

# CSE476 – Mobile Communication Networks

## PROJECT REPORT

Furkan ÖZEV

161044036

### ASSIGNMENT 1: Web Server

I developed a simple Web server in Python that is capable of processing only one request.

My Web server create a connection socket when contacted by a client (browser).

It receive the HTTP request from this connection;

It parse the request to determine the specific file being requested

It get the requested file from the server's file system.

It create an HTTP response message consisting of the requested file preceded by header lines;

Then, It send the response over the TCP connection to the requesting browser.

If a browser requests a file that is not present in my server, my server will return a "404 Not Found" error message.

### CODE FILL IN START:

Server tcp port is 6789.

```
port = 6789
```

It create a socket, bind it to a specific address and tcp port with using bind() function, Then the socket has been put into listening mode with using listen() function.

```
#Prepare a server socket
#Fill in start
serverSocket.bind('', port)
print ("Server socket binded to %s" %(port))
# put the socket into listening mode
serverSocket.listen(1) #5
print ("Server socket is listening")
#Fill in end
```

The connection has been established with using accept() function. It will listen requests.

```
#Fill in start
connectionSocket, addr = serverSocket.accept()
print("Connection address:", addr)
#Fill in end
```

If there is an empty request, ignore it. Because sometimes there are empty requests originating from the browser.

```
if not message:
    connectionSocket.close()
    continue;
```

The requested file is read with using read() function.

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

An HTTP header line is sent to the socket with using send() function. At this stage, the connection is tested. The message to be sent is encrypted with the encode() function.

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

The content of the requested file is sent to the client with using send() function. The message to be sent is encrypted with the encode() function.

```
#Send the content of the requested file to the client
for i in range(0, len(outputdata)):
    connectionSocket.send(outputdata[i].encode())
connectionSocket.close()
```

A reply message is sent to the client as the file cannot be found with using send() function. This message is an html file that is error page.

```
#Send response message for file not found
#Fill in start
mes = "HTTP/1.1 404 Not Found\r\n\r\n"
connectionSocket.send(mes.encode())

mes = "<html><head><title> 404 </title></head><body><h1>404 Not Found!</h1></body></html>\r\n"
connectionSocket.send(mes.encode())
#Fill in end
```

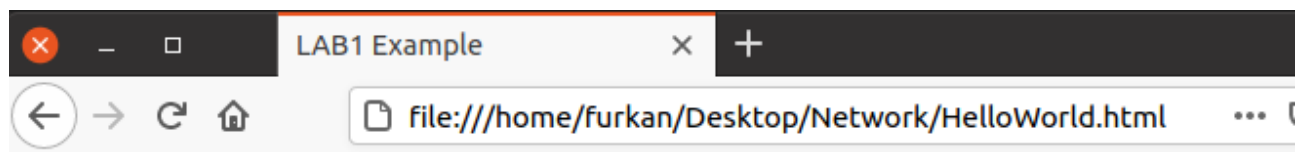
Client socket is closed with using close() function.

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

## OUTPUT AND DEMO:

My html file name is “HelloWorld.html” .

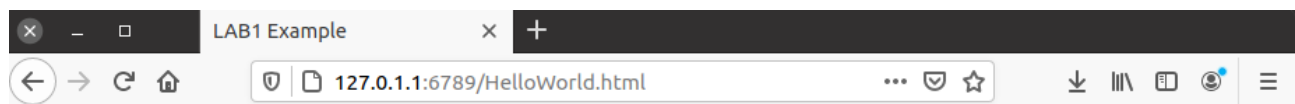
```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>LAB1 Example</title>
5  </head>
6  <body>
7      <h1>Hello World!</h1>
8      <p>This Lab1 Example<p>
9  </body>
10 </html>
```



