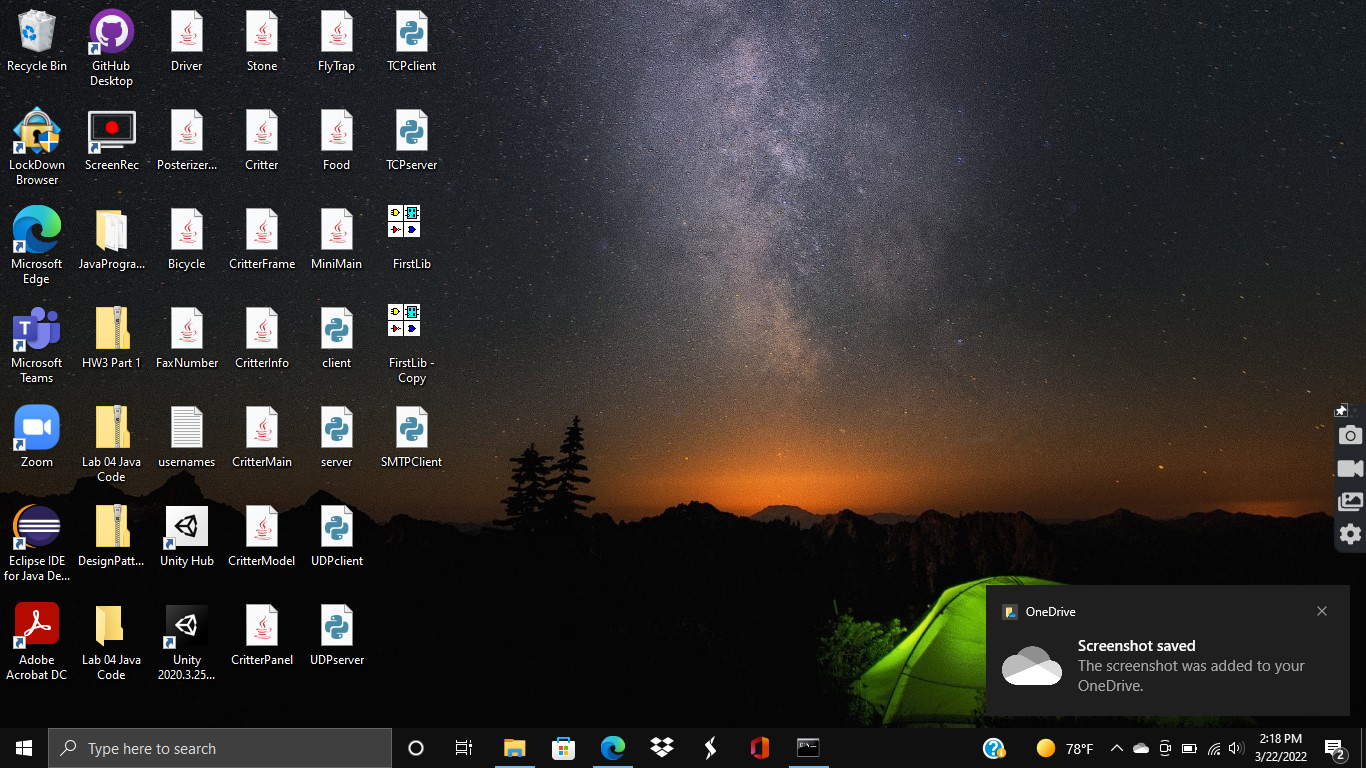
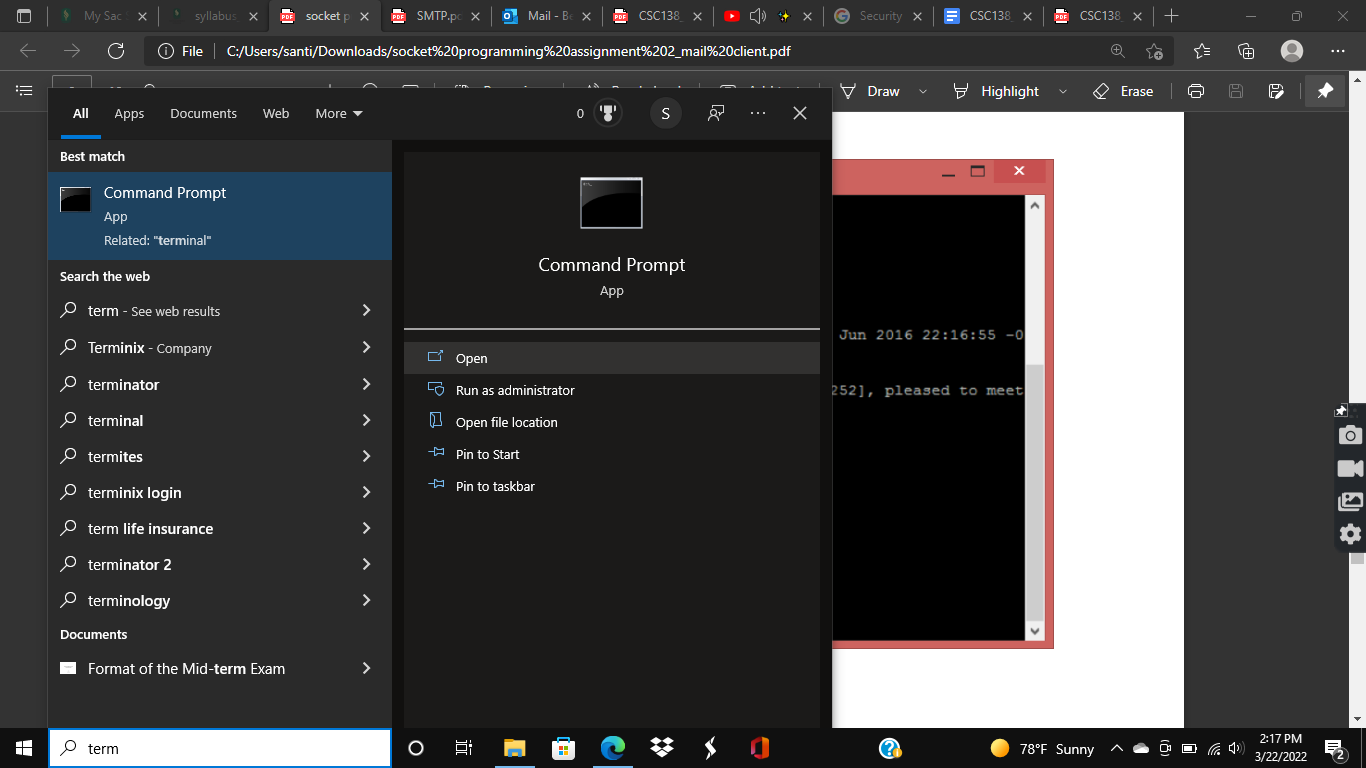
Author: Santiago Bermudez

