Uppgift

Thursday, 18 April 2024 10:55

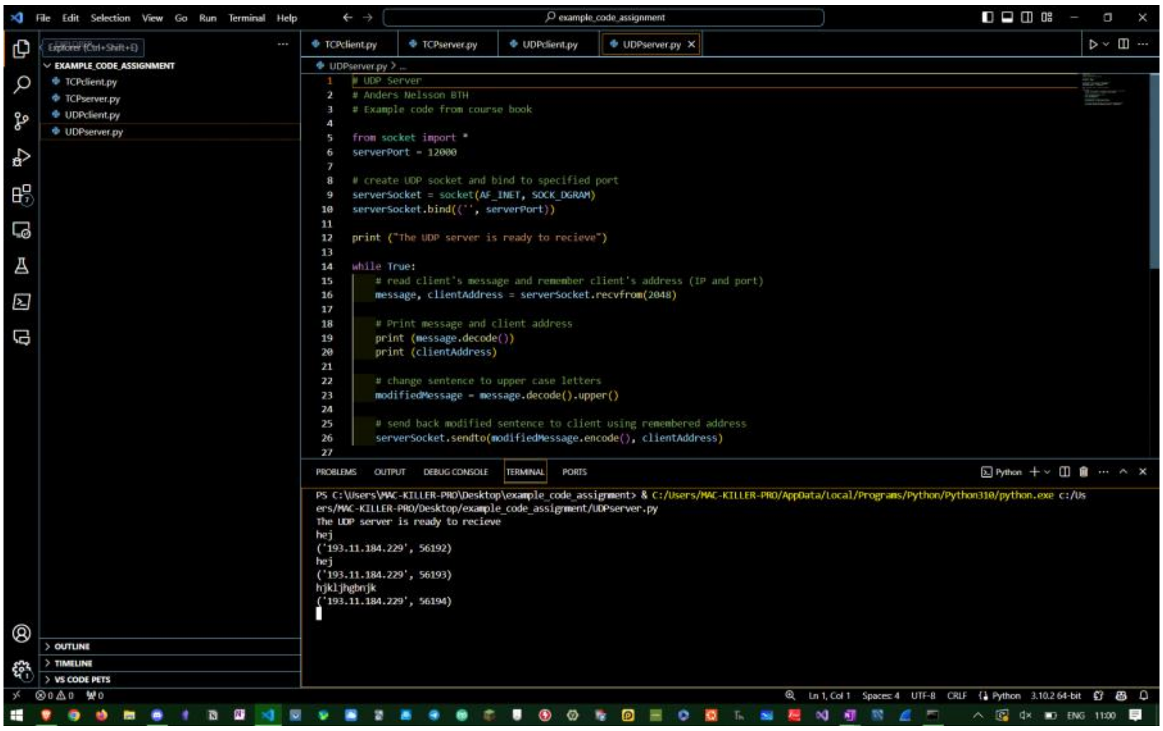
Personal info

Felix Beniamin Cenusa (working with Mathilda Ronnqvist)
Fece23@student.bth.se
My personal number: 200107113953 (no stealing)
Python code from tasks B:

```
import socket
url = "www.ingonline.nu"
path = "/tictactoe/index.php"
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (url, 80)
sock.connect(server_address)
request_message = f'GET {path}?board=xoxoeoeex HTTP/1.1\r\nHost: {url}\r\nConnection:
close\r\n\r\n'
sock.sendall(request_message.encode())
response = sock.recv(4096)
print(response.decode())
sock.close()
```

Packet content of the http request and http response TASK B3b, (i.e., included pdfs from Wireshark) (Find Below)
Summary TASK B4 (Find Below)

