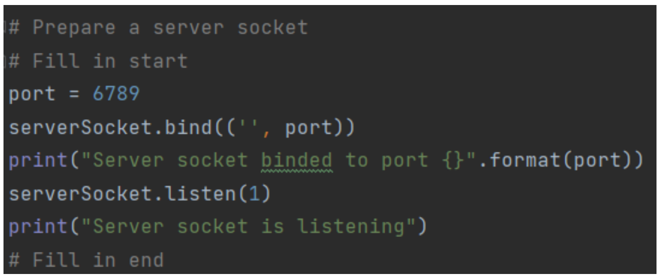
CSE 476 – MOBILE COMMUNICATION NETWORKS
TERM PROJECT ASSIGNMENTS
REPORT

Assignment 1: Web Server

In this assignment, I will develop a simple Web server in Python that is capable of processing only one request.

- Prepare a server socket

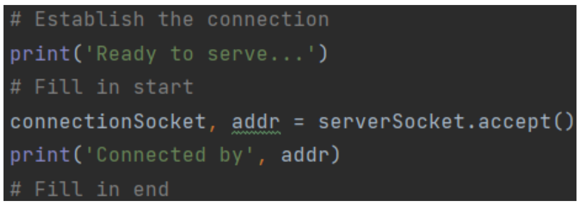
  I bind server socket to port 6789 with using bind() function. Later, I used listen() function for the socket's listening mode. I used 1 as the parameter of the listen function because the simple web server is capable of processing only one request.



```
# Prepare a server socket
# Fill in start
port = 6789
serverSocket.bind(('', port))
print("Server socket binded to port {}".format(port))
serverSocket.listen(1)
print("Server socket is listening")
# Fill in end
```

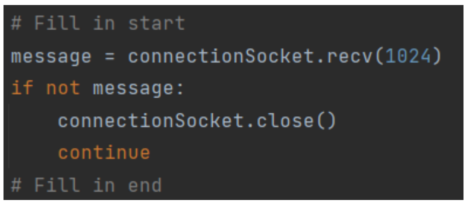
- Establish the connection

  Web server creates a connection socket when contacted by a client (browser). I established the connection with using accept() function.



```
# Establish the connection
print('Ready to serve...')
# Fill in start
connectionSocket, addr = serverSocket.accept()
print('Connected by', addr)
# Fill in end
```

- Receive the HTTP request from this connection



```
# Fill in start
message = connectionSocket.recv(1024)
if not message:
    connectionSocket.close()
    continue
# Fill in end
```

  If there is no message, the connection socket is closed and the loop continues.

- Create an HTTP response message (output message) consisting of the requested file
  The requested file is read with using read() function.

```
# Fill in start
outputdata = f.read()
print(outputdata)
# Fill in end
```

- Send one HTTP header line into socket
  The HTTP header line is sent into socket with using send() function.

```
# Send one HTTP header line into socket
# Fill in start
header_line = "HTTP/1.1 200 OK\r\n\r\n"
connectionSocket.send(header_line.encode())
# Fill in end
```

- Send response message for file not found
  First, an http header line is sent into socket. Then the response message consisting of the error message is sent into socket.

```
# Send response message for file not found
# Fill in start
response_header = "HTTP/1.1 404 Not Found\r\n\r\n"
connectionSocket.send(response_header.encode())
response_message = '<!DOCTYPE html><html><head><title> 404 Not Found </title></head><body><h1> 404 Not Found </h1></body></html>\r\n'
connectionSocket.send(response_message.encode())
# Fill in end
```

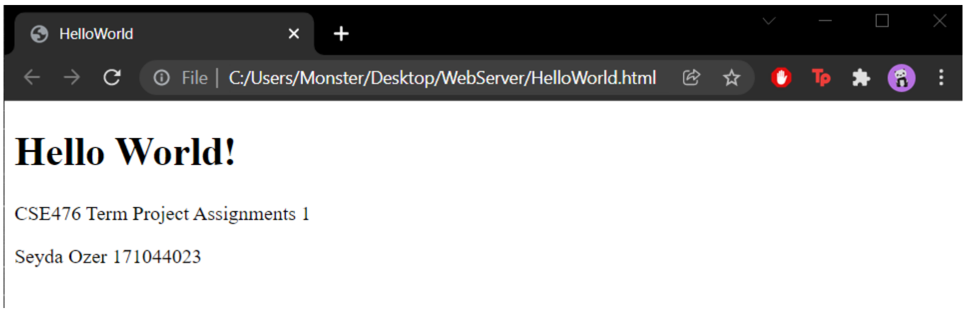
- Close client socket

```
#Close client socket
#Fill in start
connectionSocket.close()
#Fill in end
```

HelloWorld.html

The HTML file is in the same directory that the server is in.

```
<!DOCTYPE html>
<html>
<head>
    <title>HelloWorld</title>
</head>
<body>
<h1> Hello World! </h1>
<p> CSE476 Term Project Assignments 1 </p>
<p> Seyda Ozer 171044023 </p>
</body>
</html>
```

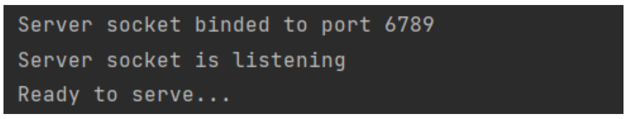


Hello World!