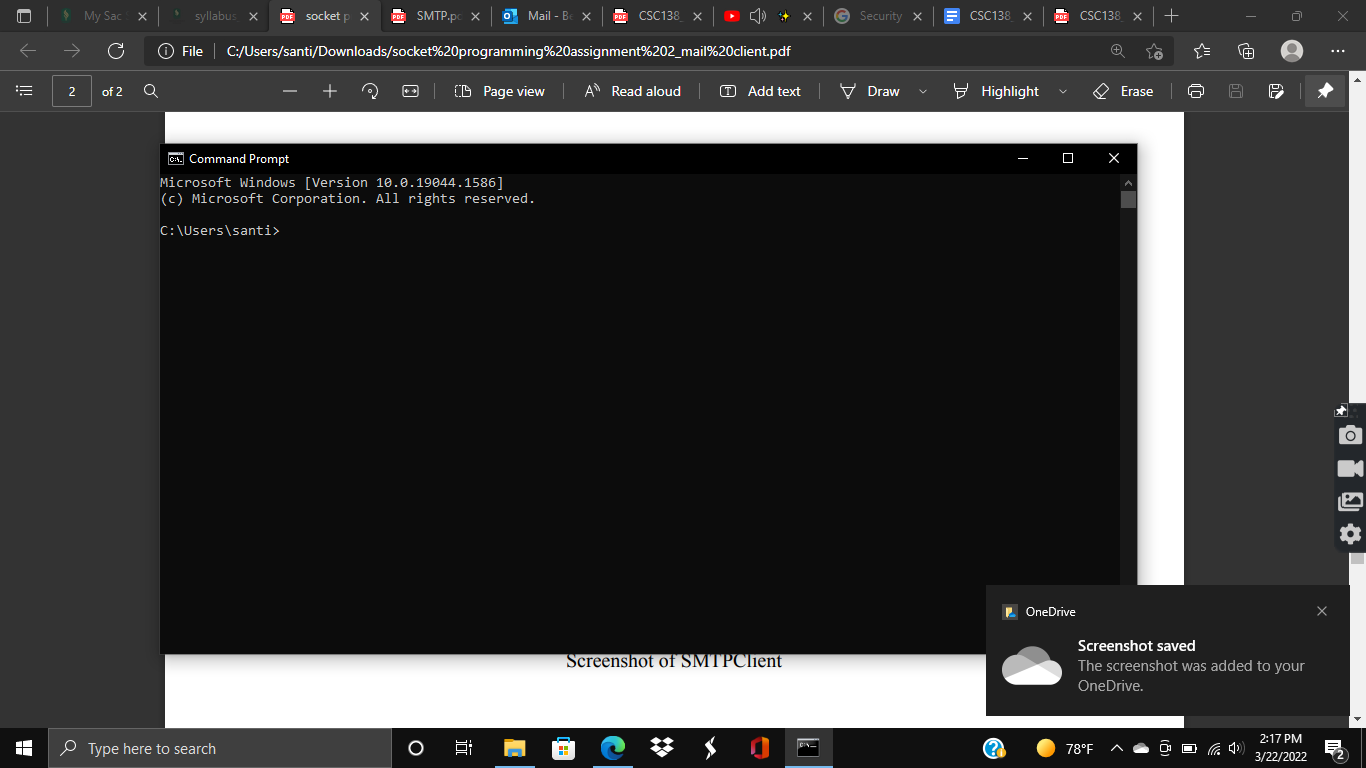
CSC 138

Section 4

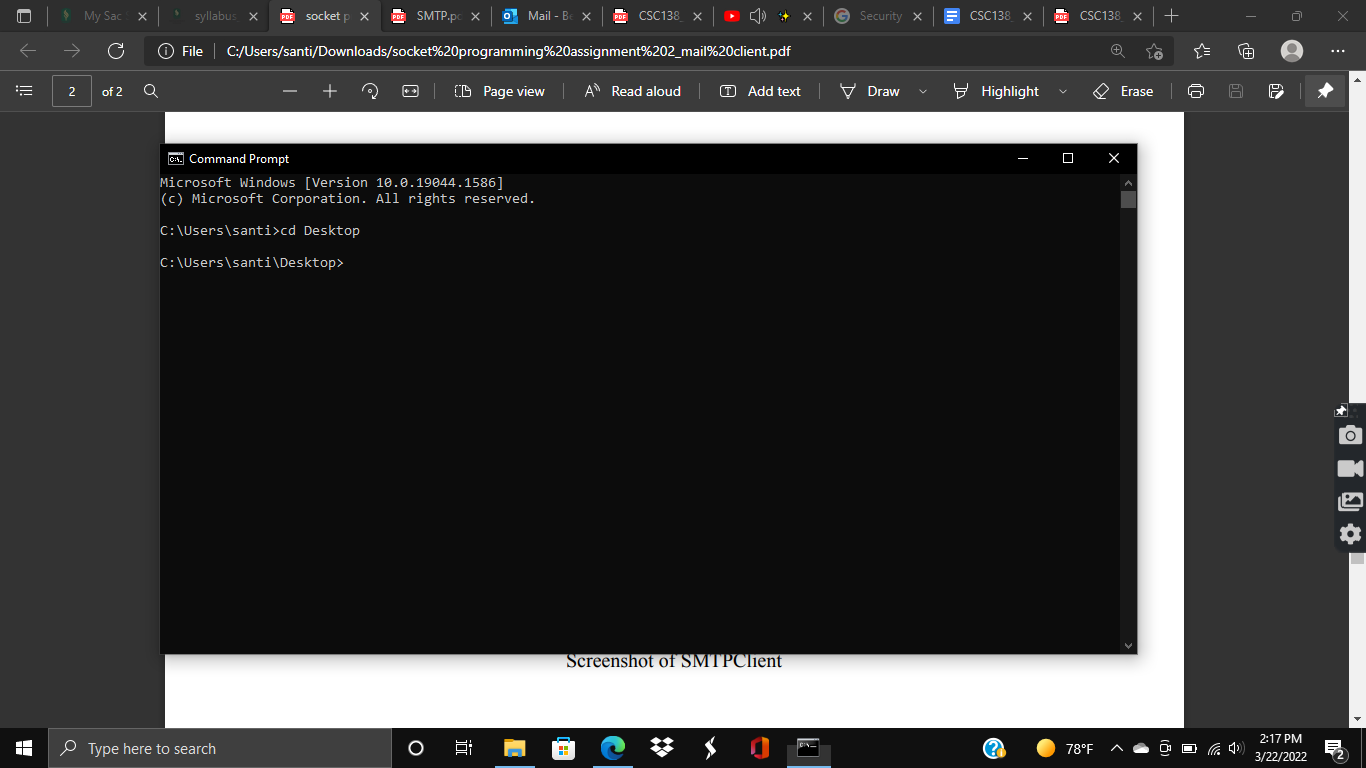
For starters and for simplicity, you will want to make sure that the code is on your desktop as shown in the upper right below.