# Hello World!

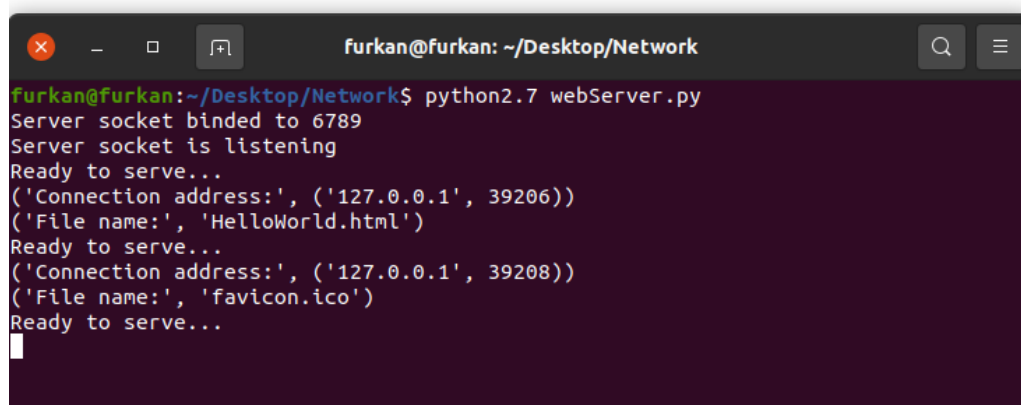
This Lab1 Example

## First Attemp In Same Computer for HelloWorld.html:

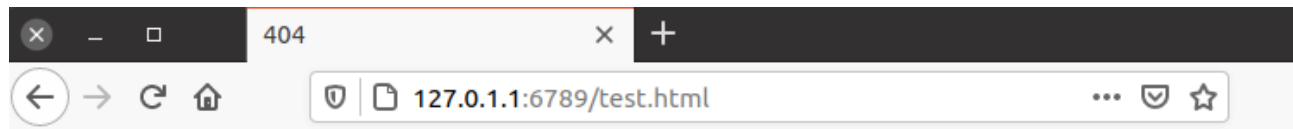


# Hello World!

This Lab1 Example



**Second Attempt In Same Computer for test.html. This file does not exist:**

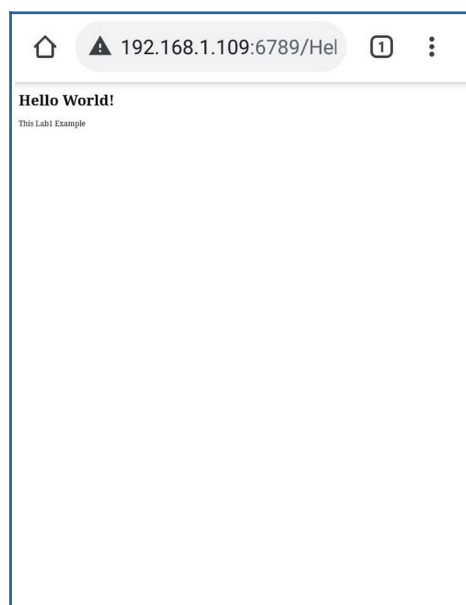


# 404 Not Found!

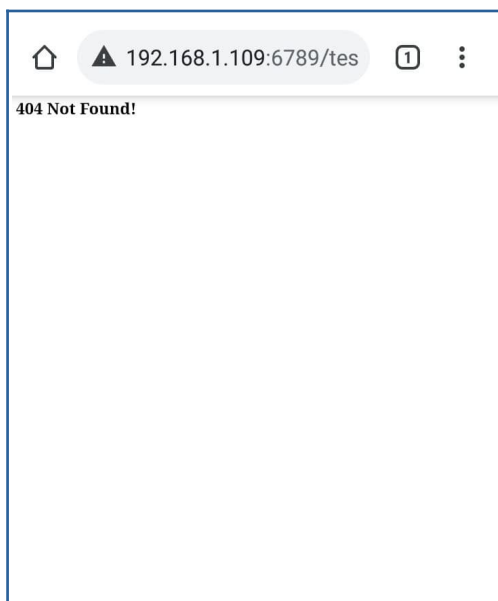
```
furkan@furkan: ~/Desktop/Network
furkan@furkan:~/Desktop/Network$ python2.7 webServer.py
Server socket binded to 6789
Server socket is listening
Ready to serve...
('Connection address:', ('127.0.0.1', 39206))
('File name:', 'HelloWorld.html')
Ready to serve...
('Connection address:', ('127.0.0.1', 39208))
('File name:', 'favicon.ico')
Ready to serve...
('Connection address:', ('127.0.0.1', 39214))
('File name:', 'test.html')
Ready to serve...
```

**First Attempt In Different Computer for HelloWorld.html:**

My server ip is 192.168.1.109



**Second Attempt In Different Computer for test.html. This file does not exist:**



```
furkan@furkan:~/Desktop/Network$ python2.7 webServer.py
Server socket binded to 6789
Server socket is listening
Ready to serve...
('Connection address:', ('192.168.1.104', 49595))
('File name:', 'HelloWorld.html')
Ready to serve...
('Connection address:', ('192.168.1.104', 49596))
('File name:', 'favicon.ico')
Ready to serve...
('Connection address:', ('192.168.1.104', 49597))
('File name:', 'HelloWorld.html')
Ready to serve...
('Connection address:', ('192.168.1.104', 49598))
('File name:', 'test.html')
Ready to serve...
('Connection address:', ('192.168.1.104', 49599))
```

## ASSIGNMENT 2: UDP Pinger

I developed a client ping program.

