



# MAJORWORK PORTFOLIO

Program Name: Blue File

Jaydn Chiert

# Table of Contents

TABLE OF CONTENTS .....	1
ABOUT MY SOLUTION .....	2
PROJECT MANAGEMENT TECHNIQUE .....	3
SOCIAL, ETHICAL AND LEGAL ISSUES .....	4
DIAGRAMS .....	5
INTERFACE, CUSTOMISATION AND COMPATIBILITY CONSIDERATIONS .....	8
LOGBOOK .....	10
TESTING AND EVALUATING .....	37
FINAL CODE .....	41
REFLECTION .....	54
REFERENCES .....	56

# About My Solution

## Comparison between other programs

The software I have developed is a file transfer program, called “Blue File”, allowing users to send and receive files up to 10gb between Windows based machines.

This software is inspired by Apple’s airdrop, which allows users to send files between Apple devices. While there are a few programs that allow Windows users to do the same, such as FileZilla or Dropbox, there is no Windows standard equivalent.

FileZilla is complex to use, requiring a separate client download and server download to work. The interface is also confusing. My program aims to fix this issue by combining the client and server into one program as well as simplifying the user interface as much as possible, catering to all home and business users.

Dropbox works by uploading a file to a server, then having other people downloading the file from there and my program aims to improve this system, allowing two users to connect directly to each other, if they are on the same network. They are then able to send the files directly to each other without having to upload to a server first.

My software aims to improve the overall ease in sending files between Windows machines by allowing two users to directly send the files to each other, improving on current programs.

## Importance of documentation

Documentation is important in software development because it helps keep track of the product, ensuring the highest quality software. Documentation is also particularly important in the maintenance of the product, allowing the original developer and other new developers to understand each aspect of the program in order to easily improve it in the future. I have documented my application

User documentation is also important in ensuring the user knows how to use the product. I have implemented user documentation as small lines of text in the main GUI of the program, to ensure they understand each aspect.

# Project Management Technique

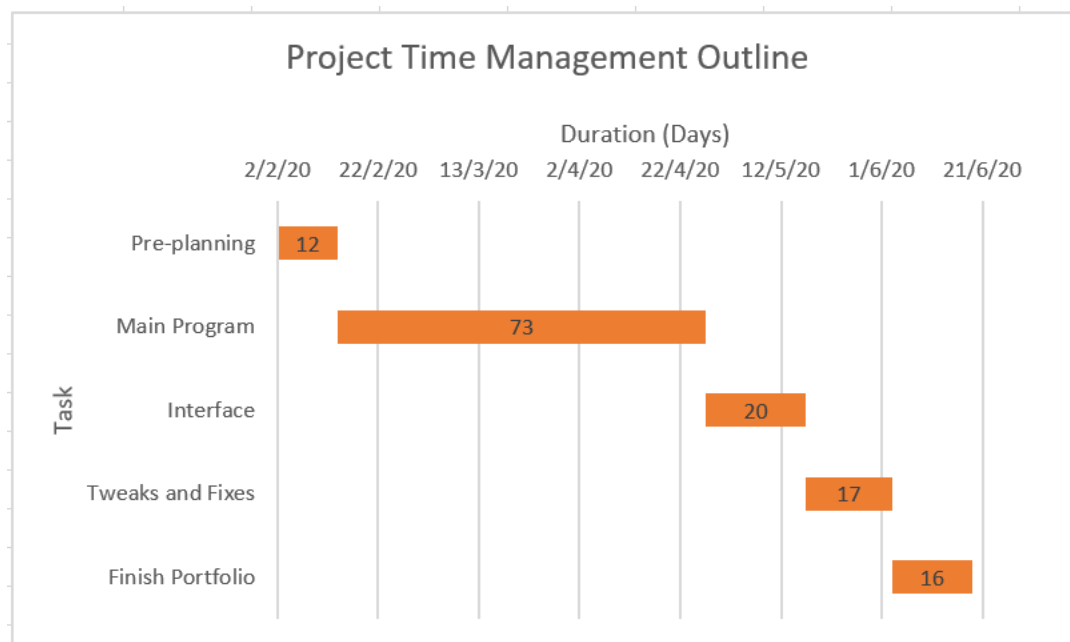
## Approach

I have decided to take a prototyping approach for this project.

The prototyping approach relies on progressive development and refinement on models (prototypes) which are then tested against the client's requirements. The client will give feedback and the developer will create a new prototype. This approach is effective for smaller budget and short time period projects.

I have chosen this approach because it works best with my style of working. The method I will use to develop my project will be to create a function or portion of the software e.g. the file sending function, I will then test it and making sure this prototype works. I will then move onto the next part. As I make larger changes that may affect the entire code, I will create new save files so nothing breaks. This approach will allow me to develop my project to the highest quality in the given time period.

## Time Management Outline



Task	Start Date	End Date	Duration (Days)
Pre-planning	2/2/20	14/2/20	12
Main Program	14/2/20	27/4/20	73
Interface	27/4/20	17/5/20	20
Tweaks and Fixes	17/5/20	3/6/20	17
Finish Portfolio	3/6/20	19/6/20	16

# Social, Ethical and Legal Issues

For my software project, I am developing a file transfer program that can be used to send files to different computers on the same network. With the development of this project, there involves a range of legal, social and ethical issues that must be faced.

## Social

I must consider a range of social issues for my project, leading to a highly ergonomically friendly software. My software should have a user-friendly interface. I must ensure the UI elements are all consistent throughout, the text is easy to read, and it is overall easy to use by keeping it simple and ensuring there are no unnecessary elements. For example, in a file transfer program, a “convert file” button would be irrelevant and confusing while a “send” button would be relevant and clear. The program should also be reliable at performing tasks without failure and be stable, not randomly crashing. There should be clear documentation outlining and explaining how to use the file transfer system or helpful messages throughout the program giving brief explanations of everything, adding to the overall ease of use. For example, in my software if there is a maximum file size capacity that can be transferred, then it should explicitly be communicated if someone tries to upload a file that is too big. Overall, the consideration of these social issues will lead to a high quality, ergonomically friendly product.

## Ethical

The development of software using networking also comes with a range of ethical issues that must be considered as it allows the potential access to multiple computers on the same internet connection. When developing my software, I must not use networking to perform inappropriate tasks such as hacking or controlling other computers; I also must ensure files are sent to the intended recipient with their permission by creating a request system or allowing them the option to not receive the file. I also must consider network traffic and make sure my system doesn't clog up the network bandwidth, slowing down other user's internet speeds. In my project, I will try to set up systems in place to deal with these issues.

## Legal

All throughout the development of the project I will be learning to code from different YouTube tutorials, websites and forums. Plagiarism is the copying of a developer's idea and passing it off as your own, and therefore a major legal factor to be considered when developing software based off what I have seen on websites. There are a couple ways around this:

1. I will change and adapt the code I find to work with my program.
2. If I do directly copy the code into the program, I must make sure to acknowledge the original developer in the documentation. Unless I asked the person, I would not legally be able to sell the code if there were modules of code written by other developers.

# Diagrams

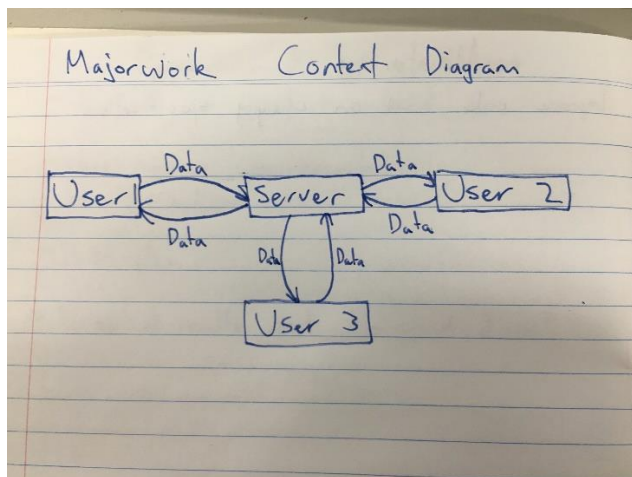
Since my software is a file transfer program, the most beneficial diagrams should show the data flow between different systems, therefore I chose to create a context diagram, simplifying the process to the bare minimum. I then created a data flow diagram, explaining the processes in more detail.

I have chosen not to use a system flowchart or structure chart because they highlight the processes within the program but do not highlight the data flow between external entities (which is the entire basis of my program).

## Context Diagram for file transfer program

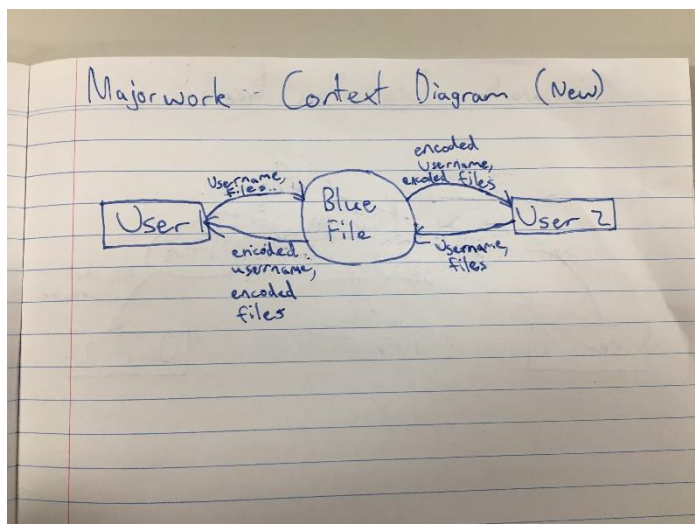
### Original

This was from my original idea, having multiple users connect to a server to exchange data.



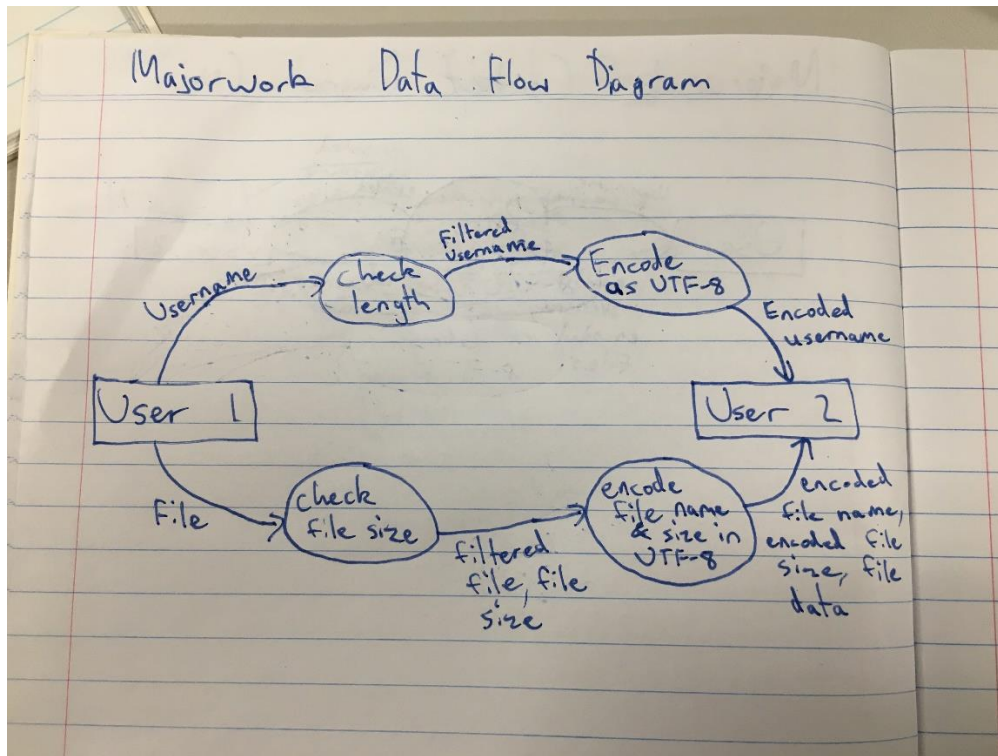
### New

This was from the current design; 2 users connect directly to each other to exchange data.