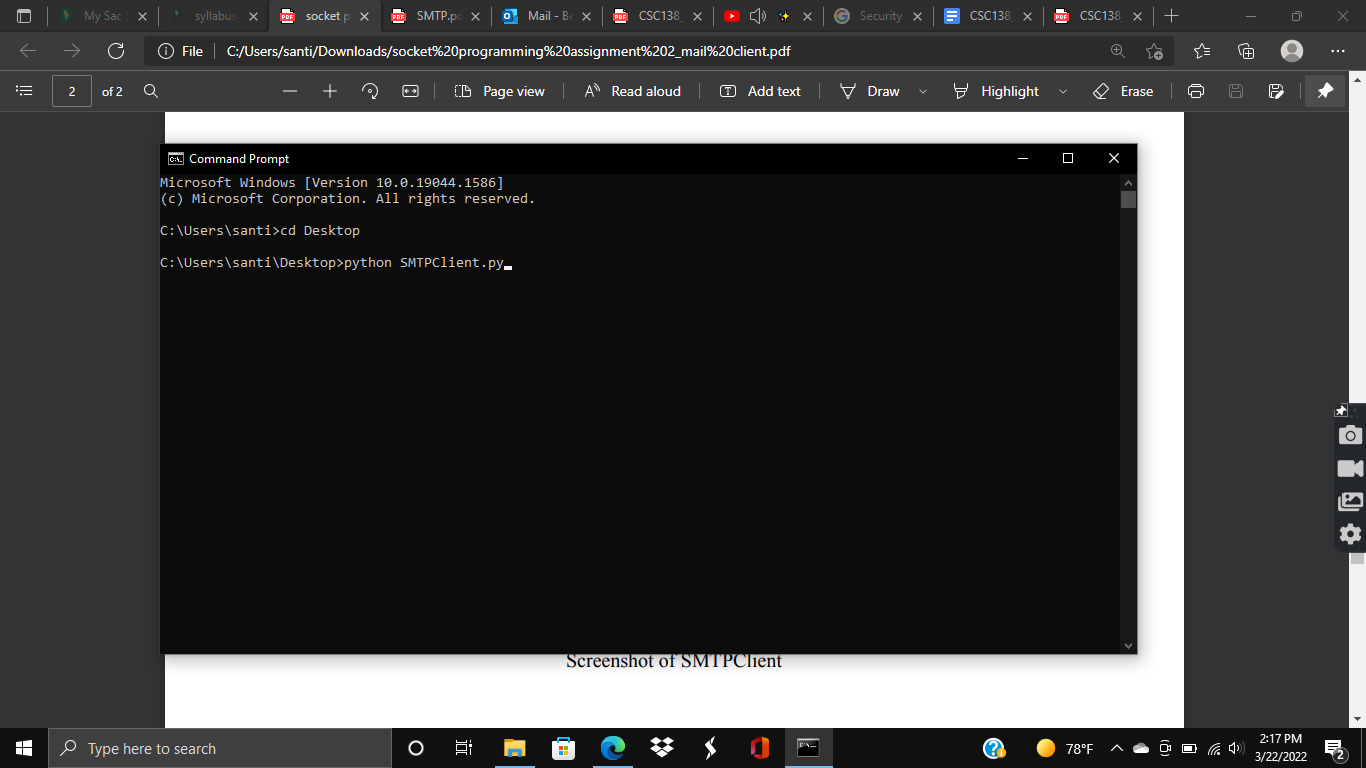
Then, you will want to open up the terminal. There is no need to sign into the ECS server this time around, you just need the terminal open.



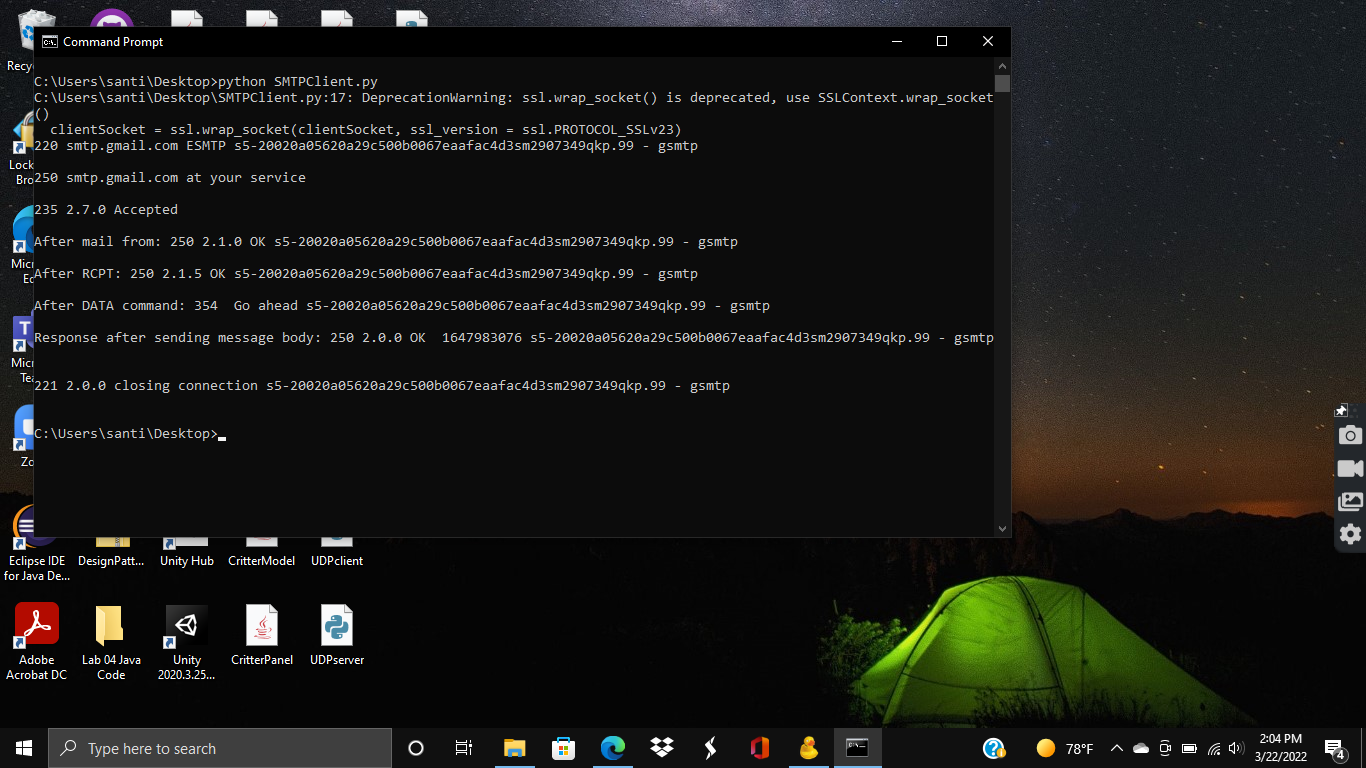
From there, you will want to change the directory to your desktop using the “cd Desktop” command.