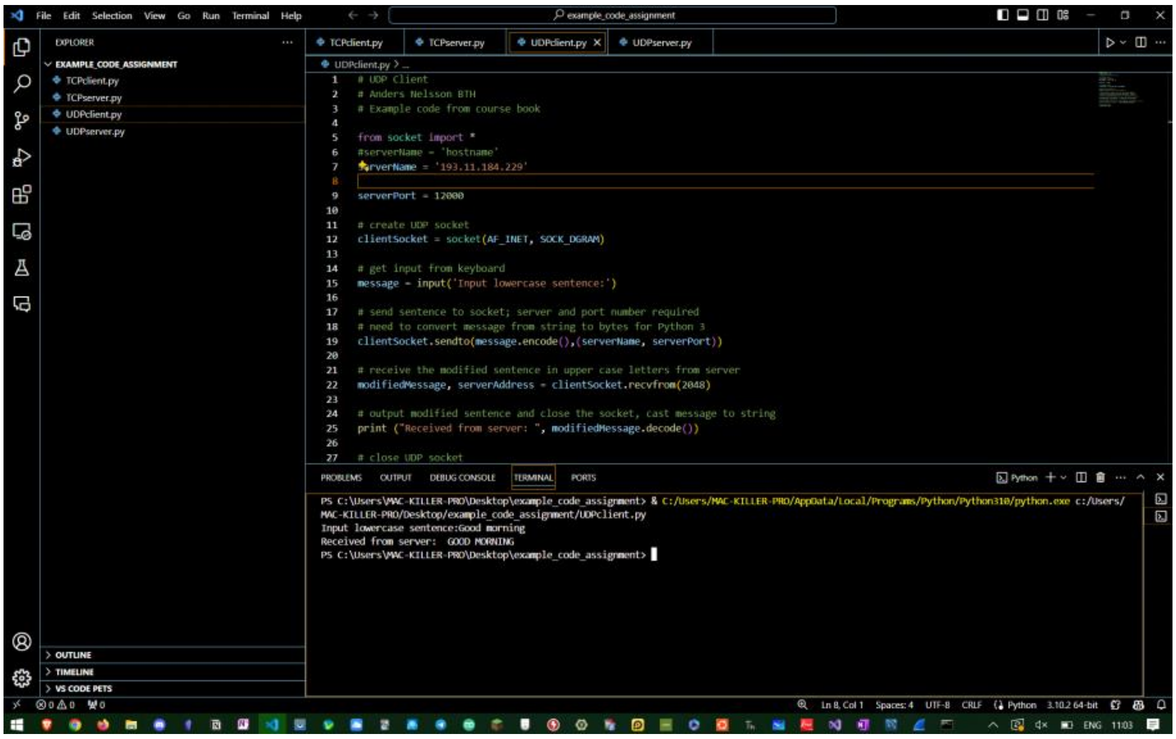
Part A



```
# UDP Server
# Anders Nelsson BTH
# Example code from course book

from socket import *
serverPort = 12000

# create UDP socket and bind to specified port
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

print ("The UDP server is ready to recieve")

while True:
    # read client's message and remember client's address (IP and port)
    message, clientAddress = serverSocket.recvfrom(2048)

    # Print message and client address
    print (message.decode())
    print (clientAddress)

    # change sentence to upper case letters
    modifiedMessage = message.decode().upper()

    # send back modified sentence to client using remembered address
    serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