My client send a simple ping message to a server, and it receive corresponding message back from the server.

Then it determine the delay between when the client sent the ping message and received the pong message. This delay is called the Round Trip Time (RTT).

My client program is to send 10 ping messages to the target server over UDP.

For each message, It determine and print RTT when the corresponding pong message is returned.

Client wait up to one second for a reply from server. If no reply is received, the client assume that the packet was lot and print a message accordingly.

In summary, Client program send the ping messages using UDP protocol; Print the response message from server, if any; Calculate and print the RTT in seconds, of each packet, if server responses; Otherwise, print "Request timed out" message.

Ping message format: Ping (number) (time)

### CLIENT CODE:

Import necessary modules.

Socket module for communication between server and client.

Time module to get current time. It will be used for calculating RTT.

```
from socket import *  
from time import *
```

Create UDP socket on localhost.

```
clientSocket = socket(AF_INET, SOCK_DGRAM)  
serverAddr = ('localhost', 12000)
```

It set the timeout value on a datagram socket. Client wait up to one second for a reply from server.

```
clientSocket.settimeout(1)
```

Client will ping the server 10 times in the loop.

Time information of the message inbound and outbound is required for the RTT calculation. Therefore, time () function was used to get instant time data.

Using the time () function, time information is stored when the ping message goes to the server.

Ping message format: Ping (number) (time)

Time information is formatted as H: M: S using the strftime () function.

The message is encrypted using the encode () function.

Using the sendto () function, the ping message is sent to the server address.

This ping message is printed on the terminal.

The client gets the response message from the socket using the recvfrom () function.

It decodes the response message with the decode () function and prints it to the terminal.

Using the time () function, time information is stored when the response message comes to the server.

RTT is calculated by taking the differences of stored time information.

If no response message is received from the server, that is, if the timeout exception is thrown, the "Request (number) Time Out" message is written to the terminal.

Then close client socket.

```
i = 1
while (i < 11):
    try:
        sendTime = time()
        message = "PING " + str(i) + " " + str(strftime("%H:%M:%S"))
        clientSocket.sendto(message.encode(), serverAddr)
        print("Sent Message: " + message)

        data, server = clientSocket.recvfrom(1024)
        print("Response Message: " + data.decode())
        responseTime = time();

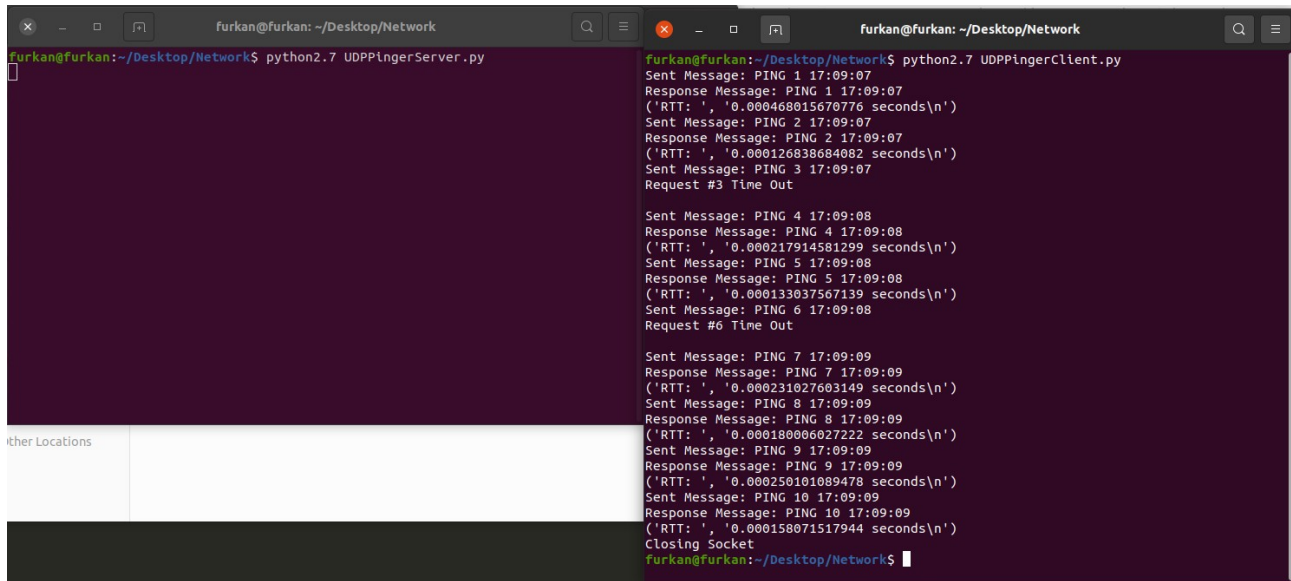
        elapsed = responseTime - sendTime;
        print("RTT: ", str(elapsed) + " seconds\n")

    except timeout:
        print("Request #" + str(i) + " Time Out\n")

    i += 1

print("Closing Socket")
clientSocket.close()
```