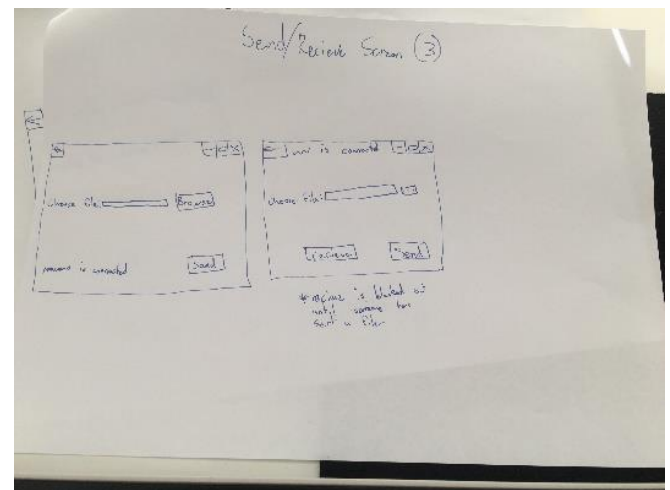
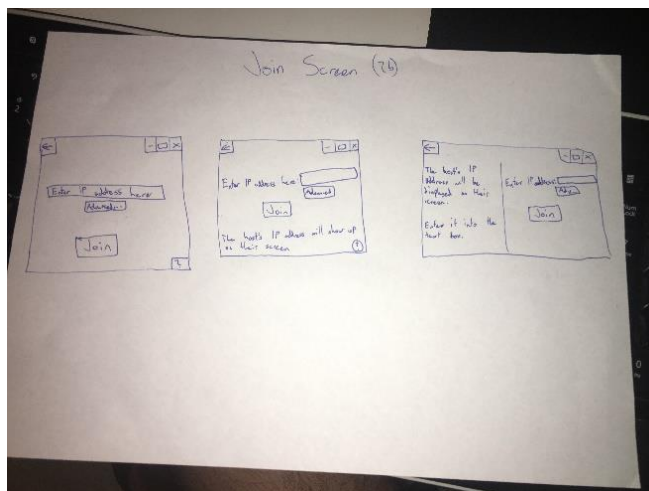
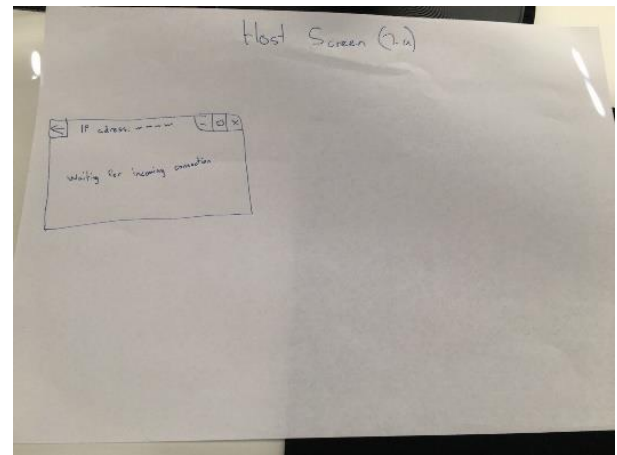
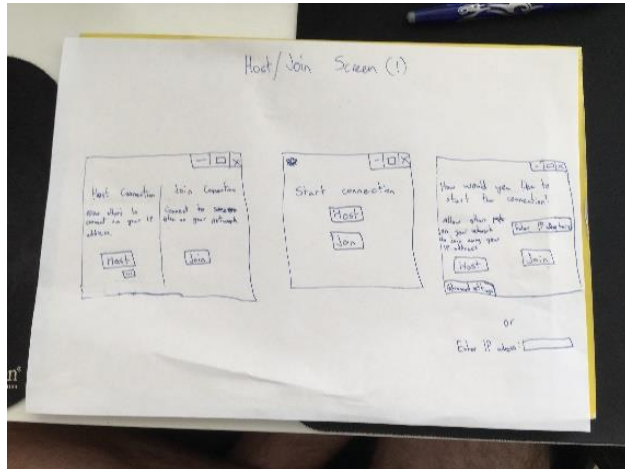


## Network Data Flow Diagram

This is from the current design.



## GUI drafts

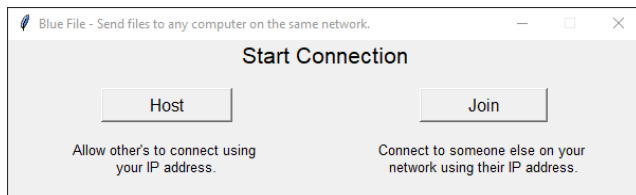




# Interface, Customisation and Compatibility Considerations

## Interface

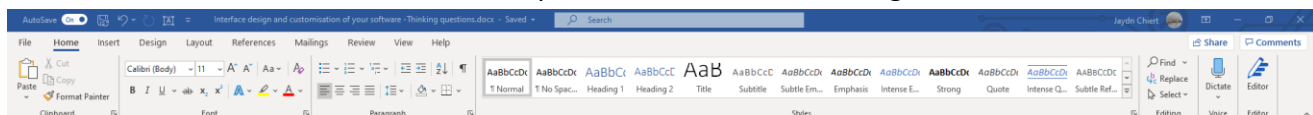
Here is an example of one draft screen in my project:



The most important part of an interface is the ability for the user to easily understand and use a screen. One aspect that draws users to a screen is the simplicity. If there are too many colours, words, or buttons on a screen it will confuse the user and they will likely not understand and therefore not use the program. My interface aims at keeping it the UI looking simple and easy to understand so I only have two buttons with a brief description of what they do for anyone that wouldn't understand what the join and host buttons do.

Another aspect of the screen is the consistency. On my UI, the title is in the middle of the screen, with the buttons evenly sized and spaced, as well as the text underneath evenly sized and spaced.

The Microsoft Word ribbon is an example of a successful GUI design:



Each page is correctly named with relevant information and buttons inside. In each page there are separated sub-categories, for example the font has everything relevant to do with editing the font will the clipboard has everything relevant to do with copying and pasting. It is simple and easy to understand which is why Microsoft Word is one of the most popular word processor software.

## Customisation

Most customisation in my software is driven by the programmer, such as the text font and size, window size and button size. None of these items are able to be customised by the user. The program will also use a default IP address and port, but I will allow the user to use a custom port under an advanced settings tab. The user is also able to choose which file they want to send and whether they want to host or join the connection.

I will use a standard GUI in my software as the program is for common/home/business users. This GUI will consist of only the information and options the user will need for example the ability to choose a file but opening the server will be a background task. The user will get feedback such as: “waiting for incoming connections” or “file could not be sent”. The actual processes will be in the background.

## Compatibility

Due to the method used for opening files, this program may be limited to the Windows Operating system. All hardware should be compatible, but it may take longer to upload a larger file on slower hard drives or slower internet connections.

When selecting the technology used, faster systems and internet connections will improve performance speed to an extent, but all users should be able to use the program regardless of the technology used.

# Logbook

17/12/19

- Started planning my software project and time management.
- My aim is to have my idea by the end of the first full week of the holidays.

25/12/19

- Asked my family if there was any software program that they would find useful in their lives. I was given a few suggestions but decided to do a bushfire tracker program using the RFS feeds. I ended up going with this idea because I thought it could be useful to a range of people and be interesting and a challenge to program.

26/12/19

- Started researching how to pull data from the internet (web scraping).
- Created a simple web scraping program, comparing GPU prices on Newegg, from a YouTube tutorial to get in order to understand it for myself.

<https://www.youtube.com/watch?v=XQgXKtPSzUI>

27/12/19

- Started coding my program. I grabbed the RFS major fire feed page just like the tutorial and set it up so I can start finding the different updates.

```
1 from urllib.request import urlopen as uReq
2 from bs4 import BeautifulSoup as soup
3
4 my_url = 'http://www.rfs.nsw.gov.au/feeds/major-Fire-Updates.xml'
5
6 #opening up a connection, grabbing the page
7 uClient = uReq(my_url)
8 page_html = uClient.read()
9 uClient.close()
10
11 #html parsing
12 page_soup = soup(page_html, "html.parser")
13
14 #grabs each update
15 updates = page_soup.findAll("item")
16
17 #finds each subset
18 for update in updates:
19
```

28/12/19

- I realised I was html parsing an xml file so I researched how to parse an xml file and fixed the issue. [http://www2.hawaii.edu/~takebaya/cent110/xml\\_parse/xml\\_parse.html](http://www2.hawaii.edu/~takebaya/cent110/xml_parse/xml_parse.html)

31/1/20

- Asked my teacher about my project to see if I can do my idea. He suggested I do something that will help the world

6/2/20

- I was thinking about how to implement my idea and researching how bushfire patterns are predicted and concluded my idea was going to be too complex for the amount of time I had so I decided to scrap it. I then looked at ideas other people had for their projects but couldn't find anything that gave me ideas. I then was just thinking about networking, which is the topic we were doing in maths and had an idea. I decided I wanted to use networking in my project, specifically make a program where people would be able to send files directly to other computers on the same network.
- I started researching how to use networking in Python.

7/2/20

- Got approval for my idea from my teacher. He also suggested to use a prototyping approach where I would develop the program overtime, adding and changing ideas as I got more familiar with what I was doing and the end result.
- I continued my research on networking in Python, following a tutorial to make a basic server and client to connect to it.

```
#Echo client
import socket

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)

print('Received', repr(data))
```

```
#Echo server
import socket

HOST = '127.0.0.1'
PORT = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

- I wasn't too sure what the end result would be but I knew the message hello world should come up. I opened the server file and then the client but for some reason the client closed straight away. The bell rung so I didn't get time to debug and fix the issue.

8/2/20

- I thought watching a YouTube tutorial might be easier and it was. I still wasn't sure of the problem from the previous day, but I did manage to setup a working client and server. A user would be able to send custom messages from the server to the client. Right now it was only setup so it would work on the host computer because I used a local IP address.

## Client

```
import socket

HEADERSIZE = 10

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((socket.gethostname(), 1234)) #connects to server

while True:
    full_msg = ''
    new_msg = True

    while True:
        msg = s.recv(16) #receives message with buffer size of 16 bytes at a time
        if new_msg:
            print(f"new message length: {msg[:HEADERSIZE]}")
            msglen = int(msg[:HEADERSIZE])
            new_msg = False

        full_msg += msg.decode("utf-8")

    if len(full_msg) - HEADERSIZE == msglen:
        print("full msg recvd")
        print(full_msg[HEADERSIZE:])
        new_msg = True
        full_msg = ''

    print(msg.decode("utf-8"))
```

## Server

```
import socket

HEADERSIZE = 10

#create the socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #AF_INET = ipv4, SOCK_STREAM = TCP
s.bind((socket.gethostname(), 1234)) #binds socket to a port
s.listen(5) #listens for incoming connections

print("The server is open")

while True:
    #our endpoint knows about the OTHER endpoint
    clientsocket, address = s.accept()
    print(f"Connection from {address} has been established")

    msg = "Welcome to the server!" #message to be sent to client
    msg = f'{len(msg):<{HEADERSIZE}}' + msg #combines headersize and message

    clientsocket.send(bytes(msg, "utf-8")) #sends message to client in utf-8 code

    #send custom message to client
    while True:
        msg = input("send msg... ")
        msg = f'{len(msg):<{HEADERSIZE}}' + msg
        clientsocket.send(bytes(msg, "utf-8"))
```

- I also added a few notes in the code to explain what everything does, so I don't forget. Most of the code I used was from the YouTube tutorial but as I learn more, I will change it to suit my own needs and fulfil my goals.

9/2/20

- I looked back over the previous code from the previous tutorial and found the problem, now that I understand how to use sockets a bit better. As soon as the message was sent to the client, it terminated its connection to the server.

14/2/20

- Started learning how to convert objects in python to bytes in order to send them over using the pickle module.
- <https://www.youtube.com/watch?v=WM1z8soch0Q&t=193s>

19/2/20

- Continued watching the video to learn and successfully converted a dictionary into bytes, sent it, received and decoded it.

## Server:

```
#converts message (dictionary) to bytes
d = {1: "Hey", 2: "There"}
msg = pickle.dumps(d)

msg = bytes(f'{len(msg):<{HEADERSIZE}}', "utf-8") + msg #combines headersize and message and converts to bytes

#sends message to client
clientsocket.send(msg)
```

## Client:

```
d = pickle.loads(full_msg[HEADERSIZE:])
print(d)
```

- Started watching a video in order to learn how to create a chatroom. This should help me learn how to send messages between two clients through the server, as opposed to sending from the server to a client.
- [https://www.youtube.com/watch?v=CV7\\_stUWvBQ](https://www.youtube.com/watch?v=CV7_stUWvBQ)

21/2/20

- Finished learning how to create a chatroom server and was ready to start the client next time so I decided to test it and there were a few errors I didn't have time to fix.

24/2/20

- Spent time fixing errors in code. There were only small typos where I had to change up some letters. I also went through the code, adding notes about what everything does to make sure I don't forget as well as fully understand it for when I start programming my own program.
- Started learning to create the client.

30/2/20

- Created the client and had a working chatroom. The next step was to use this knowledge to create my file sharing system.
- I coded the server setup, the client connects and then the server receives and accepts the connection.

### Client

```
import socket
import select
import os

HEADER_LENGTH = 10 #header is used to identify message, its source and destination

IP = "127.0.0.1"
PORT = 1234

#connect to server
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((IP, PORT))
client_socket.setblocking(False)
```

### Server

```
import socket
import select

IP = '127.0.0.1'
PORT = 1234

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #IPv4, TCP
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) #allows socket to reuse address

server_socket.bind((IP, PORT)) #binds IP and PORT

server_socket.listen() #listens for incoming connections

#manages list of clients
sockets_list = [server_socket]

clients = {}

while True:
    #accepts connection from client
    client_socket, client_address = server_socket.accept()
    print(f"Connection from {client_address} has been recieved")
```

2/3/20

- My plan was to build the program as a shell and then fill in everything in the middle so the next step was to do basic file transferring from the client to the server. My first major hall I had to overcome was accessing files. I tried using a command line:

`with open(file, read)`

But this only opened files in the same directory the program was in. I then created a test directory and inputting the directory in the `'file'` spot but an access denied message showed up.

I knew there was a module called `os` which could be used to run commands on an operating system level, therefore it should be able to open any file directory. I tested a command `os.listdir()` which should list all files in the directory.