CSE476 Term Project Assignments 1

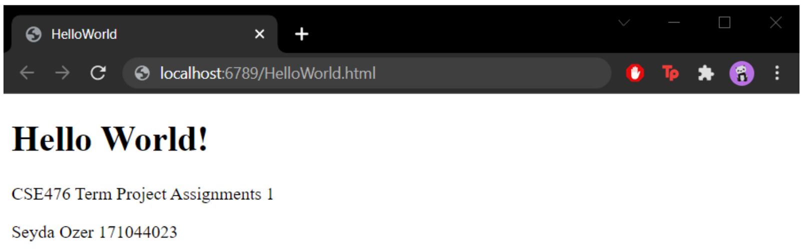
Seyda Ozer 171044023

Test for HelloWorld.html in the same computer:
Run the server program:



```
Server socket binded to port 6789
Server socket is listening
Ready to serve...
```

The browser:

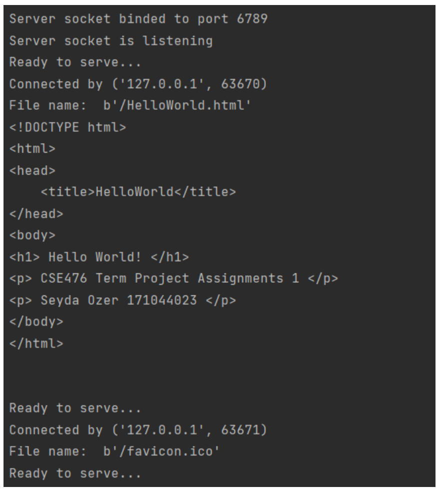


Hello World!

CSE476 Term Project Assignments 1

Seyda Ozer 171044023

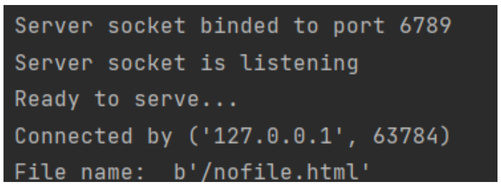
Server:



```
Server socket binded to port 6789
Server socket is listening
Ready to serve...
Connected by ('127.0.0.1', 63670)
File name:  b'/HelloWorld.html'
<!DOCTYPE html>
<html>
<head>
    <title>HelloWorld</title>
</head>
<body>
<h1> Hello World! </h1>
<p> CSE476 Term Project Assignments 1 </p>
<p> Seyda Ozer 171044023 </p>
</body>
</html>


Ready to serve...
Connected by ('127.0.0.1', 63671)
File name:  b'/favicon.ico'
Ready to serve...
```

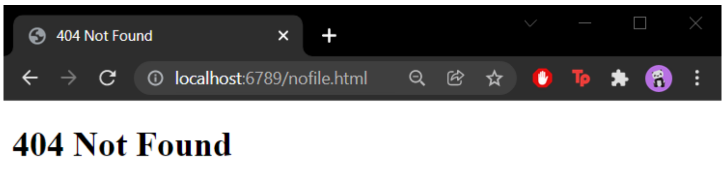
Test for nofile.html in the same computer:
This file does not exist.

Server:



```
Server socket binded to port 6789
Server socket is listening
Ready to serve...
Connected by ('127.0.0.1', 63784)
File name:  b'/nofile.html'
```

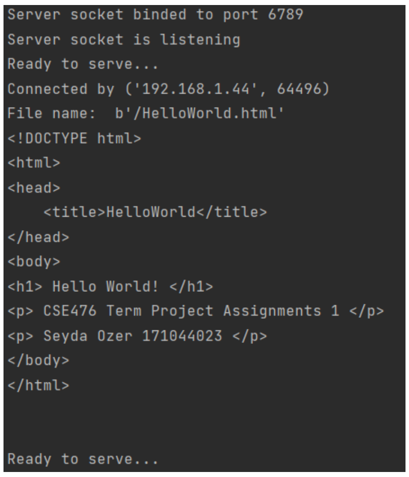
Error page:



404 Not Found

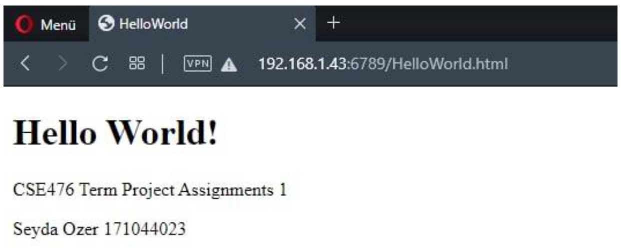
Test for HelloWorld.html in the different computer:

Server ip address: 192.168.1.43



```
Server socket binded to port 6789
Server socket is listening
Ready to serve...
Connected by ('192.168.1.44', 64496)
File name:  b'/HelloWorld.html'
<!DOCTYPE html>
<html>
<head>
    <title>HelloWorld</title>
</head>
<body>
<h1> Hello World! </h1>
<p> CSE476 Term Project Assignments 1 </p>
<p> Seyda Ozer 171044023 </p>
</body>
</html>


Ready to serve...
```

Browser:



Hello World!