## OUTPUT AND DEMO - 1:



The screenshot shows two terminal windows side-by-side. The left window is titled 'furkan@furkan: ~/Desktop/Network' and shows the command 'python2.7 UDPPingerServer.py' being executed. The right window is also titled 'furkan@furkan: ~/Desktop/Network' and shows the command 'python2.7 UDPPingerClient.py' being executed. The output in the right window shows a series of ping requests and responses with RTT values, followed by a 'Request #3 Time Out' and 'Request #6 Time Out' message. The output ends with 'Closing Socket' and the prompt 'furkan@furkan:~/Desktop/Network\$'.

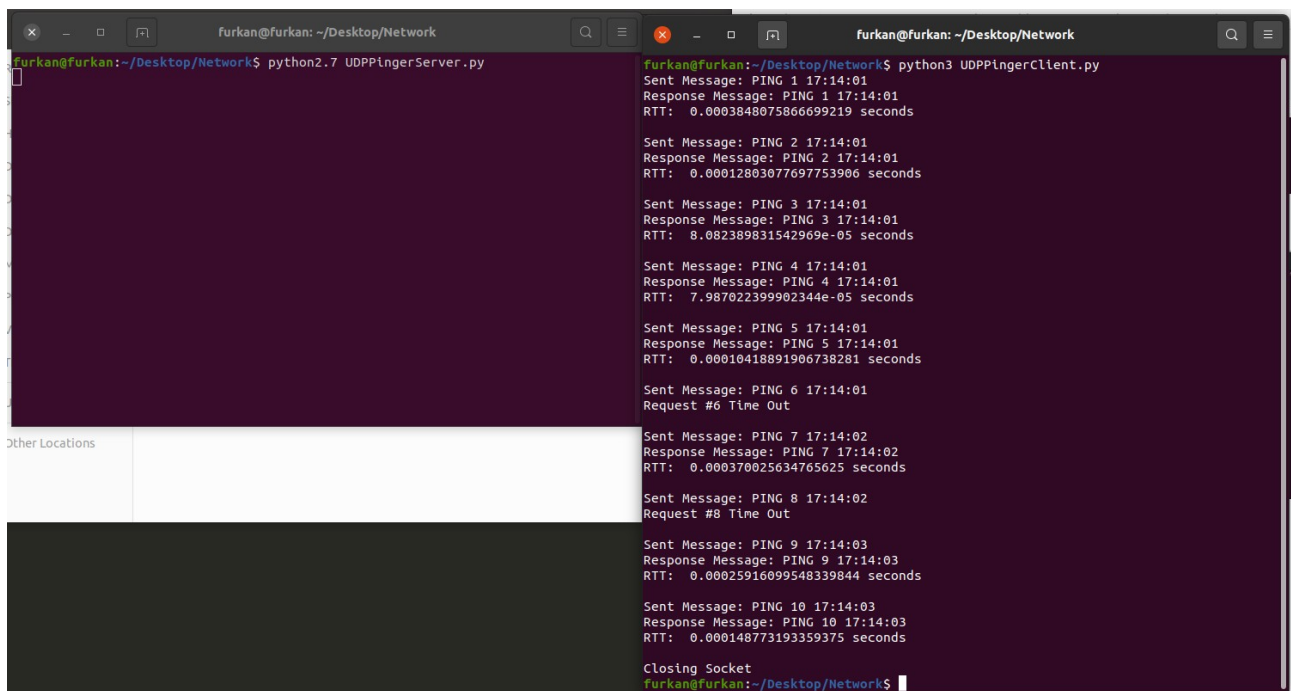
```
furkan@furkan:~/Desktop/Network$ python2.7 UDPPingerServer.py

furkan@furkan:~/Desktop/Network$ python2.7 UDPPingerClient.py
Sent Message: PING 1 17:09:07
Response Message: PING 1 17:09:07
('RTT: ', '0.000468015670776 seconds\n')
Sent Message: PING 2 17:09:07
Response Message: PING 2 17:09:07
('RTT: ', '0.000126838684082 seconds\n')
Sent Message: PING 3 17:09:07
Request #3 Time Out

Sent Message: PING 4 17:09:08
Response Message: PING 4 17:09:08
('RTT: ', '0.000217914581299 seconds\n')
Sent Message: PING 5 17:09:08
Response Message: PING 5 17:09:08
('RTT: ', '0.000133037567139 seconds\n')
Sent Message: PING 6 17:09:08
Request #6 Time Out

Sent Message: PING 7 17:09:09
Response Message: PING 7 17:09:09
('RTT: ', '0.000231027603149 seconds\n')
Sent Message: PING 8 17:09:09
Response Message: PING 8 17:09:09
('RTT: ', '0.00018006027222 seconds\n')
Sent Message: PING 9 17:09:09
Response Message: PING 9 17:09:09
('RTT: ', '0.000250101089478 seconds\n')
Sent Message: PING 10 17:09:09
Response Message: PING 10 17:09:09
('RTT: ', '0.000158071517944 seconds\n')
Closing Socket
furkan@furkan:~/Desktop/Network$
```