```
while True:
    path = input("File directory: ")
    print(path)
    print()
    files = os.listdir(path)
    print(files)
```

The code I used successfully listed all files in the directory. Now I wanted to be able to open and read the files.

3/3/20

- Worked on sending and receiving files. I was able to successful send the .txt file but was not able to receive it. This is the current code for receiving a file:

```
#recieve file
recv_data = input("Click enter to recv: ")

if recv_data:
    file = server_socket.recv(16)

    print("file recieved")
    os.open(file, os.O_RDONLY)
    data = os.read(recv_data, 1024)

    print(f"file contains: {data}")
```

- I looked up a basic file transfer system and I will try something slightly more similar to that. [https://linuxhint.com/python\\_socket\\_file\\_transfer\\_send/](https://linuxhint.com/python_socket_file_transfer_send/). In this program, after the file is received it writes that file as a new file.

4/3/20

- We had a chance to work on our projects in class so my main focus was trying to successfully download the file that was sent. What I wanted to do was use the file it received to write a new file in a new location.

<https://steelkiwi.com/blog/working-tcp-sockets/>

5/3/20

- I had access to 2 computers today, so I tried putting in the ipv4 address for one and the computers successfully connected. Next, I tried sending over a file and the file received but there was an error when trying to actually write a new file with the data inside. I didn't have time to fix it, but I will try next time.

12/3/20

- I had time to work on the program today. I started researching different methods to try and fix the issue. I think the problem was that I was maybe sending the entire file directory over to the computer than the file itself. Another reason could have been that since I was using OS commands to write the file, it may have been blocking it. I had some ideas after this to fix it and I will try using a different method of getting and opening the file to try fix it as well as a different method to open the received data. I didn't have much time left to do this, so I started turning my code into their own functions, a send function and receive function. This would be important in the future as I will be able to copy my code to other programs or call it in multiple sections of the code. I also created a send\_username function because in the future I want to send a username to the server.

```
def send_username():
    choose_username = input("Username: ")
    username = choose_username.encode("utf-8")
    client_socket.send(username)

def receive_data():
    #receives data in sets of 16bytes until all has been recieved.
    while True:
        recv_data = client_socket.recv(16)
        if not data:
            break
        print(f"file recieved: {recv_data}")

        new_name = input("New file name: ")
        new_path = input("Where do you want to save the file: ")

        save = new_path + '\\' + new_name
        new_file = os.open(save, os.O_RDWR | os.O_CREAT)
        os.write(new_file, recv_data)

    print("file saved")

def send_data():
    path = input("File directory: ")
    print(path)
    files = os.open(path, os.O_RDWR) #opens the file as read and write
    data = os.read(files, 16) #reads file
    print(f"File contains: {data}")

    #sends file
    client_socket.send(data)
    print("File sent")
```



16/3/20

- I found a forum with some code that I can try use in a similar way to mine to fix the issue: <https://stackoverflow.com/questions/46979262/python-socket-how-to-send-a-file-to-another-computer-which-is-on-a-different-ne>
- I tested a different method for opening, reading and sending data:

```
def send_data():
    path = input("File directory: ")
    print(path)
    files = open(path, "rb") #opens the file as read and write
    print(f"files = {files}")
    file_size = os.path.getsize(path) #reads file
    print(f"file_size = {file_size}")
    data = files.read(file_size)

    print(f"File contains: {data}")

    #sends file
    client_socket.send(data)

    files.close()

    print("File sent")
```

- I then also changed the way a file is opened and written after it is received:

```
def recieve_data():
    #recieves data in sets of 16bytes until all has been recieved.
    while True:
        recv_data = client_socket.recv(16)
        if not data:
            break

        print(f"file recieved: {recv_data}")

        new_name = input("New file name: ")
        new_path = input("Where do you want to save the file: ")

        save = new_path + '\\' + new_name
        new_file = os.open(save, os.O_RDWR | os.O_CREAT)
        os.write(new_file, recv_data)

    print("file saved")
```

- The while loop didn't really work as intended. My aim wasn't to create a buffer to how much data can be received at once yet, the main intention was to save data once it came through so I removed the while loop and left the `recv_data = client_socket.recv(16)` to test with. I used 2 different computers to test the program and it worked. I was able to successfully send data between 2 systems from a client to a server.
- My next step was to set create a buffer for the data received. After that I would have to reconfigure the server to handle multiple clients and forward the data to the intended client after it receives the data.

18/3/20

- I re-wrote the WHILE loop; this time it should work:

```
def recieve_data():
    #recieves data in sets of 16bytes until all has been recieved.
    input_recv = input("Press enter to recieve data:") #only runs if enter is clicked
    while input_recv:
        recv = client_socket.recv(16)
        data = recv

        #everytime data is recieved add it onto the data variable
        if recv:
            while recv:
                recv = client_socket.recv(16)
                data += recv
            #if there is no more data being recieved, break loop
        else:
            break
```

- There was an error with the data variable because it hadn't been previously assigned. I tried adding a `Data = None` assignment before the loop. The code didn't work, there was no data that was received. I tried changing the WHILE line to `while input_recv == ""`. An error then appeared saying that the "socket" object was not callable, so I tested the code not in the specific function which didn't work. I then tried changing the WHILE statement to an IF statement, but it didn't work. Finally I took a good look at my code and I realised I forgot to add the `.recv` to the `recv = client_socket.recv(16)`. I tried testing but it just got stuck in an endless loop. I tried many alterations of the code but they didn't work.

11/4/20

- I tried many different fixes for about an hour, but I couldn't figure out what was wrong with the code. When starting the function there would be no output. I then had the idea to use debug statements after every step to see what it would get up to:

```
if input_recv == "a":
    data = None
    print("data=None complete")
    while True:
        recv = client_socket.recv(16)
        print("1st recv")
        data = recv
        print("data = recv complete")
        #everytime data is recieved add it onto the data variable
        if recv:
            while recv:
                recv = client_socket.recv(16)
                print("recv2 complete")
                data += recv
                print("data +=recv complete")
            #if there is no more data being recieved, break loop
        else:
            print("continue complete")
            continue
```

- This was the output:

```
data=None complete
1st recv
data = recv complete
recv2 complete
data +=recv complete
recv2 complete
data +=recv complete
recv2 complete
data +=recv complete
recv2 complete
data +=recv complete
```

- This meant that it received all the data but couldn't move to the else statement, therefore creating an infinite loop. I tried isolating that portion of the code and only performing it and for some reason it worked but when translating it into the program it didn't I also tried mixing around break, continue and pass statements but they didn't work. I then decided to completely redo the code using this website:

<https://stackoverflow.com/questions/17667903/python-socket-receive-large-amount-of-data> with a variation of this code:

```
while True:
    chunk = s.recv(10000)
    if not chunk:
        break
    fragments.append(chunk)

print "".join(fragments)
```

- This was my variation:

```
data = []
print("data=None complete")
while True:
    recv = client_socket.recv(16)
    print("recieved data")
    #everytime data is recieved add it on
    if not recv:
        print("no recv data left")
        break
    data.append(recv)
    print("data.append(recv) complete")

print(f"file recieved: {data}")
```

- It still didn't work, and I was wondering why it stayed in the loop. I then found these websites on loop errors and receiving large amounts of data in python:
  - o <https://stackoverflow.com/questions/14201147/send-receive-infinite-loop-with-socketserver-python-connectionabortederror>
  - o <https://stackoverflow.com/questions/17667903/python-socket-receive-large-amount-of-data>
- The actual websites didn't help very much but it did help me make a realisation that the client might be continuously sending a stream of blank data to the server so the server will always be receiving data and won't be able to trigger the *if not recv:* line. I then wondered how I could fix this, as I thought back to one of the first YouTube videos I watched on sockets:
  - o [https://www.youtube.com/watch?v=Lbfe3-v7yE0&list=PLQVvva0QuDdzLB\\_0JSTTcl8E8jsJLhR5](https://www.youtube.com/watch?v=Lbfe3-v7yE0&list=PLQVvva0QuDdzLB_0JSTTcl8E8jsJLhR5)
- I remember after he sent all the data across he closed the client instantly (or something similar) so I added the line *client\_socket.close()* in the client program right after sending the data. I then tested this and it worked (except after I will have to change the receive code so it doesn't receive as a list, rather a string so I will most likely change the code back to what I originally had). My next goal will be to figure out how I can keep the

client open after sending data. After spending about 3 hours today (excluding previous days) figuring out the issue, all I had to add was 1 line of code.

13/4/20

- Last time I figured out that I had to close the client's connection to stop sending data, but I don't want to have the client's connection closed every time data is sent. I want to try obtaining the message length and send that with the file at the beginning so the server knows how much data it must receive. To do this, I created a new program and copied the send data function to make it easier so I would not need to go through all the other stages. I started by getting the file size and combining it and the file into one variable but I couldn't put two different data types (bytes and integer) into one variable so I then tried converting the size in bytes using `bytes(file_size)` but when testing it on a 71 byte file, it created 71 extra bytes which wouldn't work. I then found a website talking about a module called pickle, which can be used to convert any object into bytes: <https://pythonprogramming.net/pickle-objects-sockets-tutorial-python-3/?completed=/buffering-streaming-data-sockets-tutorial-python-3/>
- I tested turning it into bytes and it worked. I then turned it back into an integer which worked as well. I implemented the conversion from an integer to bytes into the code.

20/4/20

- I was starting to think of ways to handle for multiple clients and while I was thinking I had the idea to instead of needing a server and clients connecting to it, maybe I could create a system where you could connect one person to a host. This would not only make the code easier, but it may be less confusing for a normal user because they won't have to set up a server running in the background. It will also be faster as it is sending the data directly to the user, rather than through the server to user. Maybe when the program starts up, you can get an option to host or join a connection and then choose to send or receive the data. This would mean you could only send and receive to one person at a time, but I do not think this is a problem. I could also set it up to allow users to go back screens to choose whether to be host or join. To join, the users would just have to put in the host's IP which I could show the host.  
Maybe in the future if there's more time I could set up a system to handle for multiple clients and identify and sort data to who it is intended for; it just add so many more layers of complexity and code which I may not have time for.
- As it had been a while since last time, I then realised I still needed to implement the proper receiving system where it would download the file based on how many bytes were sent. While doing this, I remembered that one line of code was `client_socket.setblocking(False)`. I wasn't too sure what it did, but it was in one of the tutorials I watched for handling multiple clients. Since I have decided for now not to handle for multiple clients I was wondering if it was still useful so I found this site: <https://kite.com/python/docs/socket.socket.setblocking>. It basically said that if

setblocking is false, it doesn't calculate some exceptions such as what to do if there is no data being sent. Removing this line allows it so it will be able to tell when no data is sent. I won't need to worry about closing the client connection or sending the data size.

- Later today, I started putting this plan into action by creating an all in one program where it would create a server or client based on a text input. I started by creating functions that would create a server or client system. I will be able to call these later. I then copied over the send and receive functions from the other program and duplicated then modified the send function so it would work for both the server and client. The receive function should work for both. I then added a start loop that should call all the functions.

```
Start = 1
while Start == 1:

    setup = input("Host or join ")

    if setup == "host":
        server()

        mode = input("Send or Recieve ")

        if mode == "send":
            send_data_server()

        if mode == "recieve":
            recieve_data()

    if setup == "join":
        client()

        mode = input("Send or Recieve ")

        if mode == "send":
            send_data_client()

        if mode == "recieve":
            recieve_data()
```

- I tested it but it didn't work. I remembered that the socket variables must be global and I had localised them so the send and receive function weren't able to find them. I then removed the server and client functions and just copied the code in their respective places.