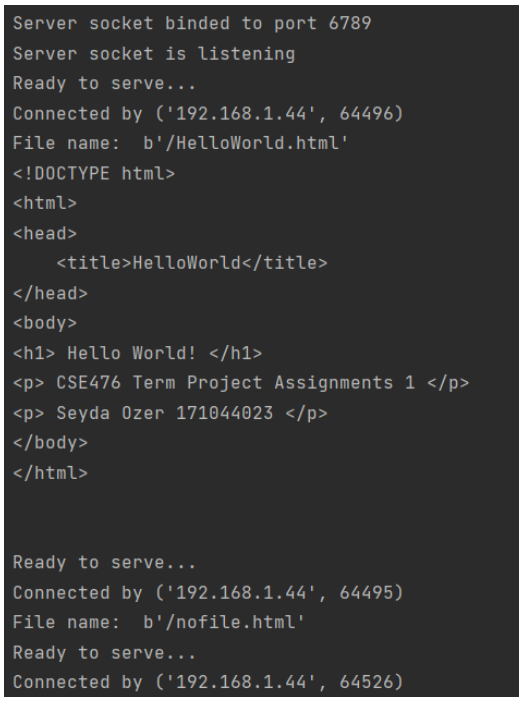
CSE476 Term Project Assignments 1

Seyda Ozer 171044023

Test for nofile.html in the different computer:

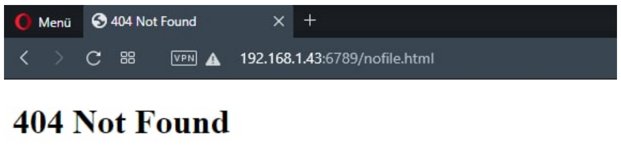
Server ip address: 192.168.1.43

This file does not exist.



```
Server socket binded to port 6789
Server socket is listening
Ready to serve...
Connected by ('192.168.1.44', 64496)
File name:  b'/HelloWorld.html'
<!DOCTYPE html>
<html>
<head>
    <title>HelloWorld</title>
</head>
<body>
<h1> Hello World! </h1>
<p> CSE476 Term Project Assignments 1 </p>
<p> Seyda Ozer 171044023 </p>
</body>
</html>


Ready to serve...
Connected by ('192.168.1.44', 64495)
File name:  b'/nofile.html'
Ready to serve...
Connected by ('192.168.1.44', 64526)
```

Browser:



192.168.1.43:6789/nofile.html

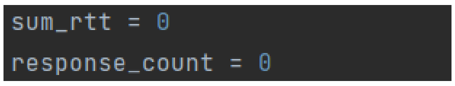
404 Not Found

Assignment 2: UDP Pinger

I implemented the UDP Pinger client program.

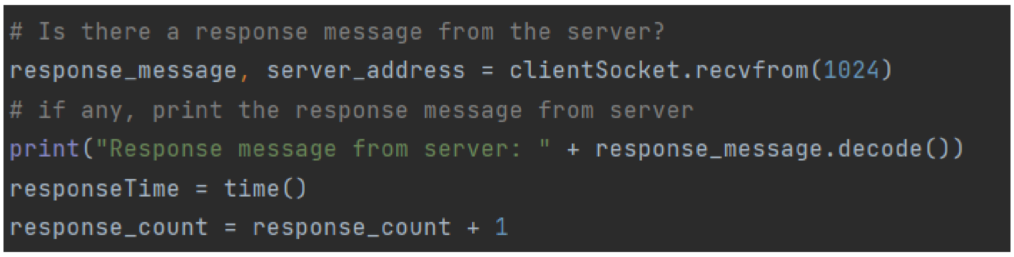
- The client send the ping message using UDP.
  Create a UDP socket for the client.
  ```
  # client socket
  clientSocket = socket(AF_INET, SOCK_DGRAM)
  ```
  Message format: Ping sequence_number time
  This message is sent with sendto() function.
  ```
  sendTime = time()
  client_message = 'Ping ' + str(i) + ' ' + str(strftime("%H:%M:%S"))
  # send the ping message using UDP
  clientSocket.sendto(client_message.encode(), server_addr)
  print('Client send the message: ' + client_message)
  ```
- Print the respond message from server, if any
  The client is received message from server with recvfrom() function.
  ```
  # Is there a response message from the server?
  response_message, server_address = clientSocket.recvfrom(1024)
  # if any, print the response message from server
  print("Response message from server: " + response_message.decode())
  responseTime = time()
  ```
- Calculate and print the round trip time (RTT), in seconds, of each packet, if server responses
  ```
  # calculate and print RTT, if server responses
  rtt = responseTime - sendTime
  print('Round Trip Time (RTT): ' + str(rtt) + ' seconds')
  ```
- If the server does not response
  The client cannot wait indefinitely for a reply to a ping message. The client wait up to one second for a reply.
  Set the timeout value on the client socket
  ```
  # set the timeout value on the clientSocket
  clientSocket.settimeout(1)
  ```
  ```
  # otherwise, print "Request timed out"
  except timeout:
      print('Request timed out')
  ```

- Optional exercise 1: Report the minimum, maximum, and average RTTs at the end of all pings and calculate the packet loss rate (in percentage)
  I used sum_rtt variable for summation of the all RTTs values and response_count variable for counting the response messages from server.