## OUTPUT AND DEMO – 2:



The screenshot shows two terminal windows side-by-side. The left window is titled 'furkan@furkan: ~/Desktop/Network' and shows the command 'python2.7 UDPPingerServer.py' being executed. The right window is also titled 'furkan@furkan: ~/Desktop/Network' and shows the command 'python3 UDPPingerClient.py' being executed. The output in the right window shows a series of ping requests and responses with RTT values, followed by a 'Request #6 Time Out' and 'Request #8 Time Out' message. The output ends with 'Closing Socket' and the prompt 'furkan@furkan:~/Desktop/Network\$'.

```
furkan@furkan:~/Desktop/Network$ python2.7 UDPPingerServer.py

furkan@furkan:~/Desktop/Network$ python3 UDPPingerClient.py
Sent Message: PING 1 17:14:01
Response Message: PING 1 17:14:01
RTT: 0.0003848075866699219 seconds

Sent Message: PING 2 17:14:01
Response Message: PING 2 17:14:01
RTT: 0.00012803077697753906 seconds

Sent Message: PING 3 17:14:01
Response Message: PING 3 17:14:01
RTT: 8.082389831542969e-05 seconds

Sent Message: PING 4 17:14:01
Response Message: PING 4 17:14:01
RTT: 7.987022399902344e-05 seconds

Sent Message: PING 5 17:14:01
Response Message: PING 5 17:14:01
RTT: 0.00010418891906738281 seconds

Sent Message: PING 6 17:14:01
Request #6 Time Out

Sent Message: PING 7 17:14:02
Response Message: PING 7 17:14:02
RTT: 0.000370025634765625 seconds

Sent Message: PING 8 17:14:02
Request #8 Time Out

Sent Message: PING 9 17:14:03
Response Message: PING 9 17:14:03
RTT: 0.00025916099548339844 seconds

Sent Message: PING 10 17:14:03
Response Message: PING 10 17:14:03
RTT: 0.000148773193359375 seconds

Closing Socket
furkan@furkan:~/Desktop/Network$
```



### ASSIGNMENT 3: Mail Client

I developed a simple mail client that sends email to any recipient.

This client established an TCP connection with a gmail server. (Can be changed on request)

It dialogue with the mail server using the SMTP protocol.

It send an email message to a recipient via the mail server.

Then close the TCP connection with the mail server.

I also did the optional exercises:

Google mail server address: smtp.gmail.com      port: 587

I added a Transport Layer Security(TLS) and Secure Sockets Layer(SSL) for authentication.

I added TLS/SSL commands for using Google mail server.

### CODE FILL IN START:

Import necessary modules.

Socket module for communication between server and client.

Base64 module to encode email and password. It will be used for authentication.

Ssl module use to connect gmail smtp server.

```
from socket import *  
from base64 import *  
import ssl
```

Gmail port is 587

Sender account information for authentication.

You can change the sender by changing the information here.

```
port = 587  
username = "networkdeneme00@gmail.com"  
password = "Deneme123."
```

I chose gmail smtp server with 587 port.

```
#Choose a mail server (e.g. Googlemailserver) and call it mailserver  
#Fill in start  
mailserver = ("smtp.gmail.com", port)  
#Fill in end  
#Create socket called clientSocket and establish a TCP connection with mailserver
```

I create client socket and establish a TCP connection between mail server and client.

```
#Fill in start  
clientSocket = socket(AF_INET, SOCK_STREAM)  
clientSocket.connect(mailserver)  
#Fill in end
```

It send an HELO command.

This part is edited as gmail smtp helo command format.