```
if setup == "host":
    #setup server
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #IPV4, TCP
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) #allows socket to reuse address

    server_socket.bind((IP, PORT)) #binds IP and PORT
    print(f"server is open on {IP}:{PORT}")

    #Establish connection to client
    server_socket.listen() #listens for incoming connections
    client_socket, client_address = server_socket.accept() #allow for incoming connections
    print(f"Connection from {client_address} has been established")

    mode = input("Send or Recieve ")

    if mode == "send":
        send_data_server()

    if mode == "recieve":
        recieve_data()

if setup == "join":
    #setup client
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #setup socket
    client_socket.connect((IP, PORT)) #connect to server

    print(f"connected to server hosted on {IP}:{PORT}")
```

- I was thinking about the next steps in my project and decided that after testing if this all works, I should set up a GUI. It is earlier than I planned in development but it should aid in anything else I need to introduce as some code will need to be modified to fit the GUI. Other things that will need to be introduced are IP input, visible username, potentially

handling for multiple clients, as well as exception errors. I think most of these will be easiest to introduce after a GUI is developed.

21/4/20

- As I was testing, I found that I didn't need to create separate send functions for the client and server so I turned it into one. I also found that my idea about setblocking may have been wrong so I will probably have to go back to my original idea. Both the client and server are able to send currently.
- I edited my send function to send the file size at the beginning as part of a 10 character header:

```
def send_data(HeaderLength = 10):  
    path = input("File directory: ")  
    print(path)  
    files = open(path, "rb") #opens the file as read and write  
    print(f"files = {files}")  
  
    file_size = os.path.getsize(path) #obtains file size  
    print(f"file_size = {file_size}")  
  
    data = files.read(file_size) #reads file  
    print(f"File contains: {data}")  
  
    file_size_bytes = pickle.dumps(file_size) #convert file size to bytes  
    print(f"file_size_bytes = {file_size_bytes}")  
  
    full_send = bytes(f"{file_size:<{HeaderLength}}", "utf-8") + data #file size is sent as a string and left aligned as part of the header  
    print(f"sent: {full_send}")  
    #sends file  
    client_socket.send(full_send)  
  
    files.close()  
    print("File sent")
```

22/4/20

- I worked on the receive function using my knowledge from all past learning tools including trial and error, YouTube and websites. This is what I came up with:

```
def recieve_data(Headerlength = 10):  
    #recieves data in sets of 16bytes until all has been recieved.  
    data = bytes()  
  
    while True:  
        Header = client_socket.recv(Headerlength) #recieve header  
        print(f"header: {Header}")  
  
        file_size = int(Header[:Headerlength]) #convert header into integer  
        while True:  
            if len(data) != file_size:  
                new_data = client_socket.recv(16) #recieve next set of data  
                data += new_data #add it onto the data variable  
  
            else:  
                break  
        break
```

- It first receives the header which has the file size as a string. I then convert it into an integer and continue receiving data until it is equal to the file size. When testing, I didn't break the first *while True* loop which meant it received the first bit of the message and tried converting it into an integer as well. I don't think I need that first *while True* because it only needs to receive the header once.
- I tried removing the first *while True* loop and it worked. I was so excited. Next, I will start work on the GUI as planned.

24/4/20

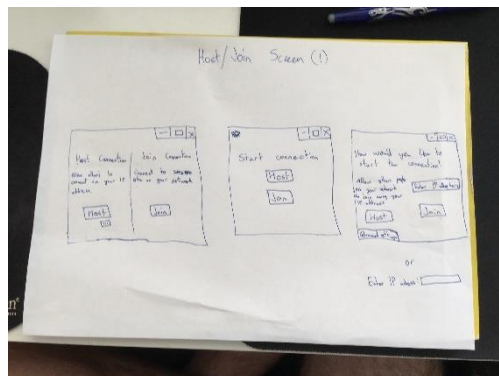
- I started learning how to create a GUI using the Tkinter module in python using this tutorial:  
<https://www.youtube.com/watch?v=yQSEXcf6s2I&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV&index=1>

26/4/20

- I continued learning how to create GUI programs in python by continuing to watch the YouTube series. I also learnt that global variables can be created with functions which may end up being useful in the future.

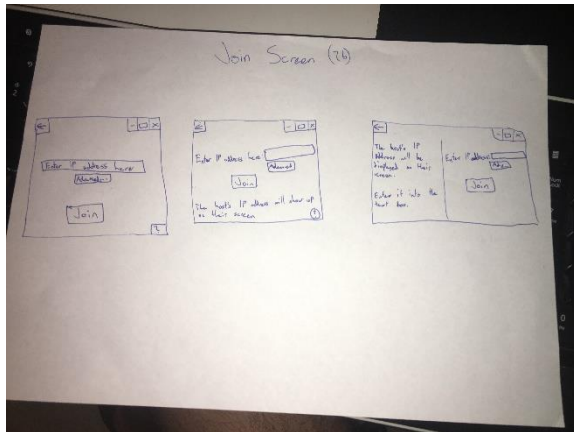
27/4/20

- I started designing ideas on paper for my GUI. I googled images of Host/Join GUIs to get inspiration and see how others have designed them. I came up with 3 potential designs that I will survey my friends and family to see what they would prefer best in order to come up with my decision:



- Some feedback I received included, the use of the first design if I put the buttons up the top and the explanation at the bottom as well as using the cog or advanced setting rather than the dots. I was also told that people don't necessarily know what an IP address is so it should be explained in the next page. (I was going to show the host their IP address in the next page anyway.) The second option was also preferred a lot of the time because it was the most simple and easy to look at and thought there should be a start connection heading up the top.

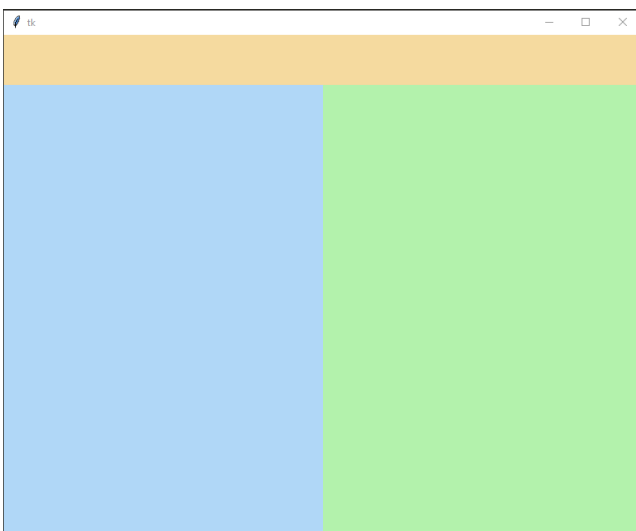
- I then designed the join screen:



- The middle design was the most liked because it gave some information but didn't take up the screen. Opinions were mixed about whether text should be in the text box or next to.

28/4/20

- I started work on my GUI in a separate file to simplify the process, using stubs in place of the actual function. I will implement the actual functions after the GUI is completed.
- I started work on the first host/join screen by creating frames. This will allow me to easily organise the different widgets, and buttons by choosing which frame to place the widgets in. I used colours as guides to where the frames are, but they will be removed for the final product. The top one will be for the title and back button; the left will be for the host and the right will be to join.



```
#Top frame
top_frame = tkinter.Frame(root, bg = "#f5da9f")
top_frame.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)

#Left frame
left_frame = tkinter.Frame(root, bg = "#b0d7f7")
left_frame.place(relx = 0, rely = 0.1, relwidth = 0.5, relheight = 1)

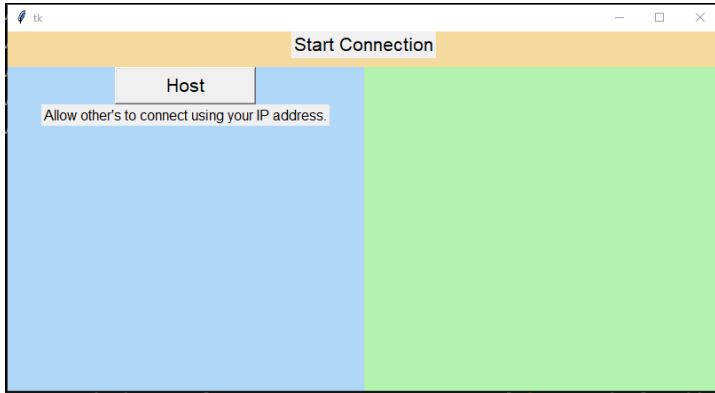
#Right frame
right_frame = tkinter.Frame(root, bg = "#b3f2ac")
right_frame.place(relx = 0.5, rely = 0.1, relwidth = 0.5, relheight = 1)
```

- I then started testing out methods to change the window size; the standard method is a `root.geometry("x*y")` but I had seen others use a `.canvas(width = x, height = y)`. The canvas is like a drawing tool that anything can be placed in so I used that and defined

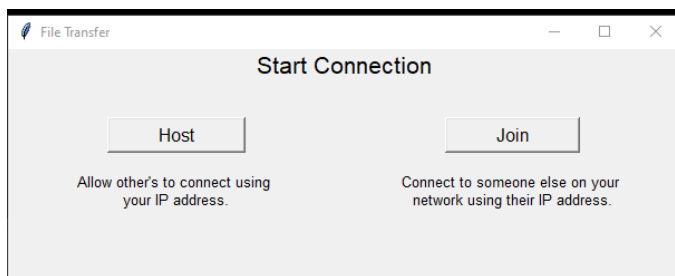


global variables to place into it. I tested out a couple sizes but I will only know what size I want once I add in some buttons.

- I then started trying to add in some buttons, playing around with window sizes, placement, text sizes. Here is one of those drafts:



- After doing this one, I decided to remove the colours because I felt like I wasn't getting an accurate representation of what I wanted and eventually I came up with this:



1/5/20

- I added code that deletes everything from the page except for the frames when clicking host, in a *Page\_2()* function. I then would need to add the GUI for the host page. This code would work for any page if I copied it into the function. I was thinking about this approach when I realised the frames are not in an ideal position for all pages so I might include a switching page which deletes everything including the frames off the page but it was getting late so I will do that next time.

3/5/20

- I deleted the *Page\_2()* function and replaced it with a *switch()* function where I will have a variable "number" inside the function that will determine which page to switch to after. This is the code:

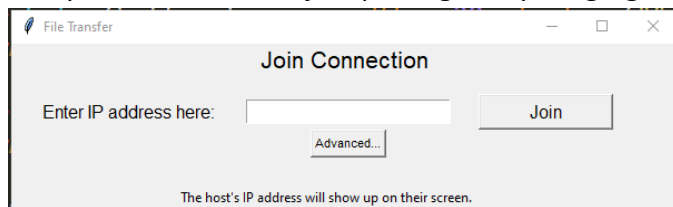
```
def switch(number):
    for frames in root.winfo_children():
        frames.destroy()

    if number == page_1:
        page_1()
    if number == page_2a:
        page_2a()
    if number == page_2b:
        page_2b()
    if number == page_3:
        page_3()
```

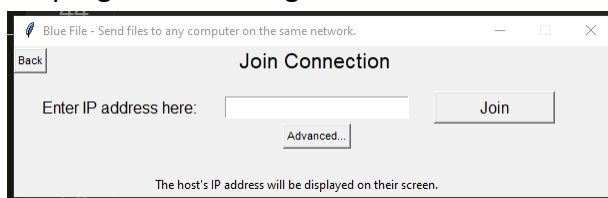
- After coding this I realised it could be simplified and would have to require me to add an *if number = page\_x*. In the button command line, I added *command = lambda: switch(page\_2a)*. I changed the switch function to:

```
def switch(number):
    for frames in root.winfo_children():
        frames.destroy()
    number()
```

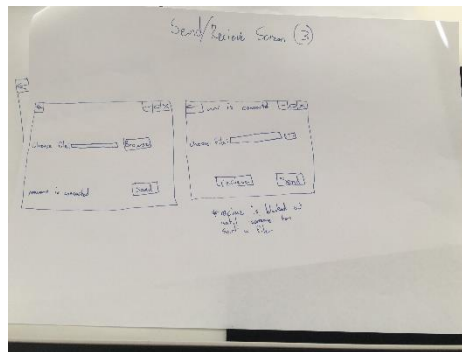
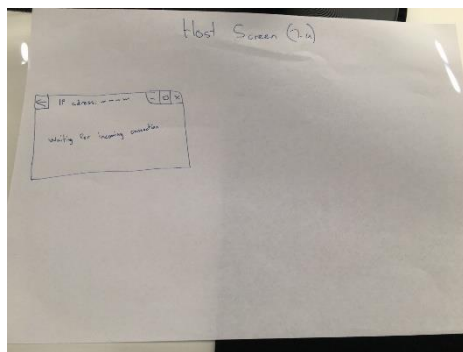
- I then created the join screen, and unlike the first screen, I used *.grid* for placement rather than the *.pack* because it would be easier to create the layout in a grid like system rather than just placing everything right underneath each other.



- I then added a back button which goes back to the previous page and a name for the program with a tagline:



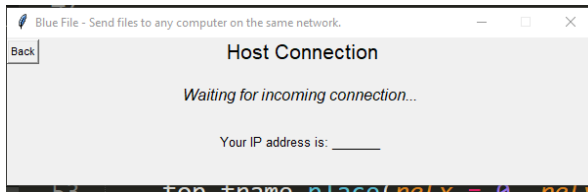
- I also designed drafts for my last two screens in the program:



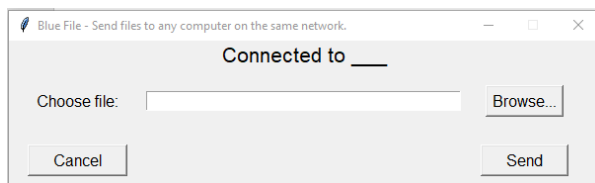
- The main feedback I got was to have a title for each screen. There were also mixed opinions for whether there should be a receive button or a pop up when an item is being received. I will play around with this during while programming.

4/5/20

- I added the host's screen using the .pack method in the bottom frame and a .grid() in the top frame as they were the most simple options for what I wanted:



- This page will in future display the users IP address and automatically switch to the next screen when a client connects to the server.
- Later today, I added the final page:



- To get the browse button working, I had to import a separate module, called tkinter.filedialog, and create a function to get the file. Once the file has been selected it copies the directory, as a string, into the entry box. Right now, the send button does nothing but once I copy the send function into this program, send that selected file. I also had to setup the file entry box as a global variable if I wanted to use it in a different function. The browse file function is:

```
def open_file():  
    #open file browser  
    file = tkinter.filedialog.askopenfilename(initialdir = "", title = "Select a file")  
    #insert file directory into the file entry field  
    file_entry.insert(0, file)
```

5/5/20

- I had a pretty large chunk of time today, so I completed quite a few tasks. Firstly, some screens had 'back' buttons while others had 'cancel' buttons which was inconsistent, so I changed every screen to a 'cancel' button in the bottom left. It took some time to adjust these so they would all be in line with each other.
- I then had to fix up the title placement because some of them had changed when removing the back button due my use of the grid system and removing the first column. I used the pack system for the title so they would all be centred the same.
- The GUI was pretty much as complete as it could be, so it was time to combine the two programs by implementing the file transfer program into the GUI to make it functional. To do this I first created a close connection sequence which first blocks the socket from being able to send and receive data and then destroys the socket. The program can tell whether it's the client or server socket by passing either the client\_socket or server\_socket variable into the function. I tested it and it worked for the client but not

for the server. This was because the `server_socket` was not a global variable. I changed it into a global variable, but it came up with a different error message. It was not able to shut down the server socket. After some testing, I found that I had to shut down its connection to `client_socket` by using the `client_socket` variable rather than the `server_socket` variable and then close the server after. This is the code I came up with:

```
def close_connection(mode):  
    client_socket.shutdown(socket.SHUT_RDWR)  
    print("shutdown")  
    mode.close()  
    print("close")
```

- I then created implemented this function into an exit and cancel function:

```
def cancel():  
    #close_connection()  
    switch(page_1)  
  
def exit():  
    #close_connection()  
    root.destroy()
```

- I then had to call the exit function when clicking the 'x' button to close the window:

```
root.protocol("WM_DELETE_WINDOW", exit)
```

- I then copied all the functions from the main program into the GUI program. Now I needed to properly implement them by calling them at the appropriate time. I opened to the Join Connection page to first implement the ability to get a persons IP address from what they type in the entry and then connect to the server. I added the first line in the `client_setup` function to do this. I also had to change the `enter_ip` variable in page 2b to a global variable to be able to use it in this function. I then added an exception error in case the client cannot connect or find the IP. Right now, it only prints the error in the console, but I will edit this later on:

```
def client_setup():  
    #get IP from enter_ip entry  
    IP = enter_ip.get()  
  
    try:  
        #setup client  
        print(IP)  
        global client_socket  
        client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) #setup socket  
        client_socket.connect((IP, PORT)) #connect to server  
    except:  
        print("ERROR: Cannot connect to given IP.")  
        return False  
  
    print(f"connected to server hosted on {IP}:{PORT}")
```

- After doing this, I realised I would need to get the users IP address to start up the server. Up until this point (except a couple times for testing), I have been using the standard "127.0.0.1" local IP. In the Tkinter documentation, I found that you could use `socket.gethostname()` to get a user's hostname (Jaydn's computer) as a string and `socket.gethostbyname()` to convert it into the proper IPV4 format as a string (xxx.xxx.xxx.xxx):

```
#Defines socket variables
hostname = socket.gethostname()
IP = socket.gethostbyname(hostname)
PORT = 1234
Headerlength = 10
```

- After I get everything working, I'll create the ability to customise the Host's IP address and port under an advanced settings button. I then looked up a list of ports and how they work. The port 32561 was unassigned so I will use this as the default port, but I will give the ability to change it if it doesn't work.
- I then tried implementing the server\_setup function into the program. I added an exception error which currently just returns False if it doesn't work. I also added a switch to page 3 at the end when it works. I tried getting it to work by clicking the host button, then opening the "waiting for connection" page, then setting up the server by calling the setup\_server function. When testing, the screen froze on the first page until it found a connection. I realised this was because I was wanting the program to do two tasks at once: display the waiting screen and wait for an incoming connection. I knew python programs could only do one thing at a time unless I used a threading module to control the threads used on the CPU. I didn't actually know how to use it and it was getting late so I left it for the night and will learn about the threading module tomorrow.

6/5/20

- Today I learnt how threading worked and how to implement the threading module into my program. Python programs use 1 out of the many threads on a CPU core. Each thread can perform 1 task at a time so using the threading module, I allocate another task to another thread so two tasks can be running at once.
  - <https://www.youtube.com/watch?v=ecKWiaHCEKs>
  - <https://www.youtube.com/watch?v=olYdb0DdGtM>
  - <https://www.youtube.com/watch?v=cdPZ1pJACMI>
  - <https://stackoverflow.com/questions/48180989/how-to-run-process-in-the-background-without-stopping-the-gui-in-tkinter>
- This the code I used:

```
def background(server_setup):
    th = threading.Thread(target=server_setup)
    th.start()
```

- I call this function at the bottom of the page 2a code after it has been setup. The server then runs in the background while other tasks can be performed.

8/5/20

- Today I properly implemented the exit and cancel functions that I had written earlier. When testing, I tried to close the windows with the 'X' button but for the first few screens it gave me an error saying there was no sockets open. This is because I called the close function globally rather than only calling it in the pages that had sockets open

so I fixed the issue and it worked. I also had to add an exception parameter in the `close_connection` function because if there is no client socket connected to a server socket, there is nothing for the server to shut down.

- Page 3 can either have a client or host using it and the exit and cancel functions required the `server_socket` or `client_socket` variables to be passed through them so I tried using an if... else... statement to detect which one was in use and passed it into the commands:

```
if server_socket:
    root.protocol("WM_DELETE_WINDOW", lambda: exit(server_socket))
    cancel_button = tkinter.Button(bottom_frame, text = "Cancel", font = ("", 12), padx = 20, command = lambda: cancel(server_socket))
else:
    root.protocol("WM_DELETE_WINDOW", lambda: exit(client_socket))
    cancel_button = tkinter.Button(bottom_frame, text = "Cancel", font = ("", 12), padx = 20, command = lambda: cancel(client_socket))
cancel_button.grid(column = 0, row = 2, pady = 28)
```

- I then added the switch to page 3 function in the client setup.

11/5/20

- I added in the send function. All I had to do for this was call it when clicking the send button as well as changing the way of getting the path from using an input to getting it straight from the file entry. I have not added the feedback on the GUI yet. I will do that after getting the receive function working.
- I was about to start implementing the receive function when I realised I needed a way to actually get the file name so in my send function I used a line: `os.path.basename(path)` to get the filename. I then added it in the `full_send` variable first next to the file size, separated by a colon (:). I have separated it with a colon because file names do not allow colons. This allows me to detect when the file name ends using the colon and where the file size is in the string.
- Later that night, I experimented with getting only the file name from the sent file. I started by using the same code in the receive function without the `"int"`. This was the original I tried to copy:

```
file_size = int(Header[:Headerlength]) #convert header into integer
```

- This is was the new one:

```
file_name = Header[:":"] #gets file name up to colon (:)
```

- It didn't work because that type of splitting only works with integers so I started googling ways I could split the string. Python has a `.split` function that can split up a string into a list until a certain set character and then I have to call the first item in the list:

```
name = str(full_send).split(":")[0]
```

- I then had to change the way to get the file size as I had now added extra characters to the header.
- I changed the `.split` to a `.partition` because a split breaks up the string at every point with the indicated character while a partition only separates it into the first part, then the splitting character, then the rest. I then implemented it into the code:

```

file_split = str(Header).partition(":") #splits file into 3 sections; name, size, file
file_name = file_split[0] #gets file name from split
print(f"file_name: {file_name}")

file_size = int(file_split[2][:Headerlength]) #convert header into integer
print(f"file_size: {file_size}")

data = bytes(file_split[2][Headerlength:]) #rest of the file after header becomes part of data

```

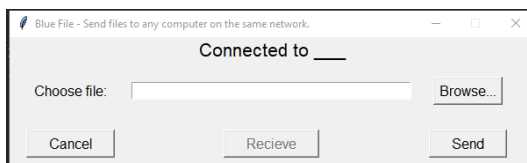
- I then had to figure out how I wanted to allow receiving of the data. To do this, I decided to add another thread for the receive function. This is because I wanted it to be constantly checking for received data in the background:

```

def background_recieve(recieve_data):
    th2 = threading.Thread(target = recieve_data)
    th2.start()

```

- I then called this function at the end of page 3. I then decided to use a receive button so the user can choose to receive a file. I added this in the middle and disabled it. It will become enabled once the receive function detects data:



- To enable the button, I added a receiving variable that would change to true after it detects data has been received. It then activates an IF statement to change the state of the button as well as allow the button to cause a receive clicked variable to become true, activating the receive function:

```

Header = client_socket.recv(4096) #recieve header
print(f"header: {Header}")

recieving = True

if recieving == True:
    recieve_button.config(state = "normal", command = change_recieve)
    recieve_button.wait_variable(recieve_clicked)
    print(recieve_clicked)

    if recieve_clicked == True:

```

- It did not work so I tried adding the *wait\_variable* as shown above. This meant it would wait until the button had been clicked, it did not work. I also tried changing the *change\_recieve* function testing global variables and mixing other things around but they did not work. It was getting late so I will continue tomorrow.

12/5/20

- The first thing I tried was removing the variable change at the bottom of the code to change it back to False, but it did not work. To fix it, I moved the button command down to where the button is first defined and removed the *wait\_variable*. I also changed the loops from IF to WHILE. It worked but then the file did not save, probably because the loop did not break or the save part was indented incorrectly.

15/5/20

- Over the past couple days I had been thinking about the issue on and off and I had an idea that I wanted to try to use to fix the issue:

```

global receiving
receiving = False

global receive_clicked
receive_clicked = False

file_name = ""

#receives data in sets of 16bytes until all has been recieved.
data = bytes()

Header = client_socket.recv(4096) #recieve header
print(f"header: {Header}")

receiving = True

while receiving == True:
    receive_button.config(state = "normal")
    print("state changed")

    while receive_clicked == True:
        print("receive button clicked")

```

- The main change in this was to define the receiving and receive\_clicked variables within the receive data function as a global variable (that way I could call them from another function. I also added the change\_receive function into the code where I originally defined the button. I then defined the file\_name and data variables before the receiving properly started. I then changed the part where it continually receives data until there is none left:

```

while True:
    if len(data) >= file_size: #continue receiving data until the length
        finish_receive()
        break
    else:
        print("else worked")
        new_data = client_socket.recv(4096) #receive next set of data
        print("new worked")
        data += new_data #add it onto the data variable
        print("new data")

```

- The finish\_receive function changes both the receiving and receive\_clicked variables in a separate function, ensuring both have been changed before the program continues. When running the program, it went through everything but had an issue writing the new file, due to a permission error but I did not have time to fix it, so I went to bed.

18/5/20

- I researched different ways to create a prompt; allowing the file to be run in administrator mode but none of them worked properly. I then tried converting the program into an exe file and that worked because I was able to set permissions for it upon start up but it would be too hard to test the program by converting it to an exe every time. I then tried opening the command prompt as administrator and running the program from there and that ended up working. This is how I will test from now on.
- I was facing an issue where the data would not be inserted into the test text file properly but I will deal with that next time.



20/5/20

- I was looking through my code wondering why the text file was not working properly. I thought that maybe when splitting the data and putting it into a list, it was messing up the formatting so I tried a different approach involving using the original data set before splitting and taking only the data:

```
data = Header[-file_size:]  
print(f"data: {data}")
```

- This essentially takes the amount of data equal to the file size from the end of the file so that the name and file size is not included.

20/5/20

- When looking back at this code, I realised it's not perfect because if the full file is not in the one set of data, it will take part of the file name and size. I then realised that I could get the length of the file name and the header and take that away from the beginning:

```
data = Header[(len(file_name)+Headerlength):]
```

- It added an empty byte beforehand, probably because of the colon between the name and header so I added a plus 1 on the end which worked:

```
data = Header[(len(file_name)+Headerlength):]
```

22/5/20

- I added an IF statement in the send file function to test if the file is bigger than the header length (which can store up to 9.9999999999gb). Right now, if it is higher, nothing will happen, but I will add a response.

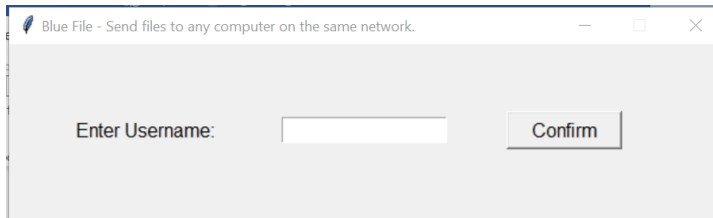
24/5/20

- I added a username variable and assigned it the computer's name (hostname). I also added a username variable (user) for the other person. Once connecting to the server, the usernames will be sent and received so a person knows who is who:

```
#send username  
client_socket.send(username)  
print("sent")  
  
#recieve username  
user = client_socket.recv(10).strip()  
print(user)
```

- I had written this in both the server and client functions and decided it would be more efficient if I just created it in its own function. I also had to globally change the user variable. The `.strip()` removes the empty header space.
- When testing, it did not work because sockets can only send bytes, so I had to encode it as utf-8 and then decode it once it had been received.

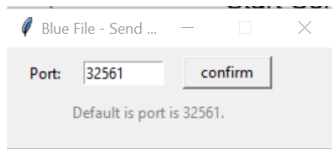
- I then created the username page. It involved slight tweaks here and there at different points to get the size and formatting correct; this is what I came up with:



- When I work on the feedback messages, I will only allow users to enter 10 characters, otherwise an error message will appear.

25/5/20

- The first thing I did was create a page for the advanced button; allowing users to change their port. Originally, I set the port parameters to between 0-65535 but the first 1023 are system reserved and I was not sure if they would respond well with the computer, so I allowed 1024-65535:



- I set up my first feedback message. I had to decide whether it would first be defined in the page or in the change port function. I ended up choosing to define it as a global variable within the page, that way I would be able to call it at any time during the program. I then called it if the port was not between 1024 and 65535:

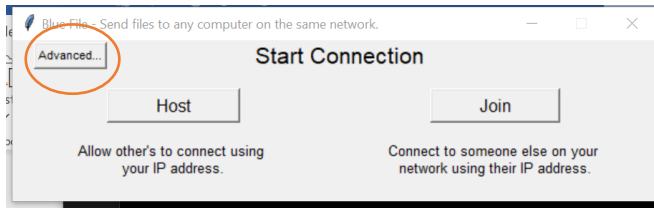
```
def change_port():
    global PORT

    #check if port is an integer
    try:
        PORT = int(advanced_entry.get())

        #check if port is a valid number
        if 1024 <= PORT <= 65535:
            advanced_window.destroy()
            print(PORT)
        else:
            port_error.grid(column = 0, row = 2, columnspan = 3)
            return False

    #if port is not an integer display error
    except:
        port_error.grid(column = 0, row = 2, columnspan = 3)
        return False
```

- Up until now, the port was only able to be changed in the join page, but I wanted it to be accessed when hosting the server as well. If I wanted it in the server page, I would have to close and restart the server to use a custom port so I decided to add it into the page where you can choose to host or join. This also meant that it was able to be accessed by the client and host, so I added it there but removed it from the join screen:



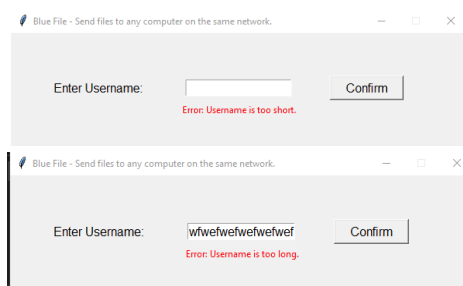
27/5/20

- Today I tasked myself to add feedback messages to every page of the program that needs. All the feedback messages were added the same way as the previous one where I would add it within the page as a global variable, calling it later in the program.
- I first added feedback regarding whether a username is too short or long. It first removes the other message, if it was already there, and then calls the required message:

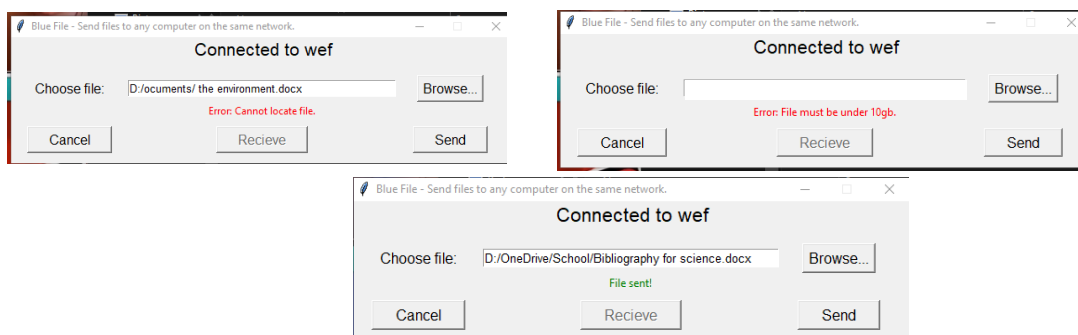
```
def get_username(number):
    global username
    username = username_entry.get()

    if len(username) > Headerlength:
        too_short.pack_forget()
        too_long.pack()
        return False

    if len(username) == 0:
        too_long.pack_forget()
        too_short.pack()
        return False
```



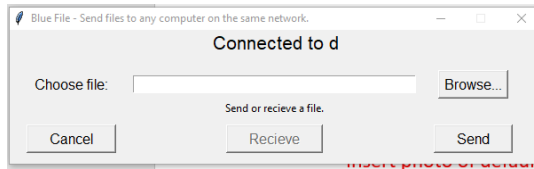
- I then created the messages for the join page. This would involve a saved file message, a 'file not found' message and a file is too big message. When adding the file not found message, I tried to use an ordinary exception, but it did not work because python already has a 'file not found' error message. I tested what would happen if I added the file not found error after the except command and it used my own code if that error happened.



- I also had to ensure that there was a way to remove any message already on the page without having to write every message so wrote a function for it which I could call at any point:

```
def remove_messages():
    for messages in bottom_frame.grid_slaves(column = 1, row = 2):
        messages.grid_forget()
```

- When testing this, the messages worked but when there were no messages, the layout messed up (probably something to do with the grid layout I used). To fix this, I set up a default 'send or receive file' message that would get replaced after another message was called.



1/6/20

- I went through the entire program today testing all the inputs and outputs. The first problem I had was when I would choose a file in the file browser and then replace it with another. The program would just add it after e.g. D:/folder/fileD:/folder2/file2. I then added a line of code that would delete everything in entry box before adding another file path.
- I then had a problem receiving large files. After some testing and adding debug statements, I realised that whenever a file was received it would have the colon splitting the header. This added extra characters that I had not accounted for. To fix this I added a -1 in when first setting up the data. This fixed the issue:

```
#Data = Header - file name - headerlength - 1 (gets rid of b' in file name)
data = Header[(len(file_name)+Headerlength-1):]
```

- I then removed all the print statements for the final file because they would not be seen by the user. It should slightly increase the speed of the program and called it done. Now I just had to finish my portfolio, ensuring I had enough documentation.

3/6/20

- I was thinking back to my code and when defining my feedback labels, I would place them and then remove them, only to call them later. Was there any point of placing them and then removing them in the first place? I should rather leave them defined and call them later, so I tested it and it worked, saving a few lines of unnecessary code.

18/6/20

- Over the past couple weeks, I had worked on and finished my portfolio and tonight was the last night before this project was due. To conclude the project, I did one final test of everything and asked a couple others to test as well. Everything worked well but the only feedback I received was to add a message letting the user know that a file was sending as well as a loading bar for receiving the file. I quickly added these in as well as assigned the send data function to a separate thread:

```
def background_send(send_data):
    th3 = threading.Thread(target = send_data)
    th3.start()
```

```
global file_sending
file_sending = tkinter.Label(bottom_frame, text = "File sending...")

#create loading bar
global loading_bar
loading_bar = tkinter.ttk.Progressbar(bottom_frame, orient = "horizontal", length = 286, mode = "determinate")
```

# Testing and Evaluating

I tested this program throughout development but for the final test after I had finished the program to make sure everything worked, I used a form of black box testing as I knew the expected outputs. To test, I created a table of inputs and expected outputs for every screen and ticked the box when something worked. Here is the final test:

## Username screen

The username must be between 1 and 10 characters and it worked:

Inputs	Expected outputs		
	Too long	Too short	accepted
a			✓
Jordan			✓
123abc!!			✓
characters! !!!!!!!!!!!!	✓		

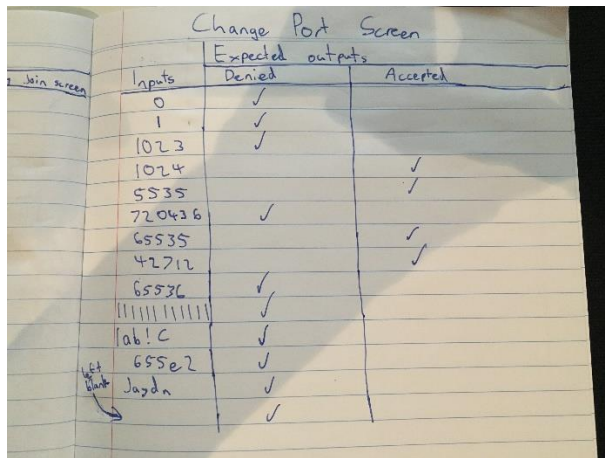
## Choose to host or join connection screen

There are 3 buttons on the page that when clicked, should move to their respected screens. It worked as well:

Inputs	Expected outputs		
	display port screen	display Host screen	display Join screen
Advanced...	✓		
Host		✓	
Join			✓

## Change port screen

The port must be a number between (and including) 1024 – 65535. When testing, it worked:



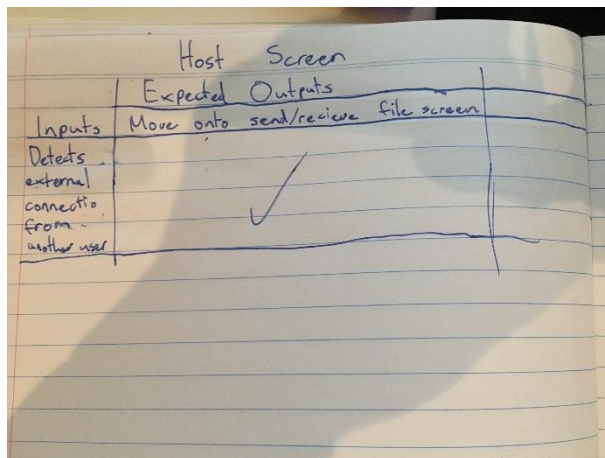
A handwritten table titled "Change Port Screen" with three columns: "Inputs", "Expected outputs", and "Accepted". The "Expected outputs" column is further divided into "Denied" and "Accepted" sub-columns. The table contains the following rows of data:

Inputs	Expected outputs	
	Denied	Accepted
0	✓	
1	✓	
1023	✓	
1024		✓
5535		✓
720436	✓	
65535		✓
42712		✓
65536	✓	
	✓	
ab!c	✓	
655e2	✓	
Jayden	✓	
	✓	

There is a handwritten note "Join screen" on the left margin and a small arrow pointing to the "Join screen" text.

## Host screen

The host screen requires another user to join, using their IP and then move onto the next screen. It worked:



A handwritten table titled "Host Screen" with two columns: "Inputs" and "Expected Outputs". The "Expected Outputs" column is further divided into "Move onto" and "send/receive file screen". The table contains the following rows of data:

Inputs	Expected Outputs	
	Move onto	send/receive file screen
Detects external connection from another user	✓	

## Join screen

A user will input a valid IP address, causing the send/receive screen to display. The IP I tested with was 127.0.0.1 which should be the only valid IP. It was successful:

For some IP's tested the response time took longer to load as it had to connect to an external computer through the network.

Handwritten table titled "Join Screen". The table has two main columns: "Inputs" and "Expected outputs". Under "Expected outputs", there are two sub-columns: "Accepted" and "Denied".

Inputs	Expected outputs	
	Accepted	Denied
127.0.0.1	✓	
127.0.0.2		✓
132.1.6.7		✓
192.169.0.1		✓
10.0.0.1		✓
202.142.76.1		✓
ab! 231		✓
Jaydn		✓
iiiiiiiiii laihia		✓

There is a handwritten note "left blank" with an arrow pointing to the "Inputs" column.

## Send or receive file screen

This screen had multiple processes that could be accomplished, I split it up into a send table and receive table.

The send function should only send existing files below 10gb and it worked. In the table below, I did not write in the file names, rather I described what times of files I sent as a person onlooking the data would not understand it without the descriptions:

Handwritten tables showing file sending and receiving results. The tables are organized into two main sections: "Send File (1st time)" and "Send File (After 1 has been sent)". Each section has a "Before File has been received" and an "After File has been received" sub-section.

**Send File (1st time)**

Inputs	Expected outputs		
	Too big	Cannot locate file	sent
(non-existent file)		✓	
(word document)		✓	✓
(27gb file)	✓		
(PDF)			✓
(.png)			✓
(.mp4)			✓

**After a file has been received**

Inputs	Expected outputs		
	Too big	Cannot locate file	sent
(non-existent file)		✓	
(word document)			✓
(27gb file)	✓		
(PDF)			✓
(.png)			✓
(.mp4)			✓

**Send File (After 1 has been sent)**

Inputs	Expected outputs		
	Too big	Cannot locate file	sent
" "		✓	
" "		✓	
" "	✓		✓
" "			✓
" "			✓
" "			✓

**After File has been received**

Inputs	Expected outputs		
	Too big	Cannot locate file	sent
" "		✓	
" "		✓	
" "	✓		✓
" "			✓
" "			✓
" "			✓

There are handwritten notes "left blank" and "same test as before" with arrows pointing to the "Inputs" column in the respective tables.



The receive function should detect when a file is ready to be received, activating the receive button. Any file should be able to be received and it worked. Again, I did not write the file names in the test data, rather the file types that had been received:

Receive File (1st time)		
Before sending a file		
Inputs	Expected Outputs	
	Activate button	Received File
(.docx)	✓	✓
(.pdf)	✓	✓
(.png)	✓	✓
(.mp4)	✓	✓

After sending a file		
Inputs	Expected Outputs	
	Activate button	Received File
(.docx)	✓	✓
(.pdf)	✓	✓
(.png)	✓	✓
(.mp4)	✓	✓

 | Receive File (After 1 has been received) |                  |               | |--|------------------|---------------| | Before sending a file                    |                  |               | | (same test data as before)               | Expected Outputs |               | | Inputs                                   | Activate button  | Received File | | " "                                      | ✓                | ✓             | | " "                                      | ✓                | ✓             | | " "                                      | ✓                | ✓             | | " "                                      | ✓                | ✓             |     | After sending a file       |                  |               | |----------------------------|------------------|---------------| | (same test data as before) | Expected Outputs |               | | Inputs                     | Activate button  | Received File | | " "                        | ✓                | ✓             | | " "                        | ✓                | ✓             | | " "                        | ✓                | ✓             | | " "                        | ✓                | ✓             | |

# Final Code