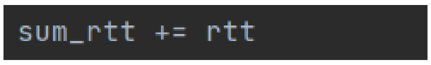
```
sum_rtt = 0
response_count = 0
```

I increment the response_count after each response.



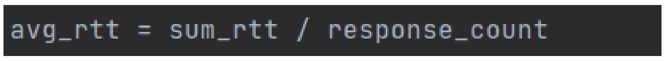
```
# Is there a response message from the server?
response_message, server_address = clientSocket.recvfrom(1024)
# if any, print the response message from server
print("Response message from server: " + response_message.decode())
responseTime = time()
response_count = response_count + 1
```

sum_rtt variable is summed with each RTT value in the loop. Thus, the total RTT value is obtained.



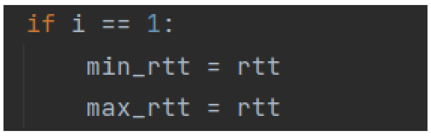
```
sum_rtt += rtt
```

sum_rtt is used for calculating average rtt.



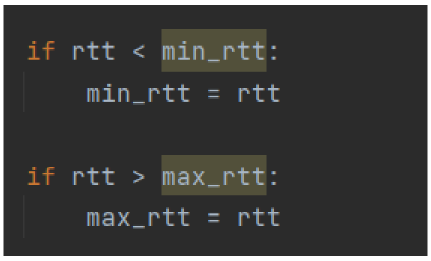
```
avg_rtt = sum_rtt / response_count
```

Initially, min_rtt and max_rtt variables keep the first rtt.



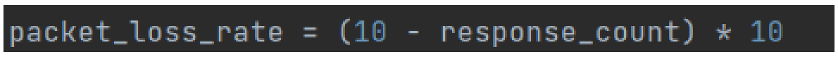
```
if i == 1:
    min_rtt = rtt
    max_rtt = rtt
```

Later the minimum rtt and maximum rtt are find in the for loop.



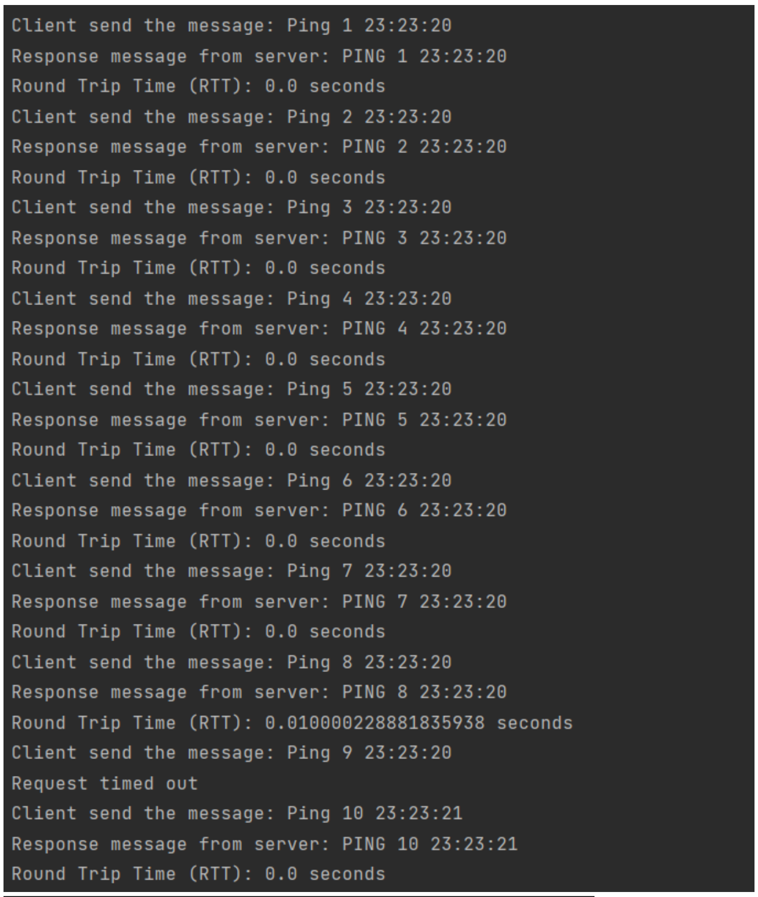
```
if rtt < min_rtt:
    min_rtt = rtt

if rtt > max_rtt:
    max_rtt = rtt
```