Helo message: "EHLO smtp.gmail.com\r\n"

This message is encrypted with the encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
#Send HELO command and print server response.
heloCommand = 'EHLO smtp.gmail.com\r\n'
clientSocket.send(heloCommand.encode())
recv1 = clientSocket.recv(1024)
print(recv1)
if (recv1[:3] != '250'):
    print('250 reply not received from server.')
```

It send an TLS command.

This part is required for connect gmail smtp server because of security reasons.

TLS message: "STARTTLS\r\n"

This message is encrypted with the encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
TLSCommand = "STARTTLS\r\n"
clientSocket.send(TLSCommand.encode())
recv2 = clientSocket.recv(1024)
print(recv2)
if (recv2[:3] != '220'):
    print('220 reply not received from server.')
```

It make an ssl socket that wrapped client socket.

This socket is required for connect gmail smtp server because of security reasons.

```
clientSocket = ssl.wrap_socket(clientSocket)
```

It send an AUTH command.

This part is required for connect mail servers because authentication is required.

AUTH message: "AUTH LOGIN (Email address encrypted with base64)\r\n"

Firstly, this email address is encrypted with the encode () function. Then it is encrypted with b64encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

Then, password is encrypted with the encode () function. Then it is encrypted with b64encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
#Send AUTH command and print server response.
#AUTH with base64 encoded user name password
clientSocket.send("AUTH LOGIN " + b64encode(username.encode()) + "\r\n")
recv3 = clientSocket.recv(1024)
print(recv3)
if (recv3[:3] != "334"):
    print('334 reply not received from server.')

clientSocket.send(b64encode(password.encode()) + "\r\n")
recv4 = clientSocket.recv(1024)
print(recv4)
if (recv4[:3] != "235"):
    print('235 reply not received from server.')
```

It send an MAIL FROM command.

This part carries the sender information.

MAIL FROM message: "MAIL FROM: <sender mail address>\r\n"

This message is encrypted with the encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
#Send MAIL FROM command and print server response.
#Fill in start
mailFrom = "MAIL FROM: <networkdeneme00@gmail.com>\r\n"
clientSocket.send(mailFrom.encode())
recv5 = clientSocket.recv(1024)
print(recv5)
if (recv5[:3] != '250'):
    print('250 reply not received from server.')
#Fill in end
```

It send an RCP TO command.

This part carries the receiver information.

RCP TO message: "RCP TO: <receiver mail address>\r\n"

This message is encrypted with the encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
#Send RCPT TO command and print server response.
#Fill in start
rcptToCommand = "RCPT TO: <furkanozev@gmail.com>\r\n"
clientSocket.send(rcptToCommand.encode())
recv6 = clientSocket.recv(1024)
print(recv6)
if (recv6[:3] != '250'):
    print('250 reply not received from server.')
#Fill in end
```

It send an DATA command.

It indicates that the mail content information will come after the data command.

DATA message: "DATA\r\n"

This message is encrypted with the encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
#Send DATA command and print server response.
#Fill in start
dataCommand = "Data\r\n"
clientSocket.send(dataCommand.encode())
recv7 = clientSocket.recv(1024)
print(recv7)
if (recv7[:3] != '354'):
    print('354 reply not received from server.')
#Fill in end
```

This part carries the content information of the mail..

message: "SUBJECT: (mail subject)\n(mail content text, image or together)\r\n.\r\n"

This message is encrypted with the encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
#Send message data.
#Fill in start
message = "SUBJECT: SMTP Mail Client Test\nSMTP Mail Client Test Content" + msg + endmsg
clientSocket.send(message.encode())
recv8 = clientSocket.recv(1024)
print(recv8)
if (recv8[:3] != '250'):
    print('250 reply not received from server.')
#Fill in end
```

It send an QUIT command to terminate.

It indicates the termination of the connection.

QUIT message: "Quit\r\n"

This message is encrypted with the encode () function.

Then it is sent to the mail server with the send () function.

Then print server response message. If no response is received, the error message is printed on the terminal.

```
# Send QUIT command and get server response.
Quit = "Quit\r\n"
print(Quit)
clientSocket.send(Quit.encode())
recv9 = clientSocket.recv(1024)
print(recv9)
if (recv9[:3] != '221'):
    print('221 reply not received from server.')
print("Mail Sent")
```

The connection is terminated using the close () function.