```
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Users/MAC-KILLER-PRO/AppData/Local/Programs/Python/Python310/python.exe c:/Us
ers/MAC-KILLER-PRO/Desktop/example_code_assignment/UDPserver.py
The UDP server is ready to recieve
hej
('193.11.184.229', 56192)
hej
('193.11.184.229', 56193)
hjkljhgbnjk
('193.11.184.229', 56194)
```

Here me and Mathilda made a UDP connection. I ran the server and she ran the client with my IP adress that I found by doing ipconfig on cmd and taking my IPv4 adress.



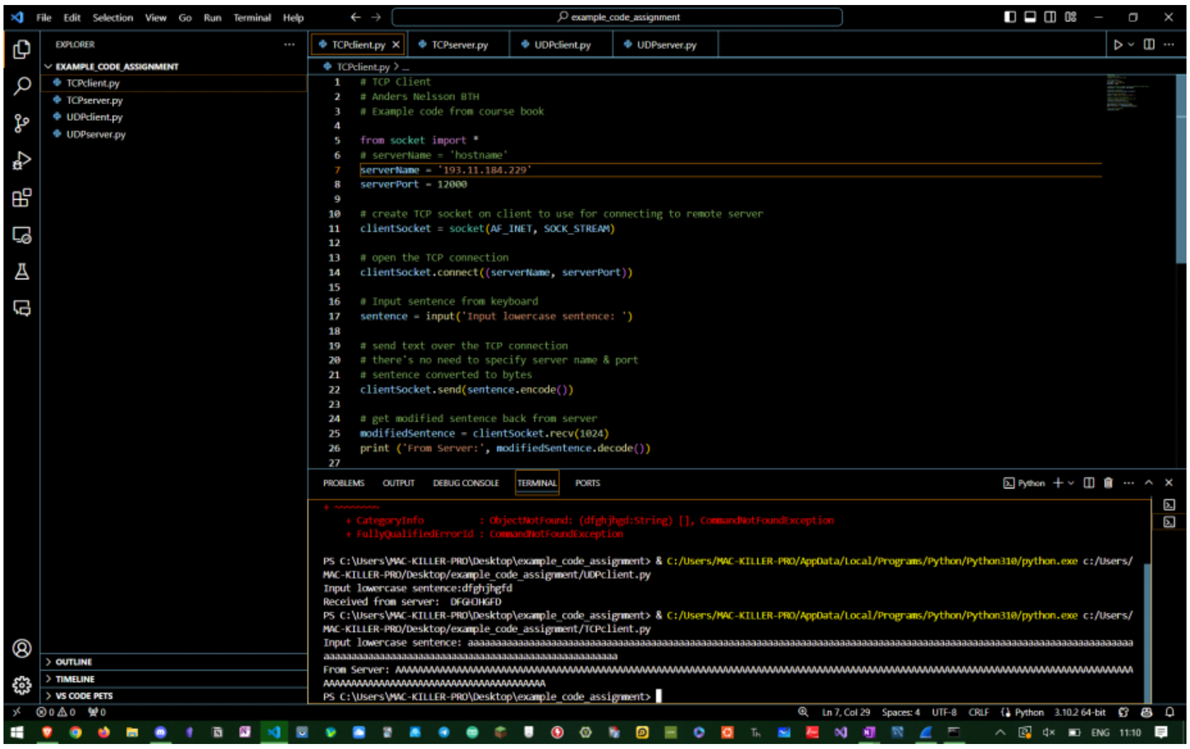
```python
# UDP Client
# Anders Nelsson BTH
# Example code from course book

from socket import *
#serverName = 'hostname'
serverName = '193.11.184.229'

serverPort = 12000

# create UDP socket
clientSocket = socket(AF_INET, SOCK_DGRAM)

# get input from keyboard
message = input('Input lowercase sentence:')

# send sentence to socket; server and port number required
# need to convert message from string to bytes for Python 3
clientSocket.sendto(message.encode(),(serverName, serverPort))

# receive the modified sentence in upper case letters from server
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

# output modified sentence and close the socket, cast message to string
print ("Received from server: ", modifiedMessage.decode())