Packet loss rate = Missing message * 10

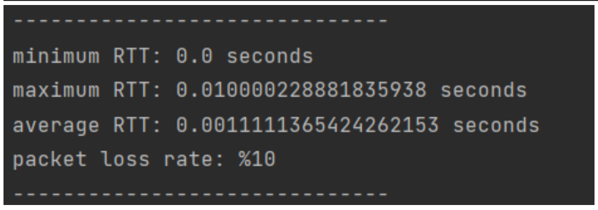


```
packet_loss_rate = (10 - response_count) * 10
```

Test



```
Client send the message: Ping 1 23:23:20
Response message from server: PING 1 23:23:20
Round Trip Time (RTT): 0.0 seconds
Client send the message: Ping 2 23:23:20
Response message from server: PING 2 23:23:20
Round Trip Time (RTT): 0.0 seconds
Client send the message: Ping 3 23:23:20
Response message from server: PING 3 23:23:20
Round Trip Time (RTT): 0.0 seconds
Client send the message: Ping 4 23:23:20
Response message from server: PING 4 23:23:20
Round Trip Time (RTT): 0.0 seconds
Client send the message: Ping 5 23:23:20
Response message from server: PING 5 23:23:20
Round Trip Time (RTT): 0.0 seconds
Client send the message: Ping 6 23:23:20
Response message from server: PING 6 23:23:20
Round Trip Time (RTT): 0.0 seconds
Client send the message: Ping 7 23:23:20
Response message from server: PING 7 23:23:20
Round Trip Time (RTT): 0.0 seconds
Client send the message: Ping 8 23:23:20
Response message from server: PING 8 23:23:20
Round Trip Time (RTT): 0.010000228881835938 seconds
Client send the message: Ping 9 23:23:20
Request timed out
Client send the message: Ping 10 23:23:21
Response message from server: PING 10 23:23:21
Round Trip Time (RTT): 0.0 seconds
```



```
------------------------------
minimum RTT: 0.0 seconds
maximum RTT: 0.010000228881835938 seconds
average RTT: 0.0011111365424262153 seconds
packet loss rate: %10
------------------------------
```

Assignment 3: Mail Client

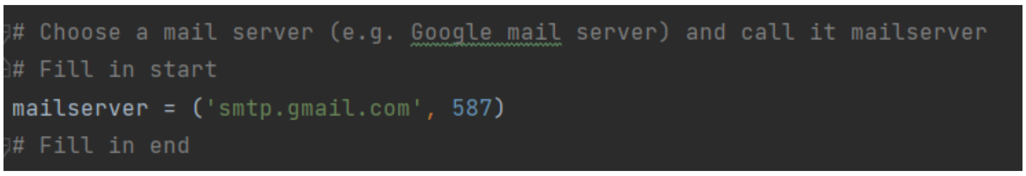
The goal of this programming assignment is to create a simple mail client that sends email to any recipient.

The SMTP Protocol:

1. Opening the connection
2. Using a secure connection (TLS)
3. Creating the Email
4. Authenticating with Gmail
5. Sending the Email

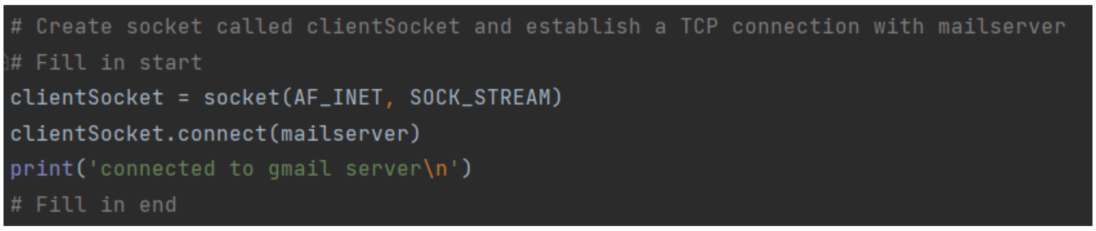
These steps are in my code as a comment line.

- Choose a mail server



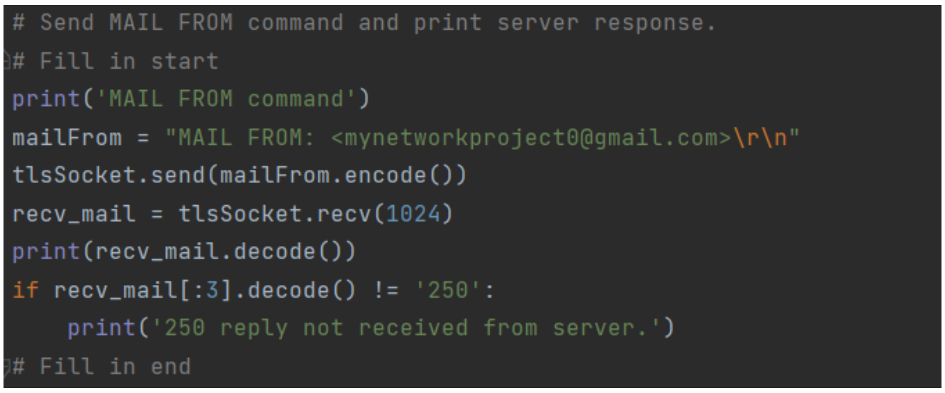
```
# Choose a mail server (e.g. Google mail server) and call it mailserver
# Fill in start
mailserver = ('smtp.gmail.com', 587)
# Fill in end
```