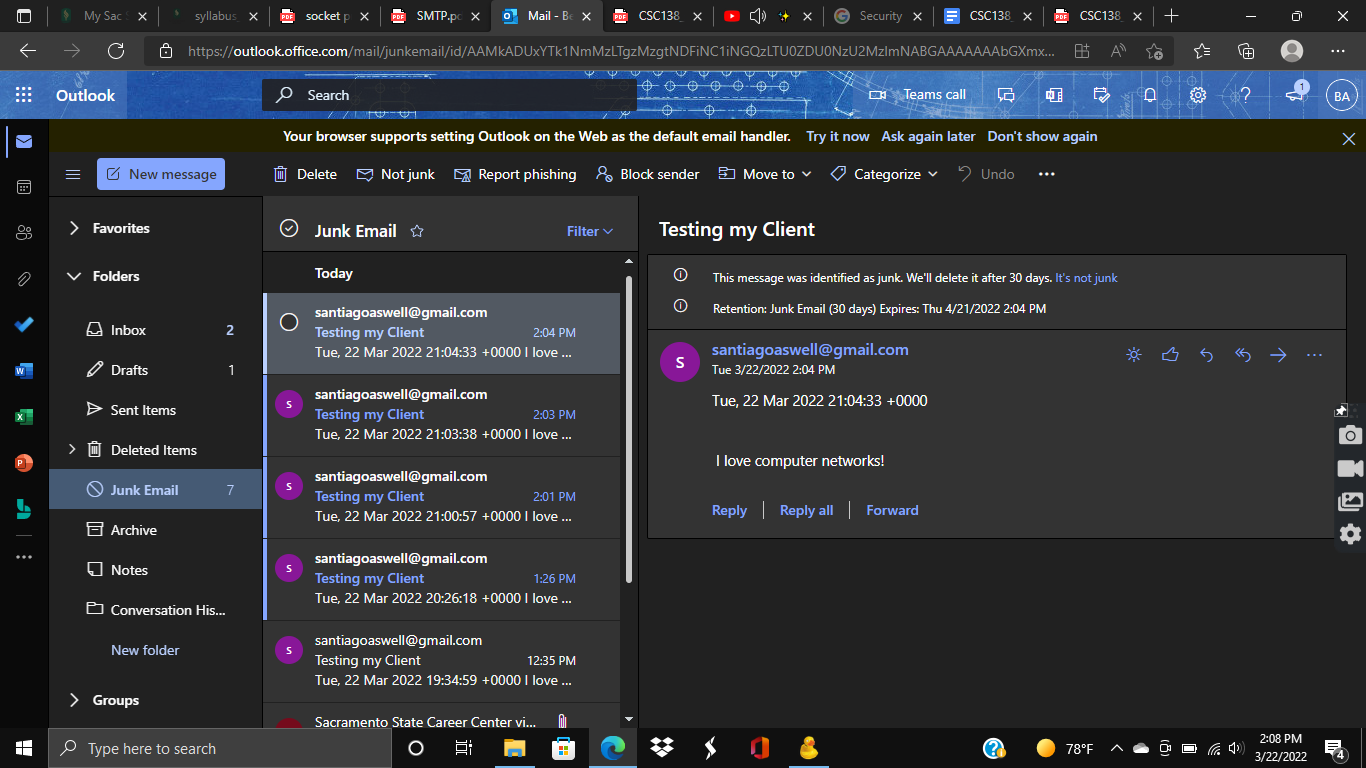
Once you have done that, you can simply execute the code.



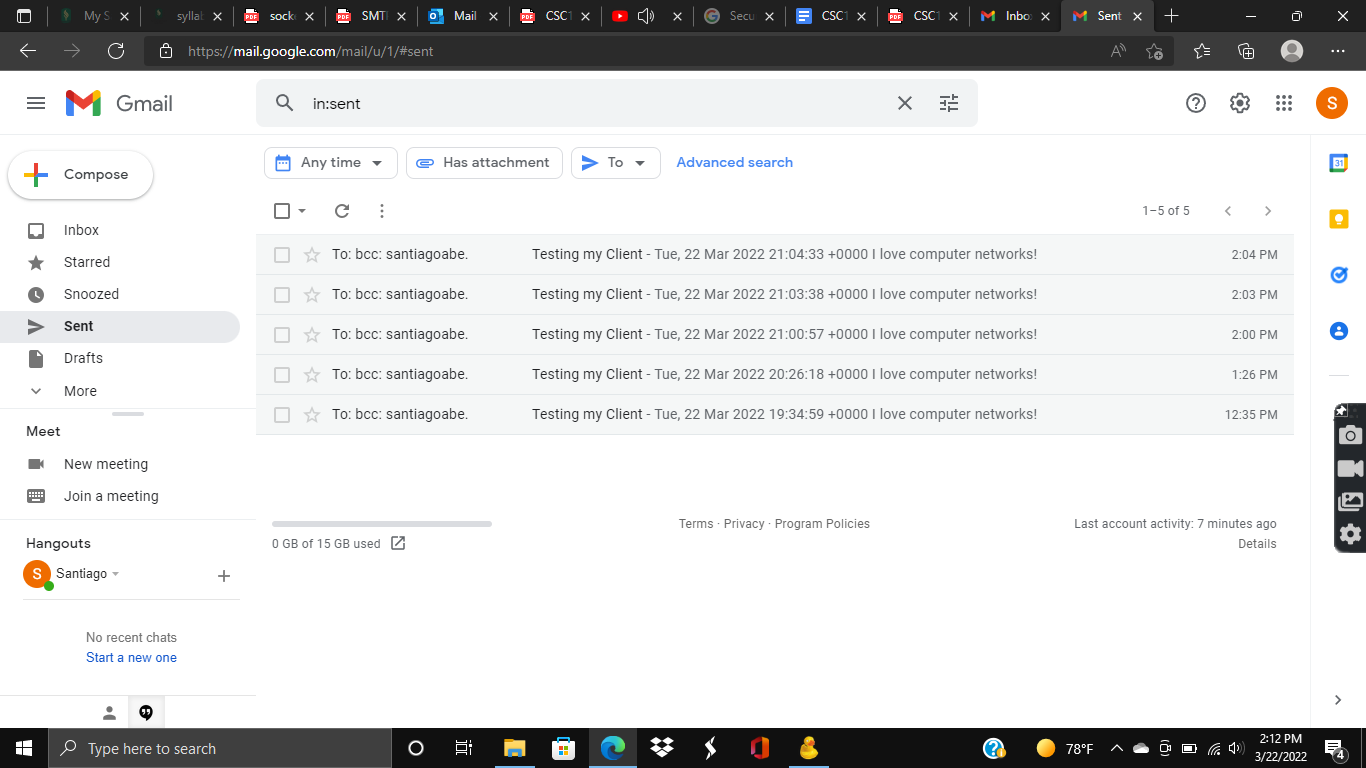
You may expect to see output like the one shown below.

