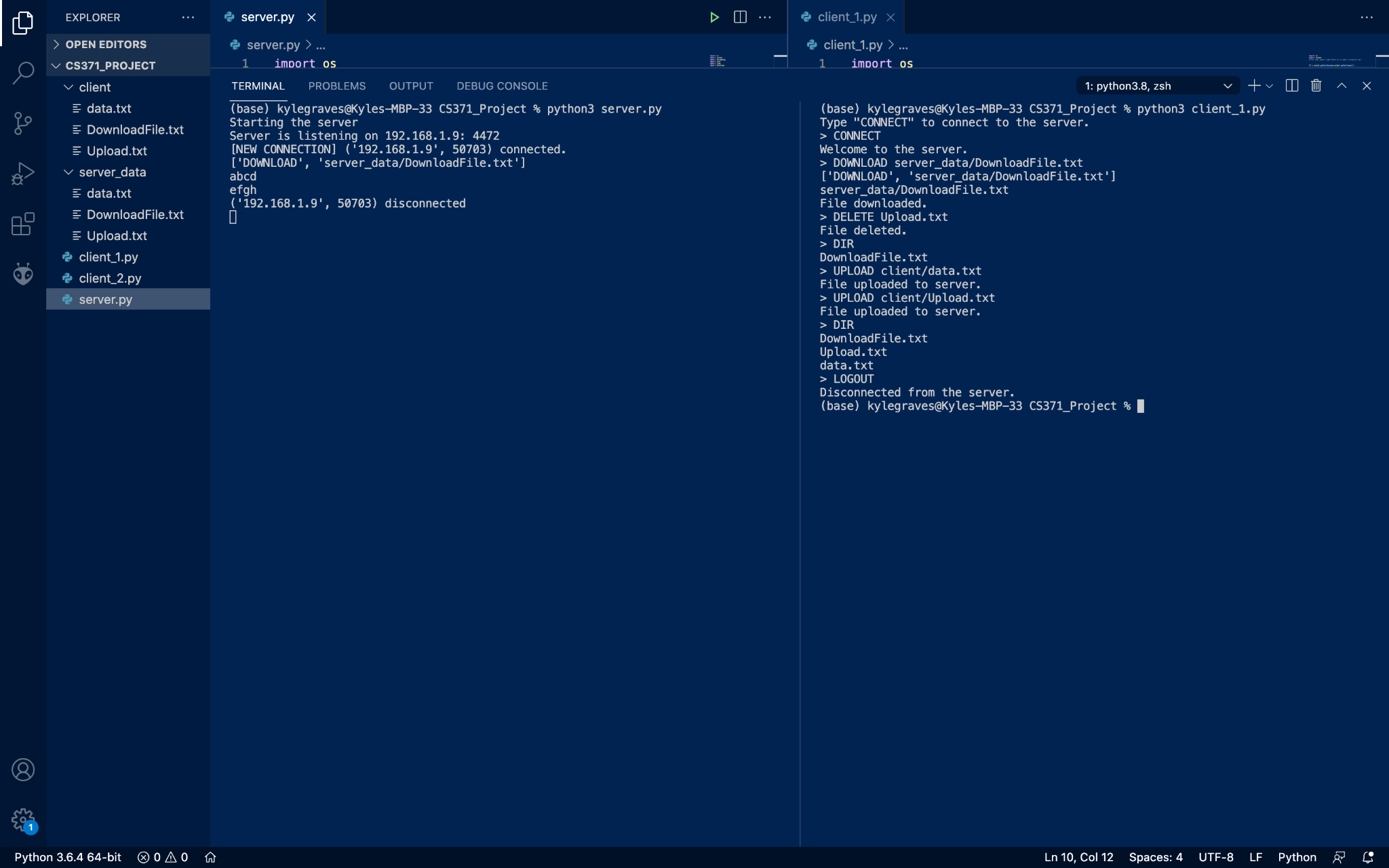
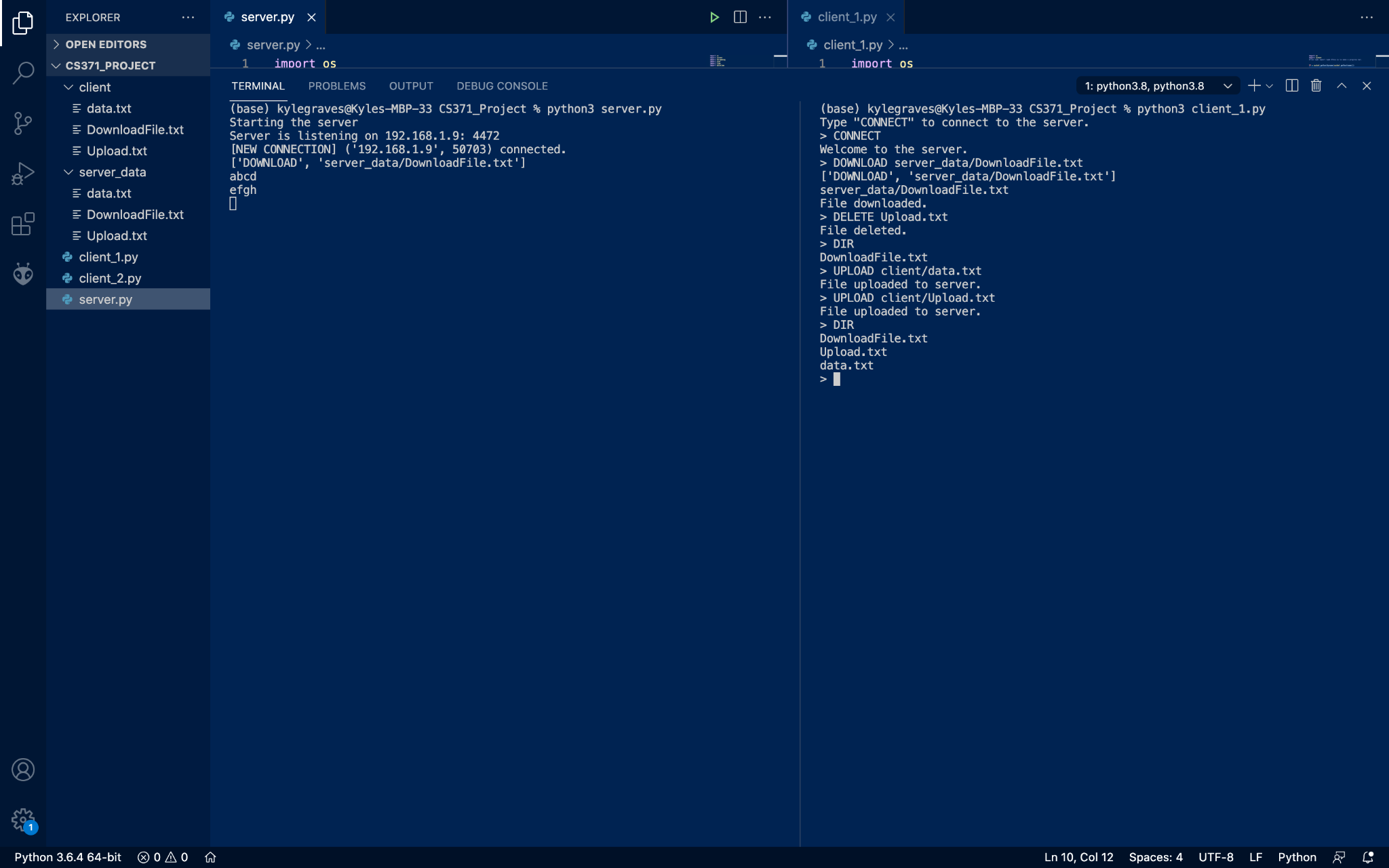
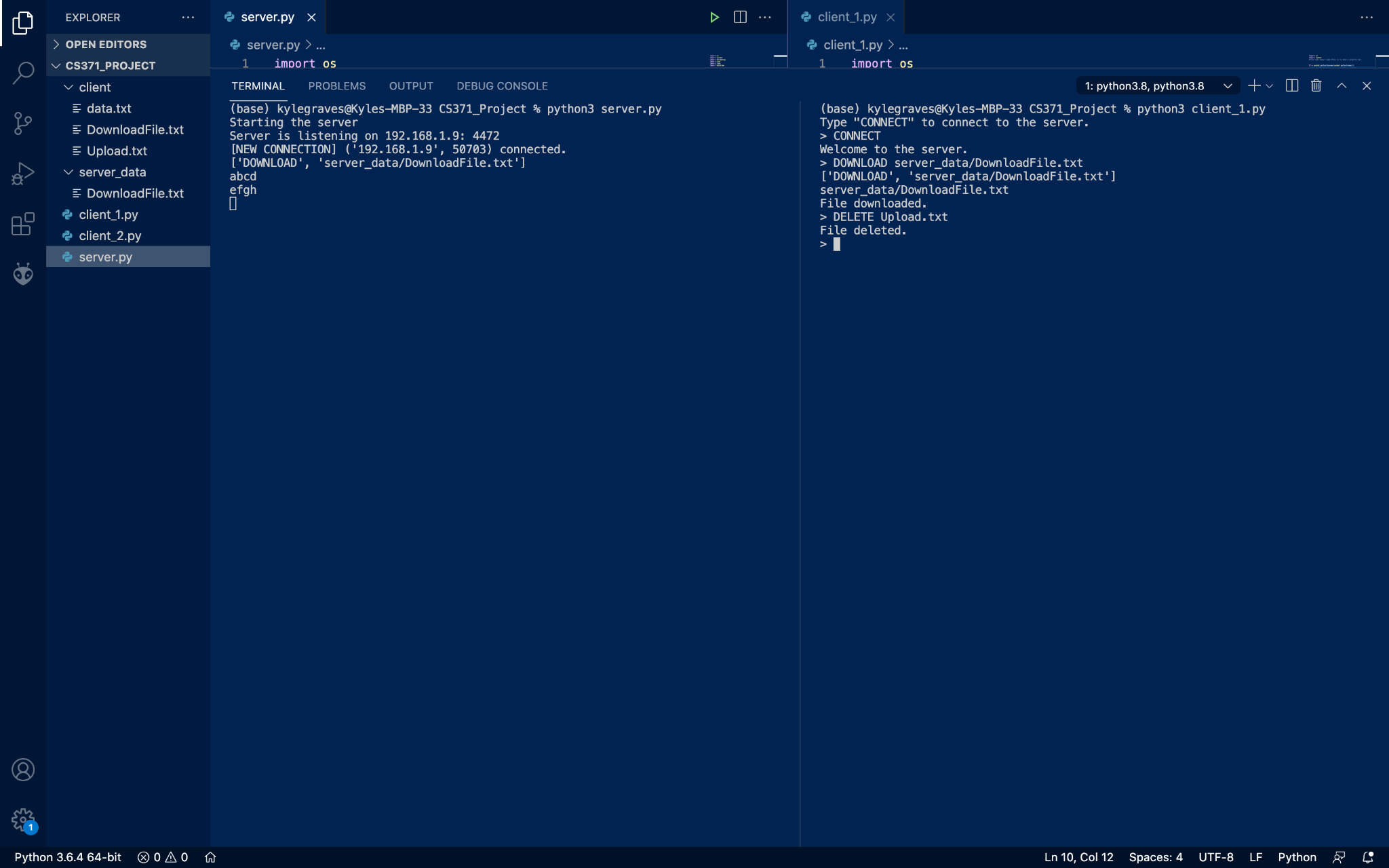
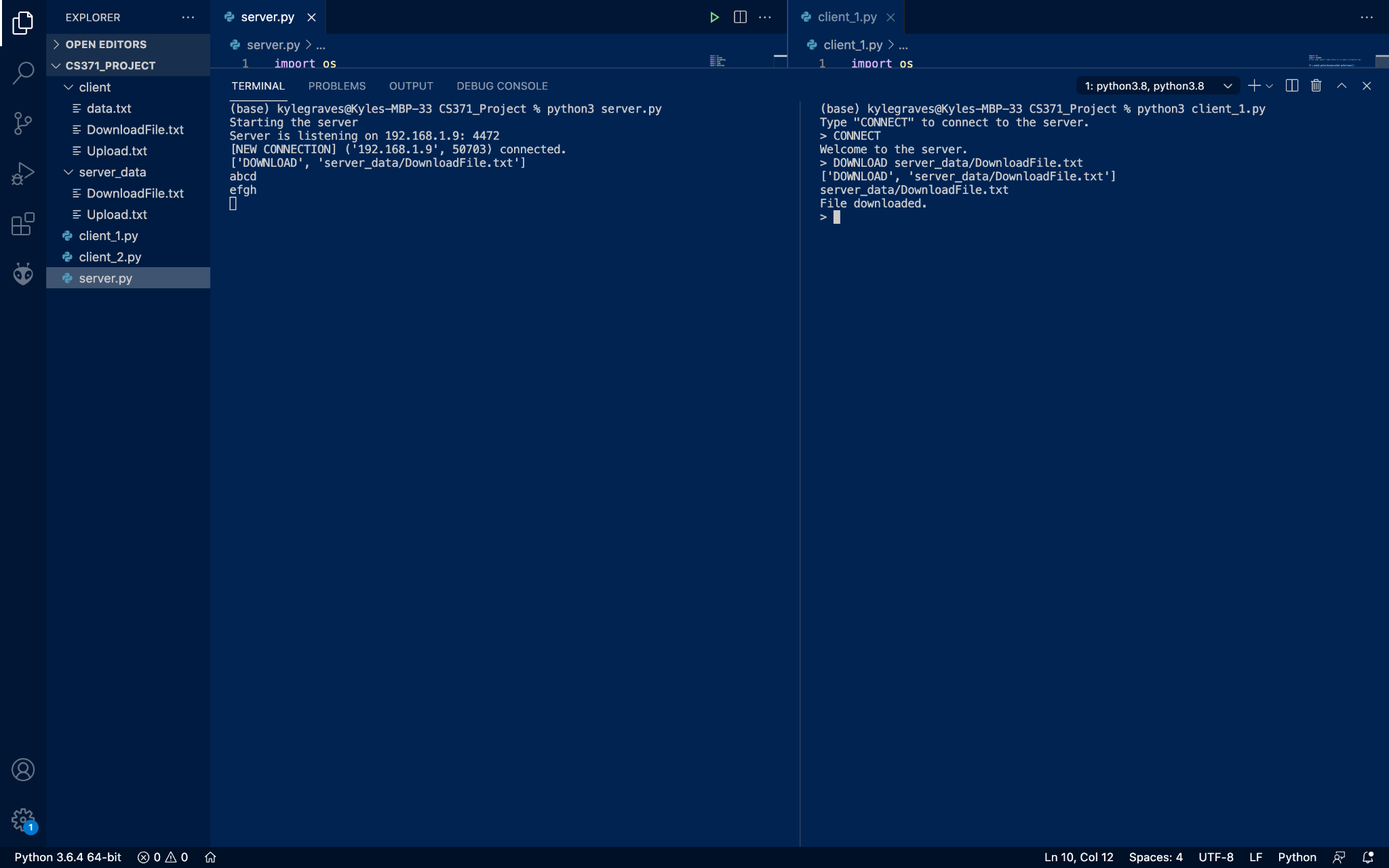
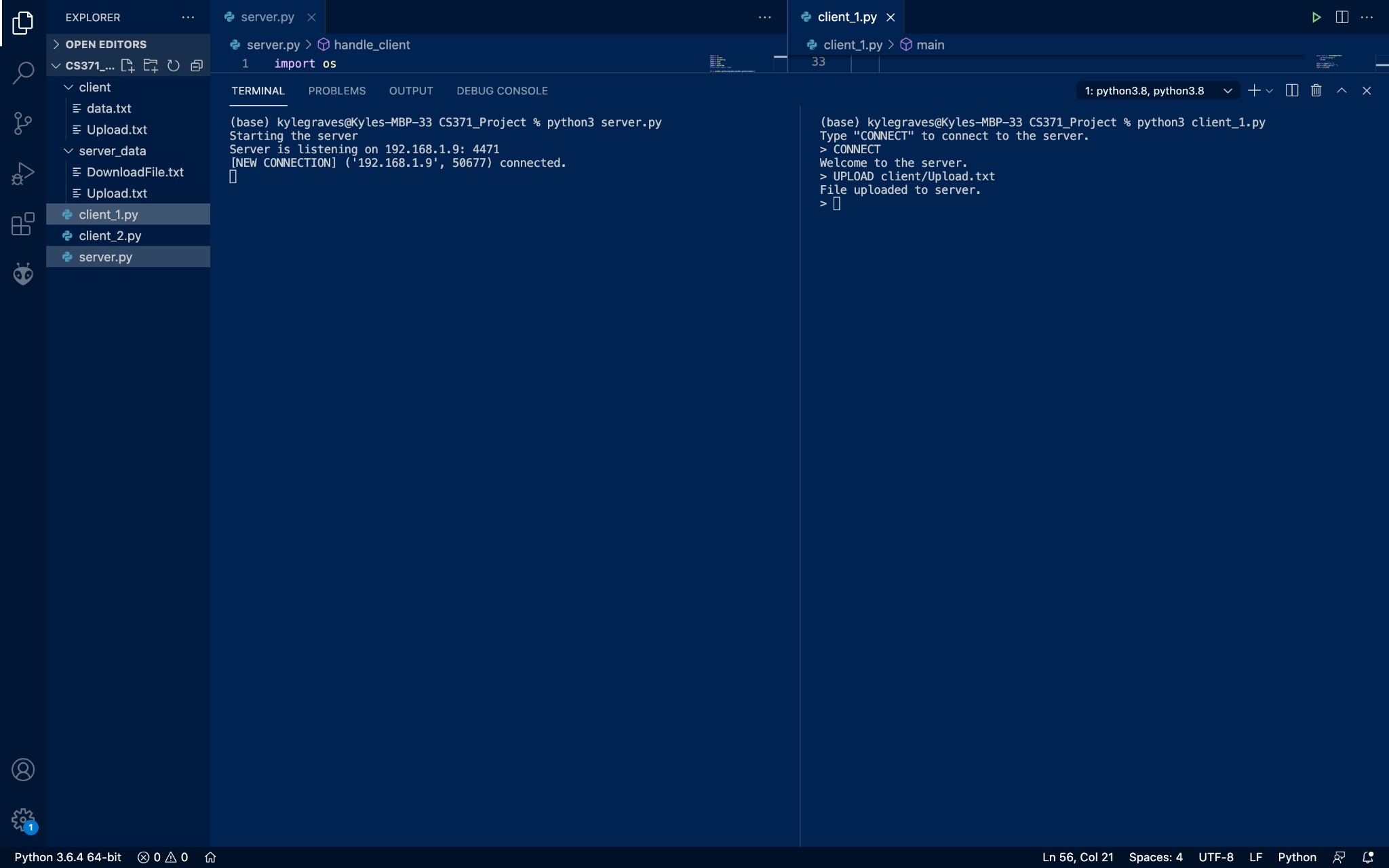
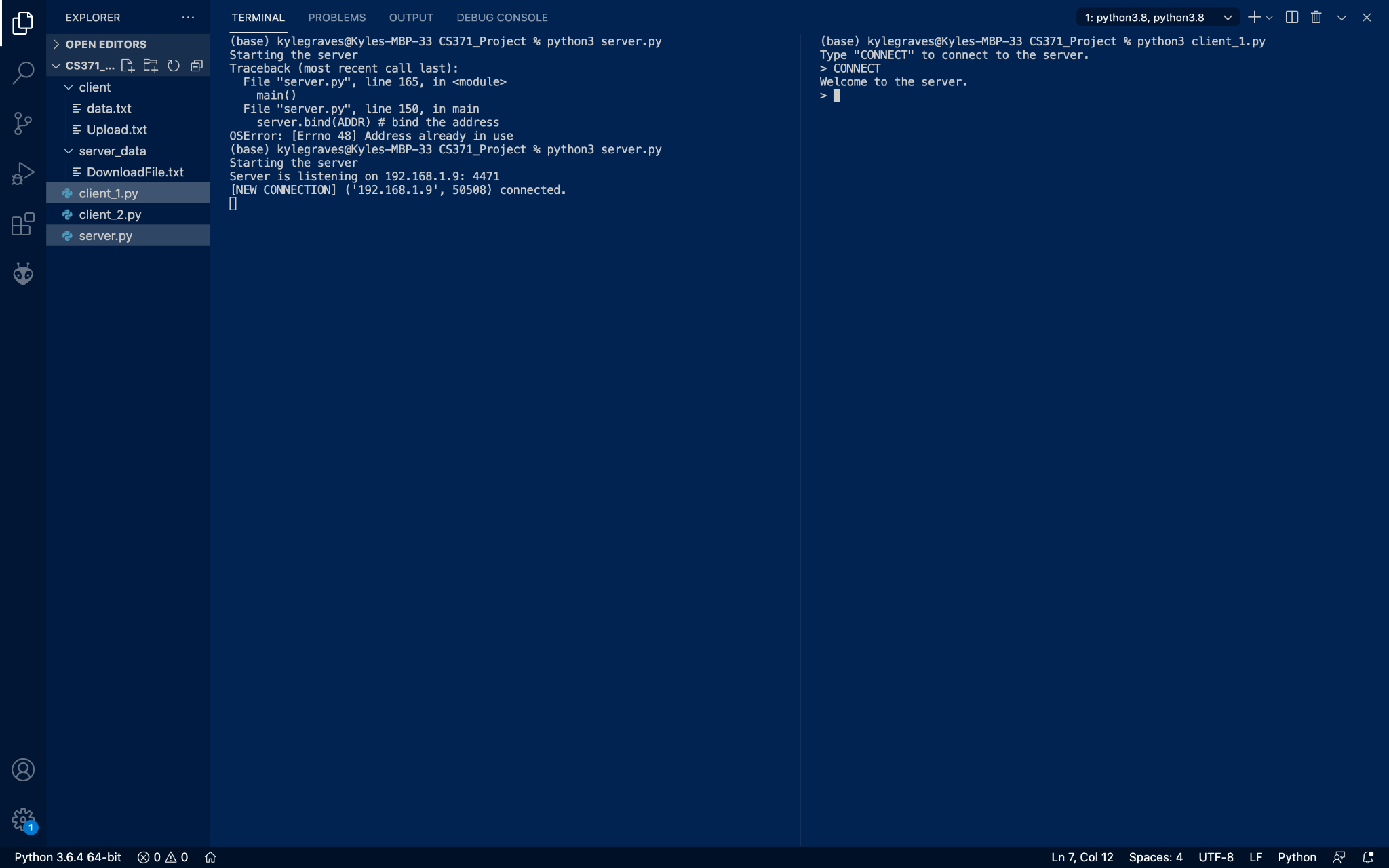
The next step after connecting to the server was to make commands to send to the server. To do this we create a user input called data, split the data into a command in the first position and call that cmd for the rest of the file. Making the commands is the next part. The Task command, like in the class demo just tells the user what commands can be used. My next