**UDP Calculator**

**Description of what it does.**

Firstly we import our library, set up our port and our server and ensure that that’s working.

Essentially the client sends three pieces of information to the server - the first number, the second number and the third number.

The first two numbers are used for the actual calculation, whereas the third is used to determine which of the operators to use, as the server has an IF statement that changes the operation dependent on the number entered.

I used a number system here as in my opinion it’s easier to validate a system with numbers than with entering the actual characters, and it significantly reduces the chance of human error, as the previous two values were also numbers so you’re just doing the same thing you’ve done twice before - and on the off chance you were to deal with a massively computer illiterate individual using the \* operator might be a problem if they were unaware of the means to use the above number characters, or using the wrong side of slash - in general using a number based system just makes more sense in my opinion in terms of how to initiate a calculation.

I’ve also included some input validation for the third number on the client side that ensures any given value entered must be between 1-4 and therefore must correspond to an action on the server, it repeats until a valid value is given. In order to do this we have to convert the string to an integer beforehand, and then once checked reconvert it back to a string so we can send it on to be encoded as a byte.

Once we’ve sent all three numbers to the server it goes through an if statement that either adds, subtracts, divides or multiples the first two numbers entered based upon what operator was chosen through the 3rd number. The server then prints the first number, the second number and using another if statement prints the action chosen in words, so “Subtract” “Add” etc - because it’s just easier to look at than a 1 or 2 for instance and is clearer for troubleshooting - not at all necessary, I could’ve just had it do the calculation and return it - but it helps to convey that the server is getting all the information.

We then convert the outgoing message to a string, then to bytes, and then it can be sent to the client address. We then close the socket as we’ve sent our calculated message to the client.

There’s a piece of code following this that essentially calls for an input, and it’s purely here for keeping the command line open because otherwise the console I was using(the regular python command line, because I needed to run both the server and client and didn’t know how in VScode) would close after the operations were complete - it’s not necessary, but depending on your means of running the python files, it may be useful, and it doesn’t really harm anything as it goes.

The client then receives the calculated value and sends it to display - the same input requirement code is here for the reason stated above.

The user can add, subtract, multiply and divide any two numbers given, even extremely large numbers - all of this is done server side and then transferred to the client - therefore it meets the brief.

**Libraries**

I only used one library - that being the socket library, which is necessary for the usage of all the low level networking we’re doing with the servers and clients. It allows for the creation of sockets, which allow for us to send information across a network. Our network is based exclusively on a single PC so we use localhost or 127.0.0.1 as our server name/host. Through various methods within the library it allows the created server and client to communicate on the server via the ports we’ve specified and sockets we’re using to transfer information between each other, without the socket library - none of this would work.

**Testing**

|  |  |
| --- | --- |
| **Multiplication** | **Divide** |
| **Addition** | **Subtraction** |
| **Validation Test** | **Testing shows that all 4 operations work and that validation is functional.** |

**TCP READER**

**Description of what it does.**

Obviously we import our libraries and set up our host and our port that we’re going to be using firstly. Then we ask the user to enter a file path for a txt file, this is our input. We then take this input and open the txt file at the given location - and we then have to read it. Once we’ve read the txt file it’s now a string, and we can then send this string to the server. The server receives the string, decodes it and then works out two things - the amount of characters, and the amount of words. We work out the amount of characters using the len function that’s built into python, and we work out the words using the len function in combination with the split function that essentially works out words by taking anything separated by whitespace as a separate word. Once we’ve worked that out, we just have to convert our integer values into a valid data type to be sent back to our client. Now, I don’t know if my method is the most efficient or the only way - but the way I did it was by converting first to a string, then to bytes, this in turn meant that they could be sent back to the client, decoded and then we sent the values to display. *Note: The TCP server will not output anything to the console as it’s just doing calculation, and then those values are returned to the client.* The program does upload the contents of a provided txt file to a server and then calculate how many words and characters respectively are contained within and then transfers that information to the client, therefore it successfully meets the brief.

**Libraries**

Again, only made use of one library - that being the socket library, no other libraries were required as I was using only a server and client system that was fully supported through the use of the socket library.

**Testing**

|  |
| --- |
| **Test 1** |
|  |
|  |
| **Test 2** |
|  |
|  |
| **Test 3** |
|  |
|  |
| **Test 4** |
|  |
|  |

Overall my testing proved that my system worked and was verified by <https://wordcounter.net/>