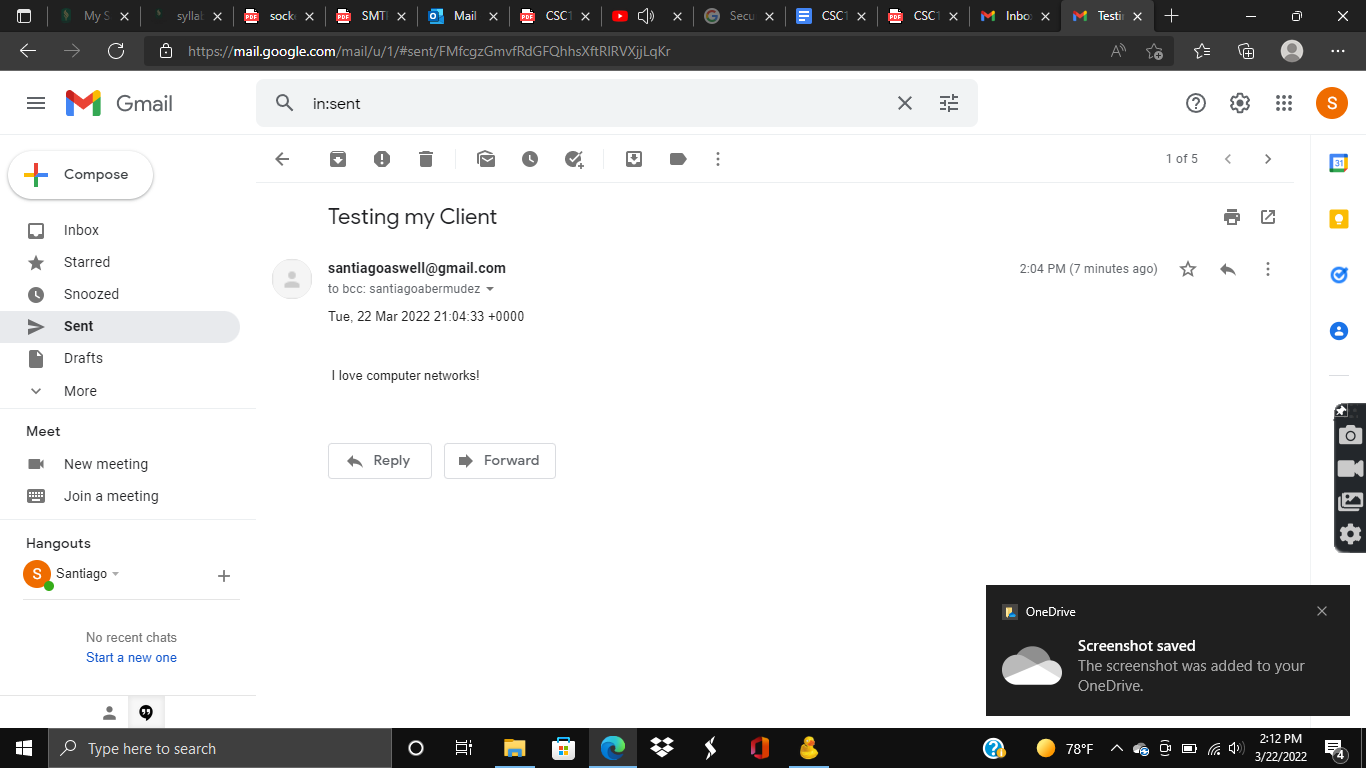


Upon running the code, you should see an email to the recipient that you have selected from the sender you chose (\*note that for security reasons I made another account for sending). I used my school email as the recipient in this case.



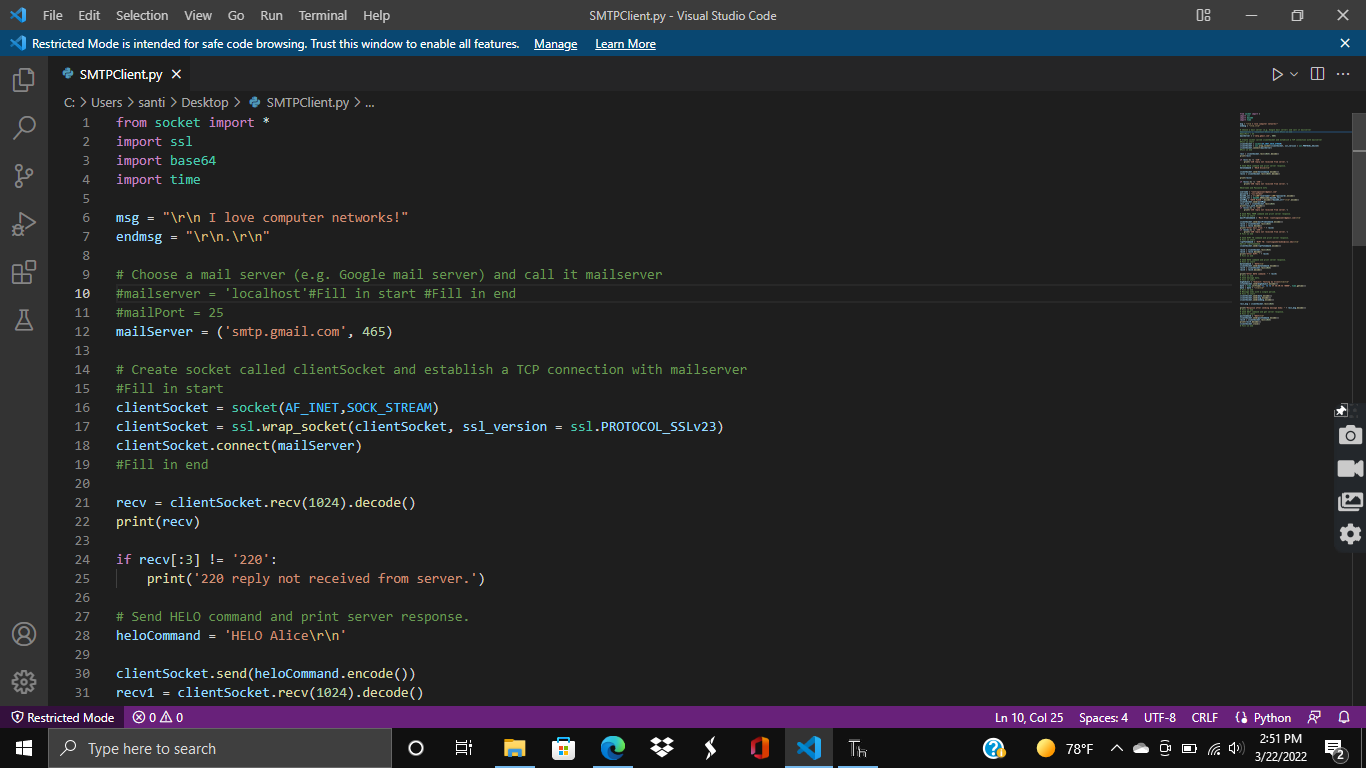
You can also see the email being sent from the made up account I used below.