# close UDP socket
```

```
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Users/MAC-KILLER-PRO/AppData/Local/Programs/Python/Python310/python.exe c:/Users/
MAC-KILLER-PRO/Desktop/example_code_assignment/UDPclient.py
Input lowercase sentence:Good morning
Received from server:  GOOD MORNING
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment>
```

Here we did the revese, I ran the client and Mathilda ran the server. I put the IP adress of Mathildas computer in the ServerName variable and ran the program and the message was received.

Sidenote: The UDP connection required me to aproove a firewall thing in windows, the TCP did not, but maybe it already got the permission.

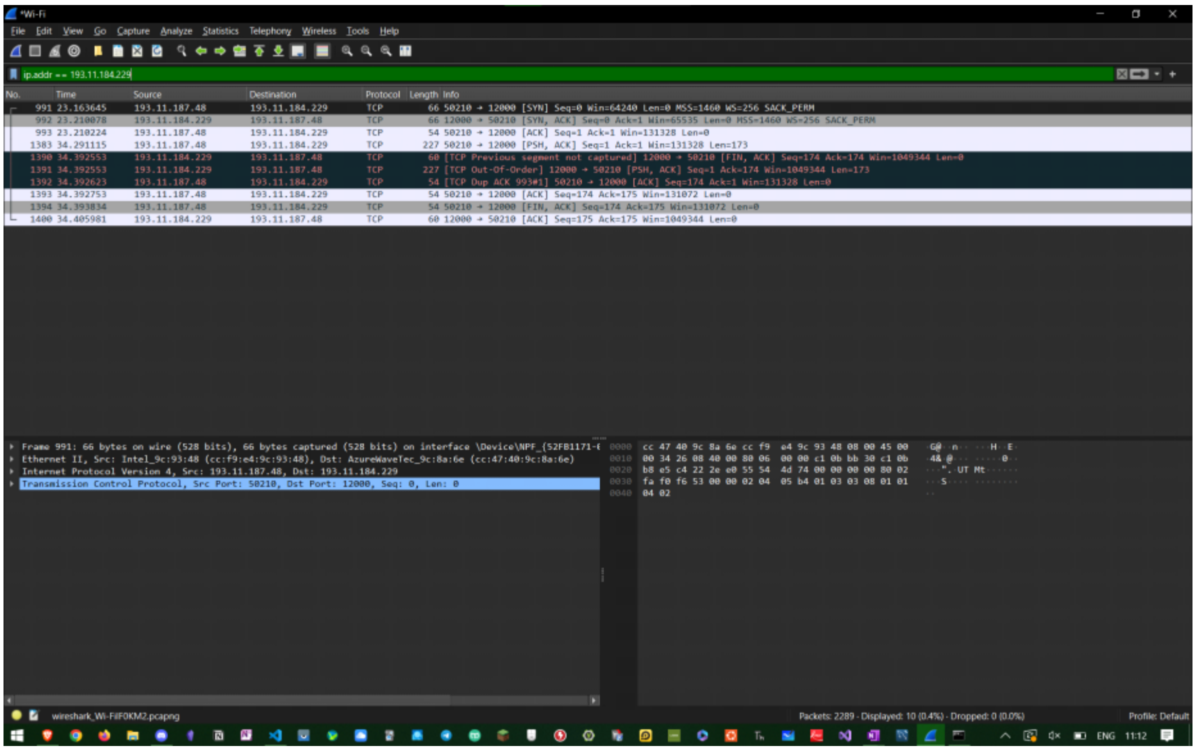
TCP conection



```python
# TCP Client
# Anders Nelsson BTH
# Example code from course book

from socket import *
# serverName = 'hostname'
serverName = '193.11.184.229'
serverPort = 12000

# create TCP socket on client to use for connecting to remote server
clientSocket = socket(AF_INET, SOCK_STREAM)

# open the TCP connection
clientSocket.connect((serverName, serverPort))

# Input sentence from keyboard
sentence = input('Input lowercase sentence: ')

# send text over the TCP connection
# there's no need to specify server name & port
# sentence converted to bytes
clientSocket.send(sentence.encode())