```
clientSocket.close()
```

## OUTPUT AND DEMO – 1:

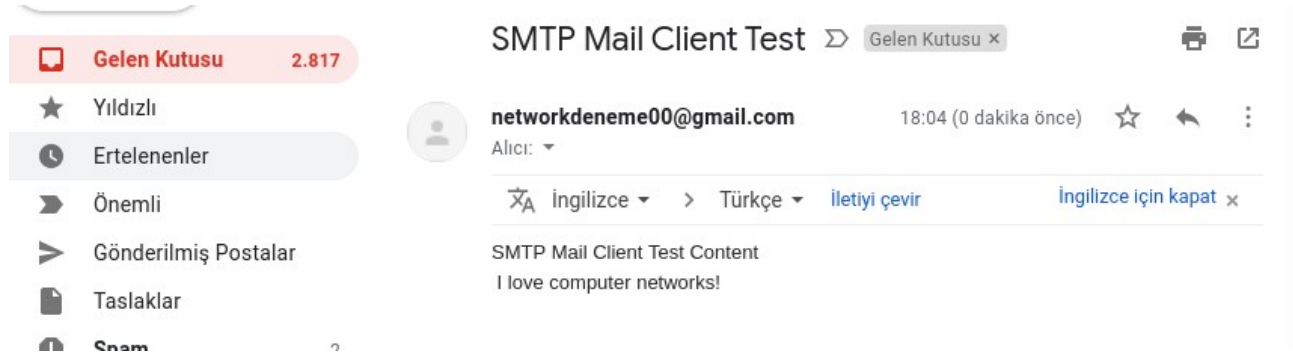
sender = "networkdeneme00@gmail.com"

receiver = "[furkanozev@gmail.com](mailto:furkanozev@gmail.com)"

**Sender:**



## Reciever:



## Terminal:

```
furkan@furkan:~/Desktop/Network$ python2.7 SMTPMailClient.py
220 smtp.gmail.com ESMTP h15sm28296653wrw.15 - gsmtip

250-smtp.gmail.com at your service, [159.146.42.169]
250-SIZE 35882577
250-8BITMIME
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8

220 2.0.0 Ready to start TLS

334 UGFzc3dvcmQ6

235 2.7.0 Accepted

250 2.1.0 OK h15sm28296653wrw.15 - gsmtip
250 2.1.5 OK h15sm28296653wrw.15 - gsmtip
354 Go ahead h15sm28296653wrw.15 - gsmtip

250 2.0.0 OK 1606748664 h15sm28296653wrw.15 - gsmtip

Quit

221 2.0.0 closing connection h15sm28296653wrw.15 - gsmtip

Mail Sent
```



## OUTPUT AND DEMO – 2:

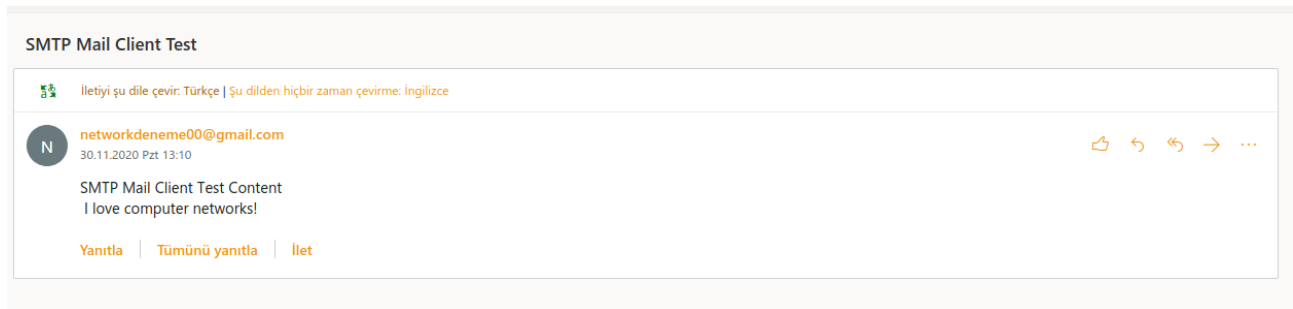
sender = "networkdeneme00@gmail.com"

receiver = "furkan.ozev2016@gtu.edu.tr"

### Sender:



### Reciever:



### Terminal:

```
furkan@furkan:~/Desktop/Network$ python2.7 SMTPMailClient.py
220 smtp.gmail.com ESMTP x10sm21896239wro.0 - gsmt
250-smtp.gmail.com at your service, [159.146.42.169]
250-SIZE 35882577
250-8BITMIME
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8