```
1 import tkinter
2 import tkinter.filedialog
3 import tkinter.ttk
4 import socket
5 import os
6 import threading
7
8
9 #Defines socket variables
10 hostname = socket.gethostname()
11 IP = socket.gethostbyname(hostname)
12 PORT = 32561
13 Headerlength = 10
14 server = False
15
16 #Defines username
17 username = "" #local username
18 user = "" #connected person's username
19
20 #Setup window
21 root = tkinter.Tk()
22 root.title("Blue File - Send files to any computer on the same network.")
23 root.geometry("600x150")
24 root.resizable(False, False)
25
26
27 def get_username(number):
28
29     global username
30     username = username_entry.get()
31
32     if len(username) > Headerlength:
33         too_short.pack_forget()
34         too_long.pack()
35         return False
36
37     if len(username) == 0:
38         too_long.pack_forget()
39         too_short.pack()
40         return False
41
42     switch(number)
43
```

```

44
45 def send_username():
46     #send username
47     client_socket.send(bytes(username, "utf-8"))
48
49     #recieve username
50     global user
51     user = client_socket.recv(10).decode().strip()
52
53
54 def switch(number):
55
56     #delete everything in the root window
57     for frames in root.winfo_children():
58         frames.destroy()
59
60     #call page number functions (switch page)
61     number()
62
63
64 def close_connection(mode):
65     try:
66         client_socket.shutdown(socket.SHUT_RDWR)
67     except:
68         pass
69     mode.close()
70
71
72 def cancel(mode):
73     close_connection(mode)
74     switch(page_1)
75
76
77 def exit(mode):
78     close_connection(mode)
79     root.destroy()
80
81
82 def remove_messages():
83     for messages in bottom_frame.grid_slaves(column = 1, row = 2):
84         messages.grid_forget()
85
86
87 def change_recieve():
88     global recieve_clicked

```

```

89     recieve_clicked = True
90
91
92 def finish_recieve():
93     global recieving
94     recieving = False
95
96     global recieve_clicked
97     recieve_clicked = False
98
99
100 def change_port():
101     global PORT
102
103     #check if port is an integer
104     try:
105         PORT = int(advanced_entry.get())
106
107         #check if port is a valid number
108         if 1024 <= PORT <= 65535:
109             advanced_window.destroy()
110         else:
111             port_error.grid(column = 0, row = 2, columnspan = 3)
112             return False
113
114     #if port is not an integer display error
115     except:
116         port_error.grid(column = 0, row = 2, columnspan = 3)
117         return False
118
119
120 def open_file():
121
122     #open file browser
123     file = tkinter.filedialog.askopenfilename(initialdir = "", title =
124     "Select a file")
125     #insert file directory into the file entry field
126     file_entry.delete(0, 'end')
127     file_entry.insert(0, file)
128
129 def background_server(server_setup):
130     th = threading.Thread(target = server_setup)
131     th.start()
132

```

```

133
134 def background_recieve(recieve_data):
135     th2 = threading.Thread(target = recieve_data)
136     th2.start()
137
138
139 def background_send(send_data):
140     th3 = threading.Thread(target = send_data)
141     th3.start()
142
143
144 def server_setup():
145
146     try:
147         #setup server
148         global server_socket
149         server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM) #IPV4, TCP
150         server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1) #allows socket to reuse address
151
152         server_socket.bind((IP, PORT)) #binds IP and PORT
153
154         #Establish connection to client
155         server_socket.listen() #listens for incoming connections
156
157         global client_socket
158         client_socket, client_address = server_socket.accept() #allow
for incoming connections
159
160         server = True
161     except:
162         return False
163
164     send_username()
165     switch(page_3)
166
167
168 def client_setup():
169
170     #get IP from enter_ip entry
171     IP = enter_ip.get()
172
173     try:
174         #setup client

```

```

175         global client_socket
176         client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM) #setup socket
177         client_socket.connect((IP, PORT)) #connect to server
178     except:
179         connection_error.grid(column = 1, row = 1)
180         return False
181
182     send_username()
183     switch(page_3)
184
185
186 def recieve_data():
187
188     global recieving
189     recieving = False
190
191     global recieve_clicked
192     recieve_clicked = False
193
194     file_name = ""
195
196     #recieves data in sets of 16bytes until all has been recieved.
197     data = bytes()
198
199     Header = client_socket.recv(4096) #recieve header
200
201     recieving = True
202
203     while recieving == True:
204         recieve_button.config(state = "normal")
205
206         while recieve_clicked == True:
207             recieve_button.config(state = "disabled")
208
209             file_split = str(Header).partition(":") #splits file
into 3 sections; name, size, file
210
211             file_name = str(file_split[0]) #gets file name from
split
212
213             #convert header to integer by taking the 2nd file split
up to the headerlength
214             file_size = int(file_split[2][:Headerlength])
215

```

```

216                                     #displays loading bar and sets the size to the
file_size-4096
217                                     remove_messages()
218                                     loading_bar.grid(column = 1, row = 2)
219                                     loading_bar["maximum"] = file_size-4096
220
221                                     #Data = Header - file name - headerlength - 1 (gets rid
of b' in file name)
222                                     data = Header[(len(file_name)+Headerlength-1):]
223
224                                     #removes b' from file name
225                                     file_name = file_name[2:]
226
227                                     while True:
228                                         if len(data) == file_size: #continue recieving
data until the length is the same length as file_size
229                                             finish_recieve()
230                                             break
231
232                                     else:
233                                         new_data = client_socket.recv(4096)
#recieve next set of data
234
235                                         #adds 4096 to the loading bars
236                                         loading_bar["value"] += 4096
237                                         loading_bar.update()
238
239                                         #add data onto the data variable
240                                         data += new_data
241
242                                     new_file = open(file_name, "wb") #create a new file with input name
243
244                                     new_file.write(data) #write to file
245                                     new_file.close() #close file
246
247                                     #ensure no other feedback messages are showing
248                                     remove_messages()
249
250                                     #show "file saved"
251                                     saved_file.grid(column = 1, row = 2)
252
253
254 def send_data():
255
256     path = file_entry.get()

```

```

257
258     try:
259         files = open(path, "rb") #opens the file as read and write
260
261     except FileNotFoundError:
262         #ensure no feedback messages show
263         for messages in bottom_frame.grid_slaves(column = 1, row = 2):
264             messages.grid_forget()
265
266         #show "cannot find file"
267         file_error.grid(column = 1, row = 2)
268         return False
269
270     file_size = os.path.getsize(path) #obtains file size
271
272     if file_size <= 9999999999:
273
274         #ensures no feedback messages show
275         remove_messages()
276         #prints "file sending..."
277         file_sending.grid(column = 1, row = 2)
278
279         file_name = os.path.basename(path) #gets file name with
extension
280
281         data = files.read(file_size) #reads file
282
283         full_send = bytes(f"{file_name}:{file_size:<{Headerlength}}",
"utf-8") + data #file size is sent as a string and left aligned as part of
the header
284         #sends file
285         client_socket.send(full_send)
286
287         files.close()
288
289         #ensure there are no feedback messages showing
290         remove_messages()
291         #show "file sent"
292         sent.grid(column = 1, row = 2)
293
294     else:
295         #ensure no feedback messages are showing
296         remove_messages()
297         #show "file too big"
298         file_too_big.grid(column = 1, row = 2)

```



```

299         return False
300
301
302 def page_user():
303
304     #create invisible frames
305     top_frame = tkinter.Frame(root)
306     top_frame.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
307
308     bottom_frame = tkinter.Frame(root)
309     bottom_frame.place(relx = 0, rely = 0.6, relwidth = 1, relheight = 1)
310
311     #create visible widgets
312     username_text = tkinter.Label(top_frame, text = "Enter Username:",
font = ("", 12))
313     username_text.grid(column = 0, row = 0, padx = 53, pady = 60)
314
315     global username_entry
316     username_entry = tkinter.Entry(top_frame, width = 15, font = ("", 12))
317     username_entry.grid(column = 1, row = 0)
318
319     confirm_button = tkinter.Button(top_frame, text = "Confirm", font =
("", 12), padx = 15, command = lambda: get_username(page_1))
320     confirm_button.grid(column = 2, row = 0, padx = 50)
321
322     #create feedback labels
323     global too_long
324     too_long = tkinter.Label(bottom_frame, text = "Error: Username is too
long.", fg = "red")
325
326     global too_short
327     too_short = tkinter.Label(bottom_frame, text = "Error: Username is too
short.", fg = "red")
328
329 def page_1():
330
331     #create invisible frames
332     top_frame = tkinter.Frame(root)
333     top_frame.place(relx = 0, rely = 0, relwidth = 1, relheight= 1)
334
335     left_frame = tkinter.Frame(root)
336     left_frame.place(relx = 0, rely = 0.3, relwidth = 0.5, relheight = 1)
337
338     right_frame = tkinter.Frame(root)

```

```

339     right_frame.place(relx = 0.5, rely = 0.3, relwidth = 0.5, relheight =
1)
340
341     #Create visible widgets
342     title = tkinter.Label(top_frame, text = "Start Connection", font =
("", 16))
343     title.grid(column = 1, row = 0, padx = 115)
344
345     advanced = tkinter.Button(top_frame, text = "Advanced...", font = ("",
8), command = page_changeport)
346     advanced.grid(column = 0, row = 0, padx = 20)
347
348     host_button = tkinter.Button(left_frame, text = "Host", font = ("",
12), padx = 40, command = lambda: switch(page_2a))
349     host_button.pack()
350
351     host_text = tkinter.Label(left_frame, text = "Allow other's to connect
using \nyour IP address.", font = ("", 10))
352     host_text.pack(pady = 15)
353
354     join_button = tkinter.Button(right_frame, text = "Join", font = ("",
12), padx = 40, command = lambda: switch(page_2b))
355     join_button.pack()
356
357     join_text = tkinter.Label(right_frame, text = "Connect to someone else
on your \nnetwork using their IP address.", font = ("", 10))
358     join_text.pack(pady = 15)
359
360
361 def page_2a():
362
363     #create invisible frames
364     top_frame = tkinter.Frame(root)
365     top_frame.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
366
367     bottom_frame = tkinter.Frame(root)
368     bottom_frame.place(relx = 0, rely= 0.3, relwidth = 1, relheight = 1)
369
370     #create visible widgets
371     title = tkinter.Label(top_frame, text = "Host Connection", font = ("",
16))
372     title.pack()
373
374     waiting = tkinter.Label(bottom_frame, text = "Waiting for incoming
connection...", font = ("", 12, "italic"))

```

```

375     waiting.grid(column = 1, row = 0)
376
377     cancel_button = tkinter.Button(bottom_frame, text = "Cancel", font =
378     ("", 12), padx = 20, command = lambda: cancel(server_socket))
379
380     ip_text = tkinter.Label(bottom_frame, text = f"Your IP address is:
381     {IP}", font = ("", 10))
382
383     background_server(server_setup)
384     root.protocol("WM_DELETE_WINDOW", lambda: exit(server_socket))
385
386
387 def page_2b():
388
389     #create invisible frames
390     top_frame = tkinter.Frame(root)
391     top_frame.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
392
393     middle_frame = tkinter.Frame(root)
394     middle_frame.place(relx = 0, rely= 0.3, relwidth = 1, relheight = 1)
395
396     bottom_frame = tkinter.Frame(root)
397     bottom_frame.place(relx = 0, rely = 0.7, relwidth = 1, relheight = 1)
398
399     #Create visible widgets
400     title = tkinter.Label(top_frame, text = "Join Connection", font = ("",
401     16))
402
403     title.pack()
404
405     enter_text = tkinter.Label(middle_frame, text = "Enter IP address
406     here:", font = ("", 12))
407     enter_text.grid(column = 0, row = 0, padx = 25)
408
409     global enter_ip
410     enter_ip = tkinter.Entry(middle_frame, width = 20, font = ("", 12))
411     enter_ip.grid(column = 1, row = 0)
412
413     join_button = tkinter.Button(middle_frame, text = "Join", font = ("",
414     12), padx = 40, command = client_setup)
415     join_button.grid(column = 3, row = 0, padx = 25)
416
417     cancel_button = tkinter.Button(bottom_frame, text = "Cancel", font =
418     ("", 12), padx = 20, command = lambda: switch(page_1))

```