command is upload. To complete this command I set a path equal to the second position of the data, used an open and read command to open the file in the path, and then set the text of the file equal to what was being read. After that was complete all that was left was to send all the information to the server. To do this I used a client.send command to send the client data in an encoded format to the server. Once this data gets to server.py, I split the data into a command at position 1, name at position 2 and text at position 3. Then I have to create a file path to send the data to, in the server. To do this I used the os function that can be imported in python. Once the file path was made, I just simply had to write the text into the given file path from the client.

The next command is download. This is very similar to upload, except you read and write on the server side only. All the client has to do is create a path and then send the path and command to the server. Once the data gets to the server the first thing you need to do is get the data separated and assigned to a variable. To do this you can set the path equal to the data in position 1, then you can split the path by “/” and set the name of the file equal to the last position of the split path because it will always be in that position. The next step is to see if the file is in the server data. To accomplish this I used the os.listdir function, which is a function in the os library that lists all the files in a given directory. Next you make a simple if statement, if file is in files given in the directory, open the file. If not send information back to the client that the file was not found. Once the file is opened, you read the file and save to a variable and then create a new path using os.path.join like we used in upload to write the file back to the clients folder.

The next command is delete. The delete command will delete a file from the server. To do this the client sends a command and the path to the server. Once received the server checks to make sure that there is anything in the folder to be deleted, and then checks the folder to see if the file given is in the folder the same way we did in the download command. To delete the file we can use os.system file remove function.

The last command is DIR, this command lists the files in the server directory. The client only has to send the command used for this. Once the command is received by the server, we check to see if anything is in the directory, if not we tell the client that the directory is empty. If not we send data back using join. Putting “x for x in files” inside of join will join all of the files together and putting “/n” will put each joined file onto a new line.

Unfortunately, I was never able to get download and upload speeds to work correctly in my commands without breaking my functions all together so I don't have any data performance stats to report for this project. 



In my screen shots you can see the files being added or removed on the right hand side.