# get modified sentence back from server
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
```

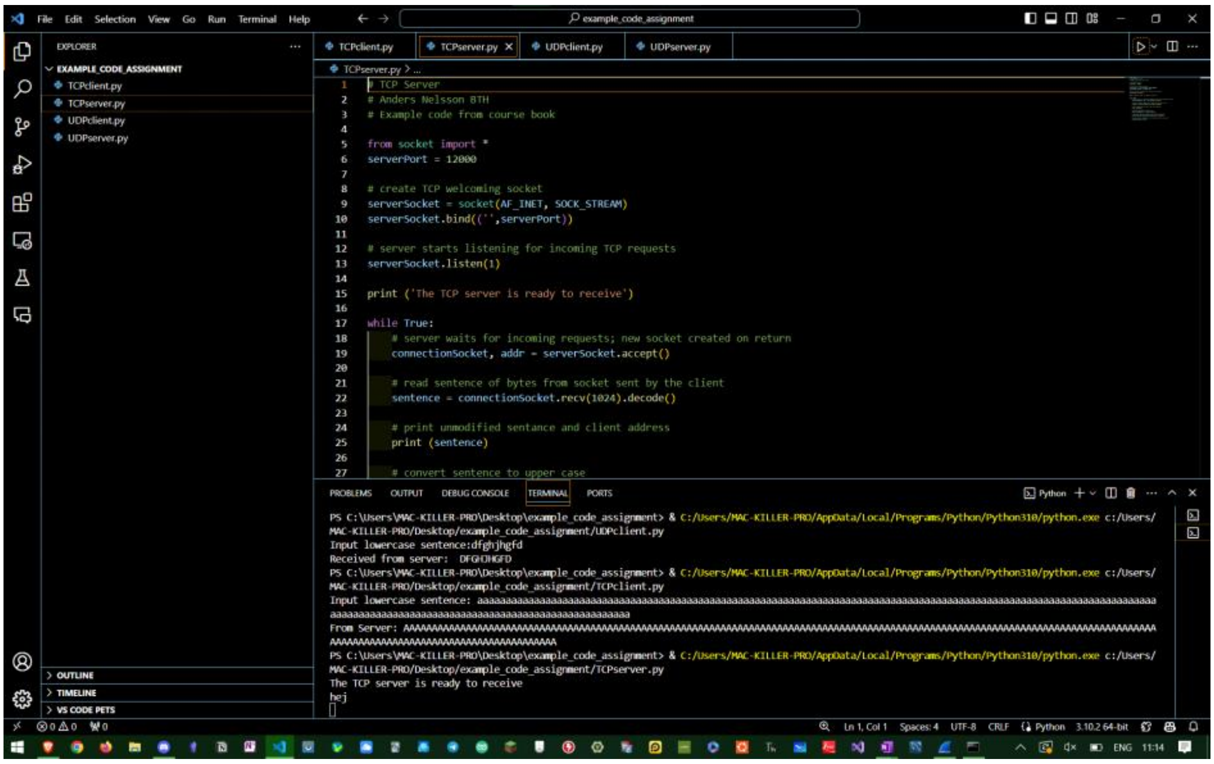
```
+ ~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (dfghjhgd:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Users/MAC-KILLER-PRO/AppData/Local/Programs/Python/Python310/python.exe c:/Users/
MAC-KILLER-PRO/Desktop/example_code_assignment/UDPclient.py
Input lowercase sentence:dfghjhgfd
Received from server:  DFGHJHGFD
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Users/MAC-KILLER-PRO/AppData/Local/Programs/Python/Python310/python.exe c:/Users/
MAC-KILLER-PRO/Desktop/example_code_assignment/TCPclient.py
Input lowercase sentence: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
From Server: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment>
```

Here I ran the TCP client and Mathilda ran the TCP server, I sent a long string of "a"'s to see if any errors would arrise, but I suppose the message must be much longer If you WANT to get a error in transmission.



