My name: Jacob Hummer
Group member names: Jacob Hummer
(just me)

Program that I want to make: Comparison of UDP vs TCP vs HTTP connection:close vs HTTP connection:persist vs HTTP/2 vs HTTP/3 for transferring N bytes and/or X files of Y bytes. Basically a speedtest of all the different networking stream types. Like https://www.speedtest.net/ but with more dials instead of one big button. Of the big four languages that we talked about in class I'd like to use Python; it's the simplest to get started using dependencies with; you just pip install something and you're good!

Something like this:

```
Myapp –server:
Running server on localhost:8000 for TCP
Running server on localhost:8001 for UDP

Myapp –client:
What's the address of the server for testing TCP?
> your typing here
What's the address of …
What format do you want to use to transfer data?
[udp, tcp, …]
> UDP
How many sockets?
> 10
How many bytes do you want to transfer over each socket?
> 100kb
Running…
Opening socket 1
Opening socket 2
Opening socket 3
…
Finished socket 1 100kb UDP at 97% integrity from localhost:8001 in
0.9seconds
Finished socket 2 100kb UDP at 50% integrity from …
…
Total speed: 100kb/s
Average integrity: 90%
Adjusted speed: 90kb/s
```

Language: Python
Summary: speedtest across UDP vs TCP vs other flavors of net traffic

Minimum viable product:

- A program that compiles
- Text based CLI tool
- Has two modes: client mode and server mode
- Let user determine whether to use TCP or UDP
- Let user determine the number of bytes to send
- Reports the findings
- Make it work on Windows

Stretch goals:

- Let user choose from more modes such as: TCP, UDP, HTTP/1.1, HTTP/2, HTTP/3, QUIC, etc.
- Let the user chose how many sockets to use (compare 1x socket to 10x sockets for example)
- Save a history of previously run tests
- Show a chart of how current test compares with previous tests
- The ability to print findings to an external file or share them via Email or something
- Compare Pyhon to another language like Java to compare speeds
- Add explainer text or other documentation as to what results you SHOULD EXPECT to see given X conditions
- Add explainer text or other documentation about what each TCP UDP HTTP/3 QUIC etc means in this context
- Add data integrity checks like “what percentage of the data did I get?” and “how many out-of-order bytes were there?”
- A hosted version of the server on AWS or Vercel or Netlify or somewhere so that users can test it from US-EAST-2 to get a longer travel distance than just from their own computer or across the room to their laptop
- Report progress as network events happen

My thoughts on how to do this:

- To do the program that compiles I can use Python
- To make it text-based I can use print() and input() and argparse
- To make the server and client I can use https://docs.python.org/3/library/socket.html and https://docs.python.org/3/library/socketserver.html
- Then need to use print() input() and run the program to do the test
- Then print the results. No integrity checking; just speed in seconds/ms.

My thoughts on how to do the stretch goal stuff:

- Have LOTS of if <option_chosen> then <do_the_option_specific_code> subroutines for each TCP UDP etc.
- Want to save prev tests in memory so can show them in a list/log

- To do the chart need something like https://github.com/piccolomo/plotext
- To save to a file is ez
- To share to email is ez since I can use a mailto: link
- Doing the same thing in Java is probably tricky and time consuming and impractical and more of a pie-in-the-sky goal
- For the documentation just explaining stuff in the README is good but maybe adding some in-GUI “UDP: A unidirectional firehose of data with no builtin integrity checking” -type descriptions or helptext
- To do data integrity maybe do some kind of protocol like where each subsequent byte is just incrementing by one so it’s like “1 2 3 4” etc for each byte as a u8 int? I don’t know.

For the additional transport mechanisms like HTTP/1.1 or whatever that are built on top of other underlying tech I’m thinking it would work like where it just treats the HTTP body text as the “stream” or data. That would mean that the extra HTTP header stuff would just count as extra handshake time to establish the inner socket. Something like HTTP/1.1 persistent over 100x 1kb files though might have a substantial advantage over plain TCP even though it has a higher per-socket overhead it would only need one connection instead of 100x (since it could have persistent mode enabled). Stuff like that would be interesting to see in real world -ish (albeit manufactured speed test) scenarios!