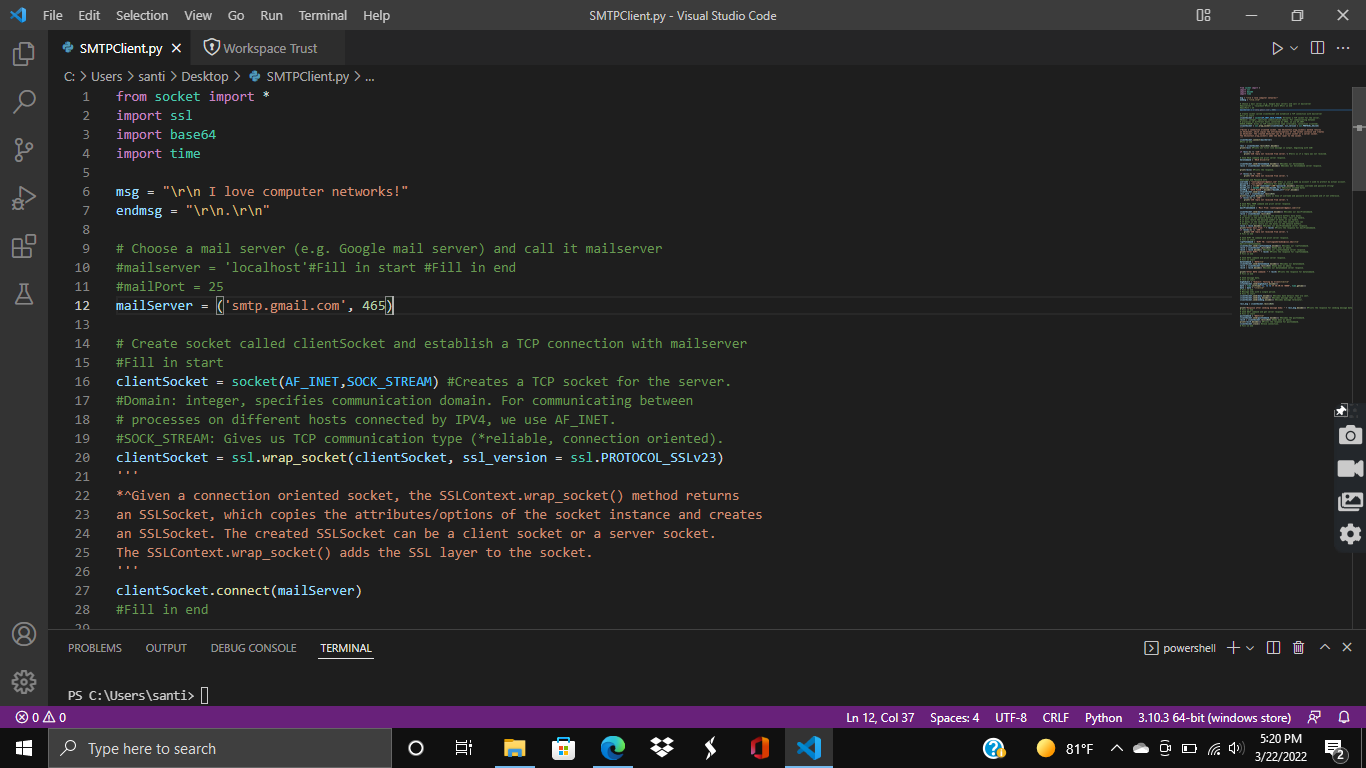


Things I learned/Mistakes I made/Choices I made and so on:

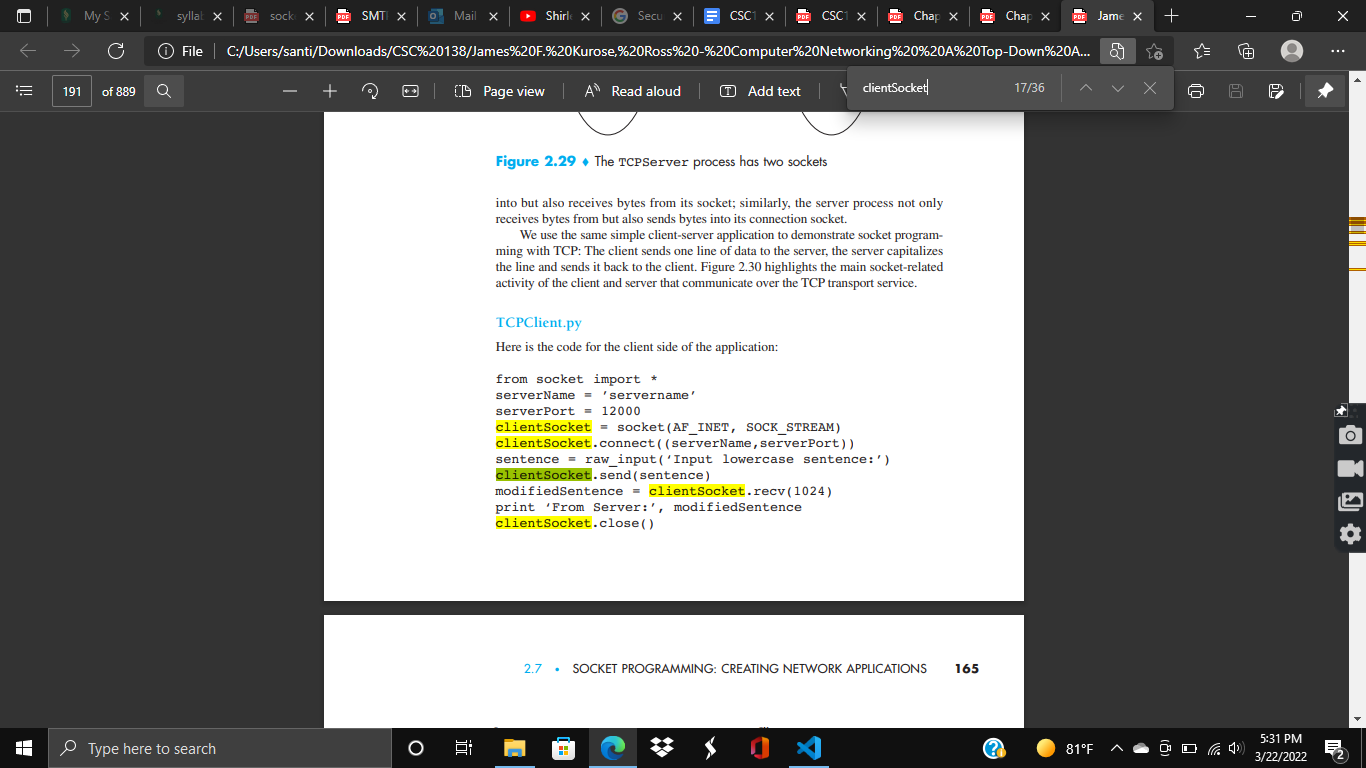
This assignment was filled with many firsts for me. The first issue I had was choosing a mail server. I decided to go along with the Google mail server to start with. Because we are to establish dialogue with the mail server using the SMTP protocol, I decided to use the following SMTP Server Address: smtp.gmail.com. The port used is 465, because that is the SSL port, which provides some encryption for security and integrity.