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



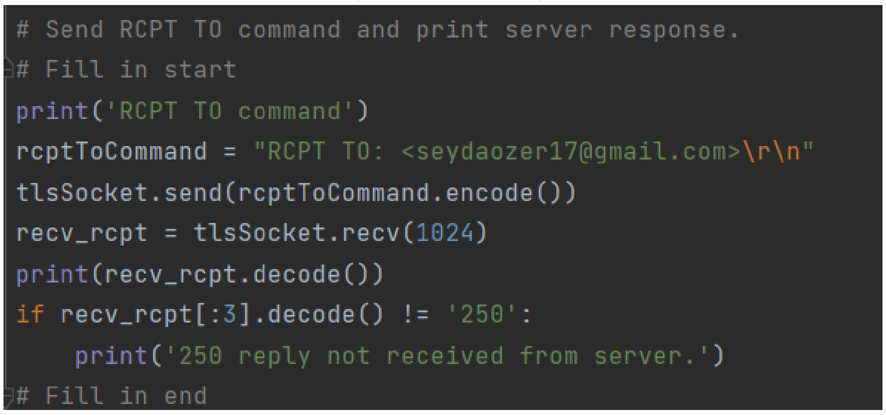
```
# Create socket called clientSocket and establish a TCP connection with mailserver
# Fill in start
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect(mailserver)
print('connected to gmail server\n')
# Fill in end
```

- Send MAIL FROM command and print server response.



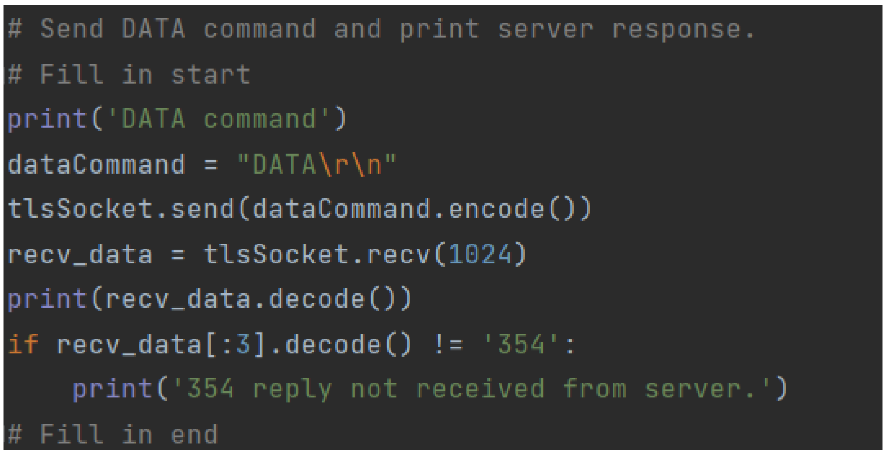
```
# Send MAIL FROM command and print server response.
# Fill in start
print('MAIL FROM command')
mailFrom = "MAIL FROM: <mynetworkproject0@gmail.com>\r\n"
tlsSocket.send(mailFrom.encode())
recv_mail = tlsSocket.recv(1024)
print(recv_mail.decode())
if recv_mail[:3].decode() != '250':
    print('250 reply not received from server.')
# Fill in end
```

- Send RCPT TO command and print server response.



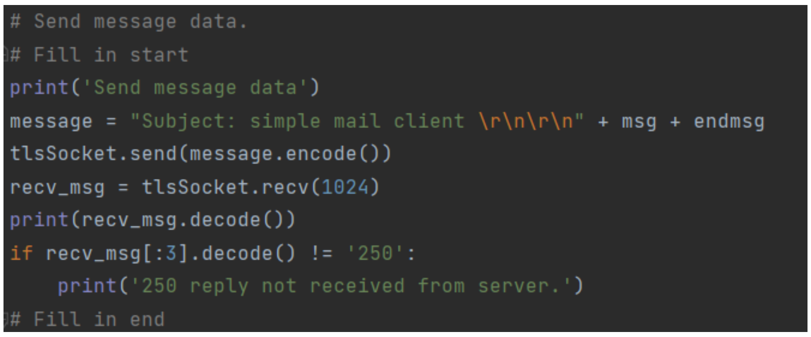
```
# Send RCPT TO command and print server response.
# Fill in start
print('RCPT TO command')
rcptToCommand = "RCPT TO: <seydaozer17@gmail.com>\r\n"
tlsSocket.send(rcptToCommand.encode())
recv_rcpt = tlsSocket.recv(1024)
print(recv_rcpt.decode())
if recv_rcpt[:3].decode() != '250':
    print('250 reply not received from server.')
# Fill in end
```

- Send DATA command and print server response.



