Client

The Client file utilizes socket programming to send files over the existing network.

First we call the needed libraries `os` and `socket`

```
import os
import socket
```

```
HEADER = 64
PORT = 5010
BUFFER_SIZE = 4096
FORMAT = 'utf-8'
SERVER = "127.0.1.1"
ADDR = (SERVER, PORT)
```

Going over the defined constants here:

Constant	Usage
PORT	The port that will be used to communicate with the server, so any time one of the clients want to send data to the server, it's supposed to send this data on that port.
SERVER	The IP that the server will be hosted on.
BUFFER_SIZE	The number of bytes that will written to received file.
ADDR	A tuple that holds both `SERVER` and `PORT` values to be used when creating the socket.
FORMAT	The format that will be used when converting the bytes received from the client to string.

Next, we create the a socket **object** `client` to be able to connect to the server with the **method** `.connect()`

```
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDR)
```

We define a function `send_file` to send the intended files.

```
def send_file(filename):
    # the name of file we want to send, make sure it exists
    filename = str(filename)

    # get the file size
    filesize = os.path.getsize(filename)
```

Using the methods provided by the `os` library such as `os.path.getsize()` we give an argument of the intended file name and it returns the file size.

```
    # send the filename and filesize
    client.send(f"{filename} {filesize}".encode())
    print(filename)
```

Starting by sending the filename and the file size for the server to have the needed info to act on before receiving the actual data using the method `.send()` that takes an argument of the string to be sent and the method `.encode` to encode the string to be sent.

```
# start sending the file
    with open(filename, "rb") as f:
        while True:
            # read the bytes from the file
            bytes_read = f.read(BUFFER_SIZE)
            if not bytes_read:
                # file transmitting is done
                break
            # we use sendall to assure transimission in busy networks
            client.sendall(bytes_read)
        f.close()
    # close the socket
    client.close()
```

Using the method `.read()` and giving it an argument of the buffer size to make sure the sent data does not exceed the buffer size, then sending it using the `.sendall()` method taking the argument as the bytes that were read and finally closing the connection with `.close()` Since the sending operation is done with an continuous loop we set a condition where if that are no bytes read we break the loop.

```
            if not bytes_read:
                # file transmitting is done
                break
```

Finally, we close the socket since it will no longer be of further use.

```
    # close the socket
    client.close()
```

Utilities

first, we did a files locator roughly based on the text locator we did before in the transmitter file in a prior attempt which exists in the previous report. The function is now called files_locator which takes two arguments the first being the directory where we try to locate the files, the second being the file extension or simply put the file type.

As usual we need the `os` library

```
import os
```

Using the os library we get all the files in the directory of choice and store it into a list called directory_content with the help of ``os.listdir()` which takes an argument of the intended directory.

```
def files_locator(dir, extension):
    # list of strings containing files names in the specified directory
    directory_content = os.listdir(dir)
```

we initialize an empty list to put in the files of the desired type only which are located with the help of the method `.endswith()`

```
    files = []
    for file in directory_content:
        if file.endswith(extension):
            files.append(file)
```

In order to sort the files we need to get their modification times which we can get using the `os.path.getmtime()` method which gives the time from a certain specific reference called epoch time.

```
    sec = []
    # for loop to get each file creation time
    for csv_items in files:
        modTimesinceEpoc = os.path.getmtime(csv_items)
        sec.append(modTimesinceEpoc)
```

The rest is a matter of bubble sorting algorithm and to finally return the names of the files in order.

```
 # initialize temporary variables int and str respectively to use in sorting
tempInt = 0
tempStr = ''
# int to store the number of text files
number_of_csv = len(files)
# bubble sorting for loop to sort the file name according to their creation time
for index in range(number_of_csv-1):
    if sec[index] > sec[index+1]:
        tempInt = sec[index+1]
        tempStr = files[index+1]
        sec[index+1] = sec[index]
        files[index+1] = files[index]
        sec[index] = tempInt
        files[index] = tempStr
```

And finally we return the order list of the files names.

```
 return files
```