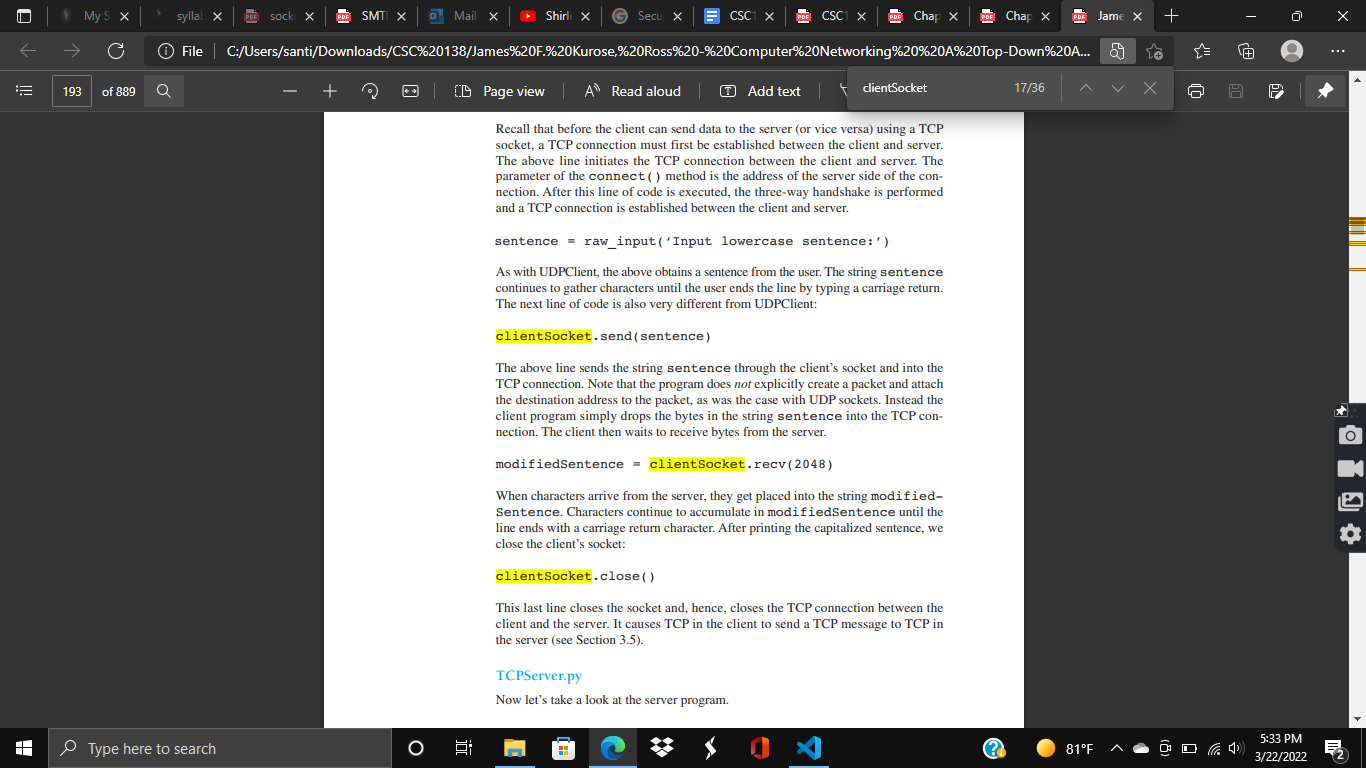


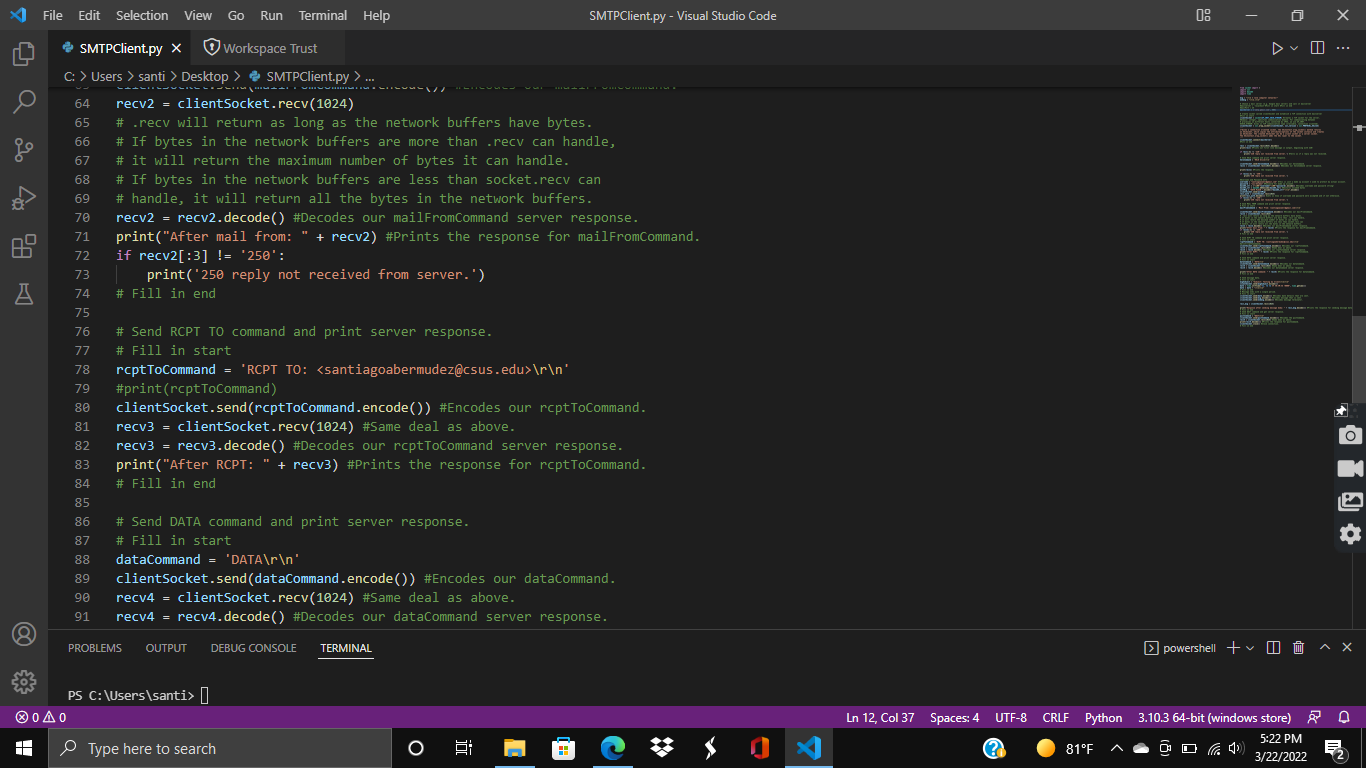
The second task I had struggled with was creating a socket called clientSocket and establishing a TCP connection with the mailserver. For this, I looked at my TCPclient code from the previous programming assignment as reference.