```

414     cancel_button.grid(column = 0, row = 0, padx = 20, sticky = "nw")
415
416     help_text = tkinter.Label(bottom_frame, text = "The host's IP address
will be displayed on their screen.")
417     help_text.grid(column = 1, row = 0, columnspan = 3, pady = 15)
418
419     #create feedback Labels
420     global connection_error
421     connection_error = tkinter.Label(middle_frame, text = "Error: Cannot
connect to given IP.", fg = "red")
422
423 def page_changeport():
424
425     #create window
426     global advanced_window
427     advanced_window = tkinter.Toplevel(root)
428     advanced_window.geometry("260x80")
429     advanced_window.resizable(False, False)
430
431     #add visible widgets
432     advanced_text = tkinter.Label(advanced_window, text = "Port:")
433     advanced_text.grid(column = 0, row = 0, padx = 15, pady = 10)
434
435     global advanced_entry
436     advanced_entry = tkinter.Entry(advanced_window, width = 10)
437     advanced_entry.grid(column = 1, row = 0)
438     advanced_entry.insert(0, PORT)
439
440     advanced_button = tkinter.Button(advanced_window, text = "confirm",
padx = 10, command = change_port)
441     advanced_button.grid(column = 2, row = 0, padx = 15)
442
443     default_text = tkinter.Label(advanced_window, text = "Default is port
is 32561.", fg = "grey")
444     default_text.grid(column = 0, row = 1, columnspan = 3)
445
446     #create feedback labels
447     global port_error
448     port_error = tkinter.Label(advanced_window, text = "Error: Port must
be between 1024 - 65535.", fg = "red")
449
450
451 def page_3():
452
453     #create invisible frames

```

```

454     top_frame = tkinter.Frame(root)
455     top_frame.place(relx = 0, rely = 0, relwidth = 1, relheight = 1)
456
457     global bottom_frame
458     bottom_frame = tkinter.Frame(root)
459     bottom_frame.place(relx = 0, rely= 0.3, relwidth = 1, relheight = 1)
460
461     title = tkinter.Label(top_frame, text = f"Connected to {user}", font =
462     ("", 16))
463     title.pack()
464
465     #create visible widgets
466     file_text = tkinter.Label(bottom_frame, text = "Choose file:", font =
467     ("", 12))
468     file_text.grid(column = 0, row = 0, padx = 25)
469
470     global file_entry
471     file_entry = tkinter.Entry(bottom_frame, width = 45, font = ("", 10))
472     file_entry.grid(column = 1, row = 0)
473
474     file_browse = tkinter.Button(bottom_frame, text = "Browse...", font =
475     ("", 12), command = open_file)
476     file_browse.grid(column = 2, row = 0, padx = 25)
477
478     send_button = tkinter.Button(bottom_frame, text = "Send", font = ("",
479     12), padx = 20, command = lambda: background_send(send_data))
480     send_button.grid(column = 2, row = 3, pady = 6)
481
482     global recieve_button
483     recieve_button = tkinter.Button(bottom_frame, text = "Recieve", font =
484     ("", 12), padx = 20, state = "disabled", command = change_recieve)
485     recieve_button.grid(column = 1, row = 3)
486
487     cancel_button = tkinter.Button(bottom_frame, text = "Cancel", font =
488     ("", 12), padx = 20)
489     cancel_button.grid(column = 0, row = 3, pady = 8)
490
491     #default message
492     global send_recieve
493     send_recieve = tkinter.Label(bottom_frame, text = "Send or recieve a
494     file.")
495     send_recieve.grid(column = 1, row = 2)
496
497     #create feedback messages
498     global file_too_big

```

```

492     file_too_big = tkinter.Label(bottom_frame, text = "Error: File must be
under 10gb.", fg = "red")
493
494     global file_error
495     file_error = tkinter.Label(bottom_frame, text = "Error: Cannot locate
file.", fg = "red")
496
497     global sent
498     sent = tkinter.Label(bottom_frame, text = "File sent!", fg = "green")
499
500     global saved_file
501     saved_file = tkinter.Label(bottom_frame, text = "File saved!", fg =
"green")
502
503     global file_sending
504     file_sending = tkinter.Label(bottom_frame, text = "File sending...")
505
506     #create loading bar
507     global loading_bar
508     loading_bar = tkinter.ttk.Progressbar(bottom_frame, orient =
"horizontal", length = 286, mode = "determinate")
509
510     #tests if the server is running in this instance
511     if server == True:
512         #if server is running, cancel and exit buttons will work in
server mode
513         root.protocol("WM_DELETE_WINDOW", lambda: exit(server_socket))
514         cancel_button.config(command = lambda: cancel(server_socket))
515
516     else:
517         #if server is NOT running, cancel and exit buttons will work
in client mode
518         root.protocol("WM_DELETE_WINDOW", lambda: exit(client_socket))
519         cancel_button.config(command = lambda: cancel(client_socket))
520
521     background_recieve(recieve_data)
522
523
524 page_user()
525 root.mainloop()

```

# Reflection

Throughout development of the project, I have learnt a great deal not only to help with my HSC but software design in general. Developing this project as reinforced my understanding in basic software principles for example; the creation of easy to call functions which allow me to simplify and better understand my code, the use of comments has allowed me to take small one or two week breaks and continue working on code as if it were the day I wrote it. The development of the project has also allowed me to better understand FOR, WHILE and IF loops; encouraging me to delve deeper and make better use of them.

The creation of the portfolio has also assisted in the creation of my project encouraging me to drawn diagrams before starting coding which helped me understand what I wanted to do and achieve better. For example, I would not have drawn the GUI diagrams before the creation of the software if I was not required, but it surprisingly helped in my design, allowing me to come up with a range of drafts and then get feedback on them, only to improve them in the final project.

The prototyping-like approach I have taken has already greatly assisted in the way I like to code. This approach inspired the formation of my ideas while learning how to use python and its included modules, thereby expanding my creative freedom. If I had rather used a structured approach the constant formation of ideas would be a lot more limited. If I changed my idea in even the slightest part way into the project, it would have been a lot harder to implement than the evolving nature of the prototyping approach.

My time management plan was also greatly successful, to the point where I finished the task early, while still fitting time in for other schoolwork. Throughout most of the year, I limited myself to about an hour a week but sometimes I could spend multiple. The quarantine situation gave me the opportunity to spend time nearly every day working on my project, allowing me to finish in advance.

If I was to redo the project differently, I would have spent more time using Python's manuals, understanding how things work myself rather than using YouTube videos and forums to teach me. For example, when learning how to use networking, I spent a large amount of time watching many YouTube videos and browsed many websites explaining how to do it in Python (all were quite similar). I think it would have been more beneficial learning how networking works in general through YouTube and the websites and then applying that knowledge in Python using their manuals. This would have allowed a deeper understanding for me as I would have tested many different approaches, seeing which works best rather than relying on tutorials to teach me.

Ultimately, the method I have taken has allowed my project to evolve and develop over time, with the help of peer and teacher feedback, my idea evolved from a simple program to simplify the process in helping understand the bushfire situation of the time to a larger, more complex

file transfer system. As I refined my file transfer program over time, I have achieved a successful program using all the skills I have learnt in the Software Design and Development course.



# References

agomcas, 2018. *How to run process in the background without stopping the Gui in Tkinter*. [Online]

Available at: <https://stackoverflow.com/questions/48180989/how-to-run-process-in-the-background-without-stopping-the-gui-in-tkinter>

[Accessed 6 May 2020].

Anon., 2020. *socket — Low-level networking interface*. [Online]

Available at: <https://docs.python.org/3/library/socket.html>

[Accessed 2 March 2020].

Anon., n.d. *XML Parsing*. [Online]

Available at: [http://www2.hawaii.edu/~takebaya/cent110/xml\\_parse/xml\\_parse.html](http://www2.hawaii.edu/~takebaya/cent110/xml_parse/xml_parse.html)

[Accessed 28 December 2019].

Codemy.com, 2019. *Create Graphical User Interfaces With Python And Tkinter*. [Online]

Available at:

<https://www.youtube.com/watch?v=yQSEXcf6s2I&list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV&index=1>

[Accessed 24 April 2020].

Dojo, D. S., 2017. *Intro to Web Scraping with Python and BeautifulSoup*. [Online]

Available at: <https://www.youtube.com/watch?v=XQgXKtPSzUI>

[Accessed 26 December 2019].

Engineer Man, 2018. *Threading vs Multiprocessing in Python*. [Online]

Available at: <https://www.youtube.com/watch?v=ecKWiaHCEKs>

[Accessed 6 May 2020].

Gabriel, M., 2013. *Python Socket Receive Large Amount of Data*. [Online]

Available at: <https://stackoverflow.com/questions/17667903/python-socket-receive-large-amount-of-data>

[Accessed 11 April 2020].

Ghosh, B., 2019. *Python Socket File Transfer Send*. [Online]

Available at: <https://linuxhint.com/python-socket-file-transfer-send/>

[Accessed 3 March 2020].

Kite, n.d. *setblocking*. [Online]

Available at: <https://kite.com/python/docs/socket.socket.setblocking>

[Accessed 20 April 2020].

Manoharan, V., 2013. *Send/receive infinite loop with SocketServer - Python - ConnectionAbortedError*. [Online]

Available at: <https://stackoverflow.com/questions/14201147/send-receive-infinite-loop-with-socketserver-python-connectionabortederror>  
[Accessed 11 April 2020].

Rosenfield, A. et al., 2013. *Python Socket Receive Large Amount of Data*. [Online]  
Available at: <https://stackoverflow.com/questions/17667903/python-socket-receive-large-amount-of-data>  
[Accessed 11 April 2020].

sentdex, 2017. *Sockets Tutorial with Python 3 part 3 - sending and receiving Python Objects with sockets*. [Online]  
Available at: <https://pythonprogramming.net/pickle-objects-sockets-tutorial-python-3/?completed=/buffering-streaming-data-sockets-tutorial-python-3/>  
[Accessed 13 April 2020].

sentdex, 2019. *Socket Chatroom server - Creating chat application with sockets in Python*. [Online]  
Available at: [https://www.youtube.com/watch?v=CV7\\_stUWvBQ](https://www.youtube.com/watch?v=CV7_stUWvBQ)  
[Accessed 19 February 2020].

sentdex, 2019. *Sockets Tutorial with Python 3 part 1 - sending and receiving data*. [Online]  
Available at: [https://www.youtube.com/watch?v=Lbfe3-v7yE0&list=PLQVvva0QuDdzLB\\_0JSTTcl8E8isJLhR5](https://www.youtube.com/watch?v=Lbfe3-v7yE0&list=PLQVvva0QuDdzLB_0JSTTcl8E8isJLhR5)  
[Accessed 11 April 2020].

sentdex, 2019. *Sockets Tutorial with Python 3 part 3 - sending and receiving Python Objects w/ Pickle*. [Online]  
Available at: <https://www.youtube.com/watch?v=WM1z8soch0Q&t>  
[Accessed 14 February 2020].

SP, S., 2017. *Python/socket: How to send a file to another computer which is on a different network?*. [Online]  
Available at: <https://stackoverflow.com/questions/46979262/python-socket-how-to-send-a-file-to-another-computer-which-is-on-a-different-ne>  
[Accessed 16 March 2020].

Stepanov, A., n.d. *How to Work with TCP Sockets in Python (with Select Example)*. [Online]  
Available at: <https://steelkiwi.com/blog/working-tcp-sockets/>  
[Accessed 4 March 2020].

Tech With Tim, 2020. *Threading Tutorial #1 - Concurrency, Threading and Parallelism Explained*. [Online]  
Available at: <https://www.youtube.com/watch?v=olYdb0DdGtM>  
[Accessed 6 May 2020].

Tech With Tim, 2020. *Threading Tutorial #2 - Implementing Threading in Python 3 (Examples)*. [Online]

Available at: <https://www.youtube.com/watch?v=cdPZ1pJACMI>

[Accessed 6 May 2020].

Tutorialspoint, n.d. *Python - GUI Programming (Tkinter)*. [Online]

Available at: [https://www.tutorialspoint.com/python/python\\_gui\\_programming.htm](https://www.tutorialspoint.com/python/python_gui_programming.htm)

[Accessed 28 April 2020].