220 2.0.0 Ready to start TLS

334 UGFzc3dvcmQ6

235 2.7.0 Accepted

250 2.1.0 OK x10sm21896239wro.0 - gsmt
250 2.1.5 OK x10sm21896239wro.0 - gsmt
354 Go ahead x10sm21896239wro.0 - gsmt
250 2.0.0 OK 1606749030 x10sm21896239wro.0 - gsmt
Quit
221 2.0.0 closing connection x10sm21896239wro.0 - gsmt
Mail Sent
```

### OUTPUT AND DEMO – 3:

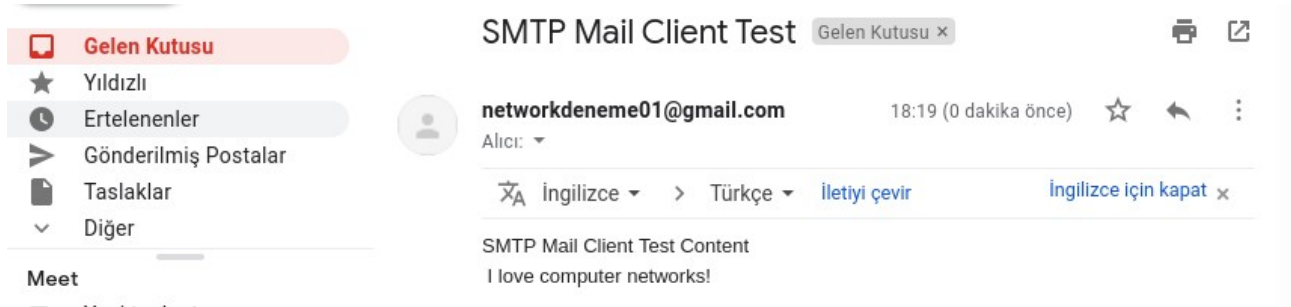
sender = "networkdeneme01@gmail.com"

receiver = "networkdeneme00@gmail.com"

#### Sender:



#### Reciever:



#### Terminal:

```
furkan@furkan:~/Desktop/Network$ python2.7 SMTPMailClient.py
220 smtp.gmail.com ESMTP b62sm17252700wmh.41 - gsmt
250-smtp.gmail.com at your service, [159.146.42.169]
250-SIZE 35882577
250-8BITMIME
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8
220 2.0.0 Ready to start TLS
334 UGFzc3dvcmQ6
235 2.7.0 Accepted
250 2.1.0 OK b62sm17252700wmh.41 - gsmt
250 2.1.5 OK b62sm17252700wmh.41 - gsmt
354 Go ahead b62sm17252700wmh.41 - gsmt
250 2.0.0 OK 1606749590 b62sm17252700wmh.41 - gsmt
Quit
221 2.0.0 closing connection b62sm17252700wmh.41 - gsmt
Mail Sent
```



## SETTINGS:

If you want to use different sender gmail account, you need to set some settings.

You must make the necessary permissions and settings on Google account and gmail settings.

### 1- Enable IMAP and POP

Q Search mail

?

⚙

☰

Settings

GeneralLabelsInboxAccounts and ImportFilters and Blocked AddressesForwarding and POP/IMAPAdd-onsChat and MeetAdvancedOfflineThemes

Forwarding:  
[Learn more](#)

Add a forwarding address

Tip: You can also forward only some of your mail by [creating a filter!](#)

POP download:  
[Learn more](#)

1. Status: **POP is enabled** for all mail

☐ Enable POP for **all mail** (even mail that's already been downloaded)

☐ Enable POP for **mail that arrives from now on**

☐ **Disable POP**

2. When messages are accessed with POP

keep Gmail's copy in the Inbox

3. Configure your email client (e.g. Outlook, Eudora, Netscape Mail)

[Configuration instructions](#)

IMAP access:  
(access Gmail from other clients using IMAP)  
[Learn more](#)

Status: **IMAP is enabled**

☒ Enable IMAP

☐ Disable IMAP

When I mark a message in IMAP as deleted:

☒ Auto-Expunge on - Immediately update the server. (default)

☐ Auto-Expunge off - Wait for the client to update the server.

When a message is marked as deleted and expunged from the last visible IMAP folder:

☒ Archive the message (default)

☐ Move the message to the Trash

☐ Immediately delete the message forever

Folder size limits

☒ Do not limit the number of messages in an IMAP folder (default)

☐ Limit IMAP folders to contain no more than this many messages

1,000

Configure your email client (e.g. Outlook, Thunderbird, iPhone)

[Configuration instructions](#)


Save Changes

Cancel

### 2- Allow device and Allow less secure app access

Your devices

You're currently signed in to your Google Account on these devices

Linux  
Istanbul, Turkey

☒ This device

Find a lost device

Manage devices

Less secure app access

Your account is vulnerable because you allow apps and devices that use less secure sign-in technology to access your account. To keep your account secure, Google will automatically turn this setting OFF if it's not being used.

On

Turn off access (recommended)