The last major issue I had struggled with was filling in the code for the various commands that were required in this assignment. For this, I had to look at the book in depth to know what was going on and understand more about the clientSocket code. I posted a few excerpts below!







\*Source code below:

from socket import \*

import ssl

import base64

import time

msg = "\r\n I love computer networks!"

endmsg = "\r\n.\r\n"

# Choose a mail server (e.g. Google mail server) and call it mailserver

#mailserver = 'localhost'#Fill in start #Fill in end

#mailPort = 25

mailServer = ('smtp.gmail.com', 465)

# Create socket called clientSocket and establish a TCP connection with mailserver

#Fill in start

clientSocket = socket(AF\_INET,SOCK\_STREAM) #Creates a TCP socket for the server.

#Domain: integer, specifies communication domain. For communicating between

# processes on different hosts connected by IPV4, we use AF\_INET.

#SOCK\_STREAM: Gives us TCP communication type (\*reliable, connection oriented).

clientSocket = ssl.wrap\_socket(clientSocket, ssl\_version = ssl.PROTOCOL\_SSLv23)

'''

\*^Given a connection oriented socket, the SSLContext.wrap\_socket() method returns

an SSLSocket, which copies the attributes/options of the socket instance and creates

an SSLSocket. The created SSLSocket can be a client socket or a server socket.

The SSLContext.wrap\_socket() adds the SSL layer to the socket.

'''

clientSocket.connect(mailServer)

#Fill in end

recv = clientSocket.recv(1024).decode()

print(recv) #Prints out first line message in output, beginning with 220!

if recv[:3] != '220':

print('220 reply not received from server.') #Tells us if a reply was not received.

# Send HELO command and print server response.

heloCommand = 'HELO Alice\r\n'

clientSocket.send(heloCommand.encode()) #Encodes our heloCommand.

recv1 = clientSocket.recv(1024).decode() #Decodes our heloCommand server response.

print(recv1) #Prints the response.

if recv1[:3] != '250':

print('250 reply not received from server.')

#Username and Password Info

username = "santiagoaswell@gmail.com" #This is just a made up account I used to protect my actual account.

password = "sui-4Cide1" #Password for made up account.

base64\_str = ("\x00"+username+"\x00"+password).encode() #Encodes username and password string!

base64\_str = base64.b64encode(base64\_str) #Resturns encoded bytes

authMsg = "AUTH PLAIN ".encode()+base64\_str+"\r\n".encode()

clientSocket.send(authMsg)

recv\_auth = clientSocket.recv(1024)

print(recv\_auth.decode()) #Lets us know if username and password were accepted and if not otherwise.

if recv1[:3] != '250':

print('250 reply not received from server.')

# Send MAIL FROM command and print server response.

# Fill in start

mailFromCommand = 'Mail From: <santiagoaswell@gmail.com>\r\n'

clientSocket.send(mailFromCommand.encode()) #Encodes our mailFromCommand.

recv2 = clientSocket.recv(1024)

# .recv will return as long as the network buffers have bytes.

# If bytes in the network buffers are more than .recv can handle,

# it will return the maximum number of bytes it can handle.

# If bytes in the network buffers are less than socket.recv can

# handle, it will return all the bytes in the network buffers.

recv2 = recv2.decode() #Decodes our mailFromCommand server response.

print("After mail from: " + recv2) #Prints the response for mailFromCommand.

if recv2[:3] != '250':

print('250 reply not received from server.')

# Fill in end

# Send RCPT TO command and print server response.

# Fill in start

rcptToCommand = 'RCPT TO: <santiagoabermudez@csus.edu>\r\n'

#print(rcptToCommand)

clientSocket.send(rcptToCommand.encode()) #Encodes our rcptToCommand.

recv3 = clientSocket.recv(1024) #Same deal as above.

recv3 = recv3.decode() #Decodes our rcptToCommand server response.

print("After RCPT: " + recv3) #Prints the response for rcptToCommand.

# Fill in end

# Send DATA command and print server response.

# Fill in start

dataCommand = 'DATA\r\n'

clientSocket.send(dataCommand.encode()) #Encodes our dataCommand.

recv4 = clientSocket.recv(1024) #Same deal as above.

recv4 = recv4.decode() #Decodes our dataCommand server response.

print("After DATA command: " + recv4) #Prints the response for dataCommand.

# Fill in end

# Send message data.

# Fill in start

msgSubject = "Subject: Testing my Client\r\n\r\n"

clientSocket.send(msgSubject.encode())

date = time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.gmtime())

date = date + "\r\n\r\n"

# Fill in end

# Message ends with a single period.

# Fill in start

clientSocket.send(date.encode()) #Encodes date details that are sent.

clientSocket.send(msg.encode()) #Encodes message that is sent.

clientSocket.send(endmsg.encode()) #Encodes message terminator.

recv\_msg = clientSocket.recv(1024)

print("Response after sending message body: " + recv\_msg.decode()) #Prints the response for sending message data.

# Fill in end

# Send QUIT command and get server response.

# Fill in start

quitCommand = "QUIT\r\n"

clientSocket.send(quitCommand.encode()) #Encodes the quitCommand.

recv5 = clientSocket.recv(1024) #Same deal as above.

print(recv5.decode()) #Prints the response for quitCommand.

clientSocket.close() #Close connection.

# Fill in end