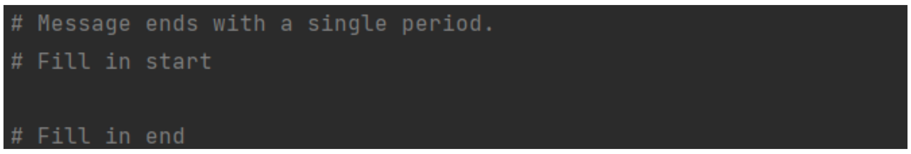
```
# Send DATA command and print server response.
# Fill in start
print('DATA command')
dataCommand = "DATA\r\n"
tlsSocket.send(dataCommand.encode())
recv_data = tlsSocket.recv(1024)
print(recv_data.decode())
if recv_data[:3].decode() != '354':
    print('354 reply not received from server.')
# Fill in end
```

- Send message data



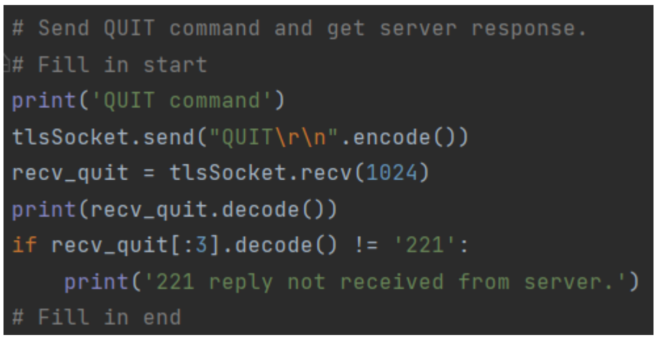
```
# Send message data.
# Fill in start
print('Send message data')
message = "Subject: simple mail client \r\n\r\n" + msg + endmsg
tlsSocket.send(message.encode())
recv_msg = tlsSocket.recv(1024)
print(recv_msg.decode())
if recv_msg[:3].decode() != '250':
    print('250 reply not received from server.')
# Fill in end
```

- I do not understand this part:



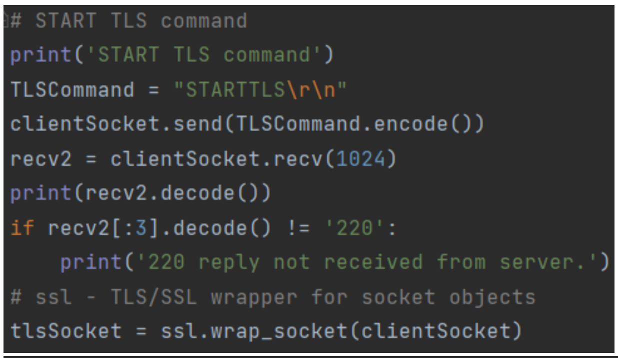
```
# Message ends with a single period.
# Fill in start

# Fill in end
```

- Send QUIT command and get server response.

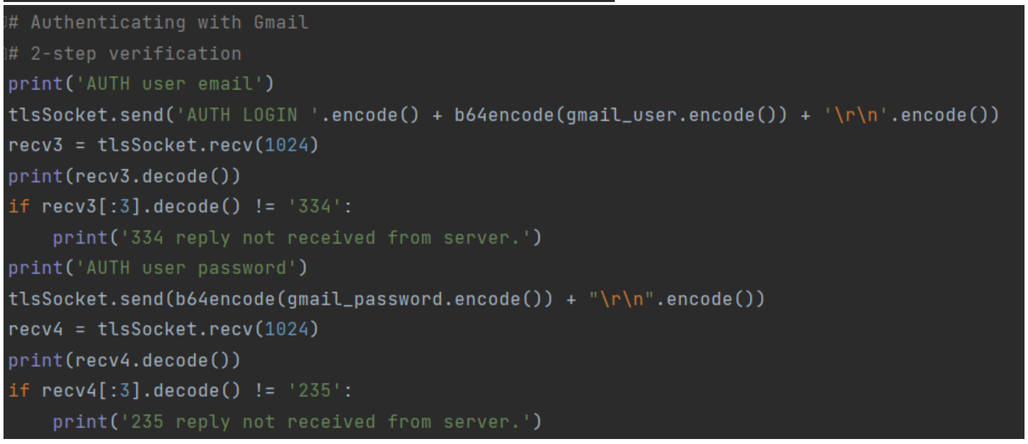


```
# Send QUIT command and get server response.
# Fill in start
print('QUIT command')
tlsSocket.send("QUIT\r\n".encode())
recv_quit = tlsSocket.recv(1024)
print(recv_quit.decode())
if recv_quit[:3].decode() != '221':
    print('221 reply not received from server.')
# Fill in end
```

- Optional exercise 1: Transport Layer Security (TLS) for authentication and security reasons, before sending MAIL FROM command



