

Due Date: Friday, October 11 at 3:00 PM.

PART I: Socket Programming Project (to be submitted as a zip file on Coursys)

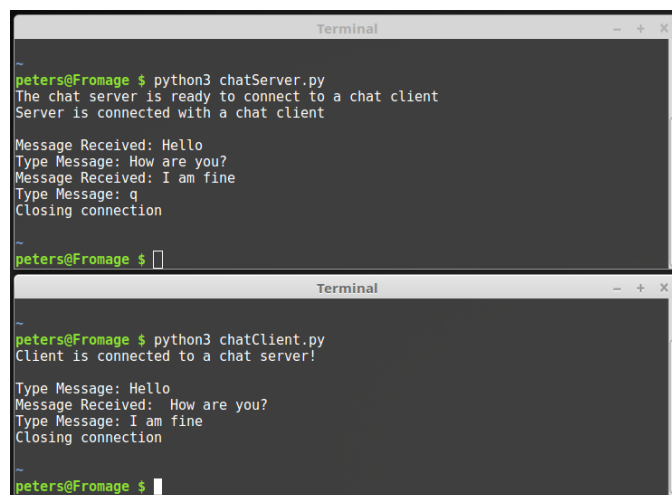
In this programming project, you will develop a simple calculator application using socket programming in Python. You will build this project in two steps starting with provided skeleton code for a chat application. You will have to modify the skeleton code to fulfill the requirements.

Step 1: The skeleton server code named `chatServer.py` and client code named `chatClient.py` are stored as a zip file which can be accessed using a link on the Homework page of the course website. The server is a TCP server which waits for a connection request from a TCP client. The client should be able to send a connection request to the server. After the connection is accepted by the server, the server and client enter into the chat mode. Most of the code for chatting is already provided. The first step of the assignment is to fill in the missing code where you find ... in the skeleton code so that server and client can talk to each other.

The code provided runs with Python 3 and was tested (before some lines were replaced with ...) in a Linux environment. It should run on Windows and Mac as well (but it was not tested on Windows or Mac). Be sure that you have Python 3 or later installed on your computer. It will not run correctly with Python 2.7 or earlier. Running the programs is easy. After you complete the two files, open two terminals (one for the server and one for the client) and change directory to where you have stored the files. Run `chatServer.py` from one terminal and then `chatClient.py` from the other terminal (the server must run before the client). You should be able to run the programs using the following commands (one in each terminal).

```
python3 chatServer.py (to start server)
python3 chatClient.py (to start client)
```

Your code should work locally (without an external IP). You should use `localhost` as your IP address in the code. (You can change it to something else later if you are interested.) If everything works, the client and server should be able to alternate sending messages to each other as shown in Figure 1. Typing `q` and `Enter` in either terminal will close the connection. (This part is already done in the skeleton code.)



```
Terminal
~
peters@Fromage $ python3 chatServer.py
The chat server is ready to connect to a chat client
Server is connected with a chat client

Message Received: Hello
Type Message: How are you?
Message Received: I am fine
Type Message: q
Closing connection

peters@Fromage $

Terminal
~
peters@Fromage $ python3 chatClient.py
Client is connected to a chat server!

Type Message: Hello
Message Received: How are you?
Type Message: I am fine
Closing connection

peters@Fromage $
```

Figure 1: Simple chat application screen capture

In summary, Step 1 is to complete the server and client so that the chat application works as expected. Save the files as `chatServer.py` and `chatClient.py` (you will submit these files).

Step 2: After completing Step 1, you should have a simple but functional chat application. In Step 2, you will modify the code to implement a calculator client-server application.

The server receives requests from clients to perform arithmetic calculations and returns the calculated values to the clients. The server should be initiated once before any clients are started and should be able to serve multiple requests from the same or different clients. The server can use the “eval” operation of python to evaluate the input received from the clients. The documentation for the eval operation in Python is here:
<https://docs.python.org/3/library/functions.html#eval>

The clients connect to the server with requests for equations to be evaluated and then print the results. A client should be able to terminate its connection with the server in a clean manner (by sending q).

Example:

Client1 sends: $2*3*4*5$
Server runs `eval(2*3*4*5)` and returns the result 120 to Client1
Client1 prints 120
Client2 sends: $2+3+4+5$
Server runs `eval(2+3+4+5)` and returns the result 14 to Client2
Client2 prints 14
Client2 sends: q
Client1 sends: $(1+3)*(5-2)$
Server runs `eval((1+3)*(5-2))` and returns the result 12 to Client1
Client1 prints 12
Client1 sends: q

In summary, Step 2 is to complete the server and client so that the calculator application works as expected. Save the files as `calcServer.py` and `calcClient.py` (you will submit these files).

IMPORTANT: Start from the skeleton code provided and do not change the function names and program structures. Only modify the code where necessary. We will test your code with a script and it must pass our tests for you to receive credit. Both server and client should run on localhost. So even if you experiment with other IP addresses, before submission please make sure they work on localhost without any external network connection. We will use Python 3 or later to run the test scripts.

Submission: Submit four python files and a few screen captures of your working chat and calculator applications (as a single zip file) to **Coursys**. Python files to be submitted are:

1. `chatServer.py`
2. `chatClient.py`
3. `calcServer.py`
4. `calcClient.py`

PART II: Questions on Chapters 2 and 3 (to be submitted in the course assignment box)

1. A file of $F = 10$ Gbits needs to be distributed to N clients/peers. The source node which contains the file has an upload rate of $u_s = 40$ Mbps, and each client/peer has a download rate of $d_i = 3$ Mbps and upload rate u . For $N = 10, 20, 40$ and $u = 256$ Kbps, 512 Kbps, 1 Mbps, prepare tables showing the minimum distribution time for both client-server distribution and peer-to-peer distribution for each combination of N and u . Note: Use 1 Gbps = 1024 Mbps and 1 Mbps = 1024 Kbps for this question.
2. UDP and TCP use a 1s complement checksum to detect bit errors (flipped bits).
 - (a) Compute the 1s complement checksum of the following three 16-bit words showing all of your work: 1101001101000100, 0111100001010101, 1110101110000101.
 - (b) Show that the checksum in part (a) can fail to detect two flipped bits.
 - (c) Describe the conditions that will result in the failure of the checksum of n 16-bit words to detect two flipped bits.

Note: part (b) is asking for a specific example; part (c) is asking for general conditions.

PART III: Wireshark Lab 2 (to be submitted in the course assignment box)

The link to Wireshark Lab 2 is on the Homework page of the course website. You should do the lab with a live connection, not the traces mentioned in the second footnote.