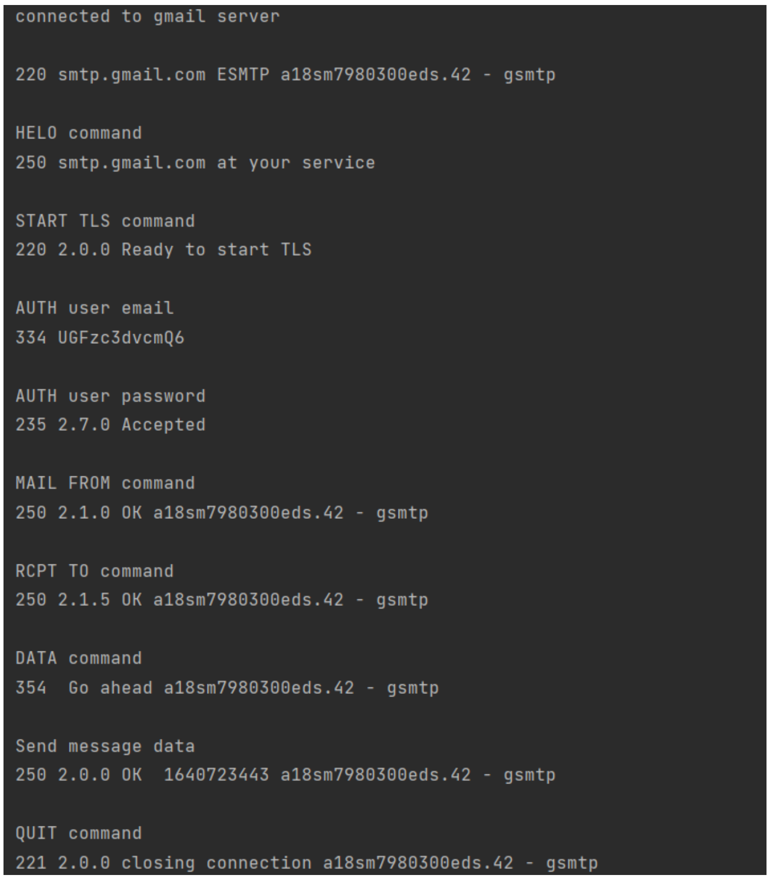
```
# START TLS command
print('START TLS command')
TLSCommand = "STARTTLS\r\n"
clientSocket.send(TLSCommand.encode())
recv2 = clientSocket.recv(1024)
print(recv2.decode())
if recv2[:3].decode() != '220':
    print('220 reply not received from server.')
# ssl - TLS/SSL wrapper for socket objects
tlsSocket = ssl.wrap_socket(clientSocket)
```



```
# Authenticating with Gmail
# 2-step verification
print('AUTH user email')
tlsSocket.send('AUTH LOGIN '.encode() + b64encode(gmail_user.encode()) + '\r\n'.encode())
recv3 = tlsSocket.recv(1024)
print(recv3.decode())
if recv3[:3].decode() != '334':
    print('334 reply not received from server.')
print('AUTH user password')
tlsSocket.send(b64encode(gmail_password.encode()) + "\r\n".encode())
recv4 = tlsSocket.recv(1024)
print(recv4.decode())
if recv4[:3].decode() != '235':
    print('235 reply not received from server.')
```

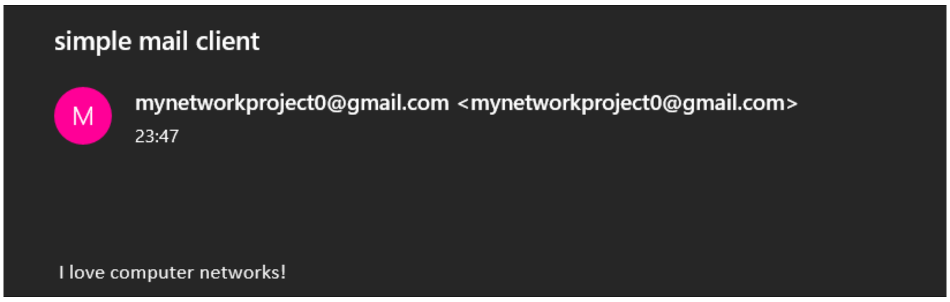
Test:

Output:



```
connected to gmail server

220 smtp.gmail.com ESMTP a18sm7980300eds.42 - gsmtp

HELO command
250 smtp.gmail.com at your service

START TLS command
220 2.0.0 Ready to start TLS

AUTH user email
334 UGFzc3dvcmQ6

AUTH user password
235 2.7.0 Accepted

MAIL FROM command
250 2.1.0 OK a18sm7980300eds.42 - gsmtp

RCPT TO command
250 2.1.5 OK a18sm7980300eds.42 - gsmtp

DATA command
354  Go ahead a18sm7980300eds.42 - gsmtp

Send message data
250 2.0.0 OK  1640723443 a18sm7980300eds.42 - gsmtp

QUIT command
221 2.0.0 closing connection a18sm7980300eds.42 - gsmtp
```

Mail:



simple mail client

M mynetworkproject0@gmail.com <mynetworkproject0@gmail.com>

23:47

I love computer networks!

Şeyda Özer

171044023