```
ip.addr == 193.11.184.229

No.  Time       Source          Destination     Protocol Length Info
 991 23.163645  193.11.187.48   193.11.184.229  TCP       66 50210 → 12000 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
 992 23.210078  193.11.184.229  193.11.187.48   TCP       66 12000 → 50210 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
 993 23.210224  193.11.187.48   193.11.184.229  TCP       54 50210 → 12000 [ACK] Seq=1 Ack=1 Win=131328 Len=0
1383 34.291115  193.11.187.48   193.11.184.229  TCP      227 50210 → 12000 [PSH, ACK] Seq=1 Ack=1 Win=131328 Len=173
1390 34.392553  193.11.184.229  193.11.187.48   TCP       60 [TCP Previous segment not captured] 12000 → 50210 [FIN, ACK] Seq=174 Ack=174 Win=1049344 Len=0
1391 34.392553  193.11.184.229  193.11.187.48   TCP      227 [TCP Out-Of-Order] 12000 → 50210 [PSH, ACK] Seq=1 Ack=174 Win=1049344 Len=173
1392 34.392623  193.11.187.48   193.11.184.229  TCP       54 [TCP Dup ACK 993#1] 50210 → 12000 [ACK] Seq=174 Ack=1 Win=131328 Len=0
1393 34.392753  193.11.187.48   193.11.184.229  TCP       54 50210 → 12000 [ACK] Seq=174 Ack=175 Win=131072 Len=0
1394 34.393834  193.11.187.48   193.11.184.229  TCP       54 50210 → 12000 [FIN, ACK] Seq=174 Ack=175 Win=131072 Len=0
1400 34.405981  193.11.184.229  193.11.187.48   TCP       60 12000 → 50210 [ACK] Seq=175 Ack=175 Win=1049344 Len=0

> Frame 991: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{52FB1171-
> Ethernet II, Src: Intel_9c:93:48 (cc:f9:e4:9c:93:48), Dst: AzureWaveTec_9c:8a:6e (cc:47:40:9c:8a:6e)
> Internet Protocol Version 4, Src: 193.11.187.48, Dst: 193.11.184.229
> Transmission Control Protocol, Src Port: 50210, Dst Port: 12000, Seq: 0, Len: 0

0000  cc 47 40 9c 8a 6e cc f9  e4 9c 93 48 08 00 45 00   ·G@··n·· ···H··E·
0010  00 34 26 08 40 00 80 06  00 00 c1 0b bb 30 c1 0b   ·4&·@··· ·····0··
0020  b8 e5 c4 22 2e e0 55 54  4d 74 00 00 00 00 80 02   ···".·UT Mt······
0030  fa f0 f6 53 00 00 02 04  05 b4 01 03 03 08 01 01   ···S···· ········
0040  04 02                                              ··

wireshark_Wi-FiIF0KM2.pcapng    Packets: 2289 · Displayed: 10 (0.4%) · Dropped: 0 (0.0%)    Profile: Default
```

I didn’t take a screenshot for the UDP protocol, but it had only 2 packets or lines visible, but it seems that TCP sends a lot more data. (this screenshot is of me running the TCP client and seeing what happened in wireshark) This is probably because TCP has to establish a connection first like a handshake or a hello and after it will send the actual data, so more stuff is sent then strictly necessary but it makes it more reliable then UDP.



```python
# TCP Server
# Anders Nelsson BTH
# Example code from course book

from socket import *
serverPort = 12000

# create TCP welcoming socket
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('',serverPort))

# server starts listening for incoming TCP requests
serverSocket.listen(1)

print ('The TCP server is ready to receive')

while True:
    # server waits for incoming requests; new socket created on return
    connectionSocket, addr = serverSocket.accept()

    # read sentence of bytes from socket sent by the client
    sentence = connectionSocket.recv(1024).decode()

    # print unmodified sentance and client address
    print (sentence)

    # convert sentence to upper case
```

```
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Users/MAC-KILLER-PRO/AppData/Local/Programs/Python/Python310/python.exe c:/Users/
MAC-KILLER-PRO/Desktop/example_code_assignment/UDPclient.py
Input lowercase sentence:dfghjhgfd
Received from server:  DFGHJHGFD
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Users/MAC-KILLER-PRO/AppData/Local/Programs/Python/Python310/python.exe c:/Users/
MAC-KILLER-PRO/Desktop/example_code_assignment/TCPclient.py
Input lowercase sentence: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
From Server: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Users/MAC-KILLER-PRO/AppData/Local/Programs/Python/Python310/python.exe c:/Users/
MAC-KILLER-PRO/Desktop/example_code_assignment/TCPserver.py
The TCP server is ready to receive
hej
```

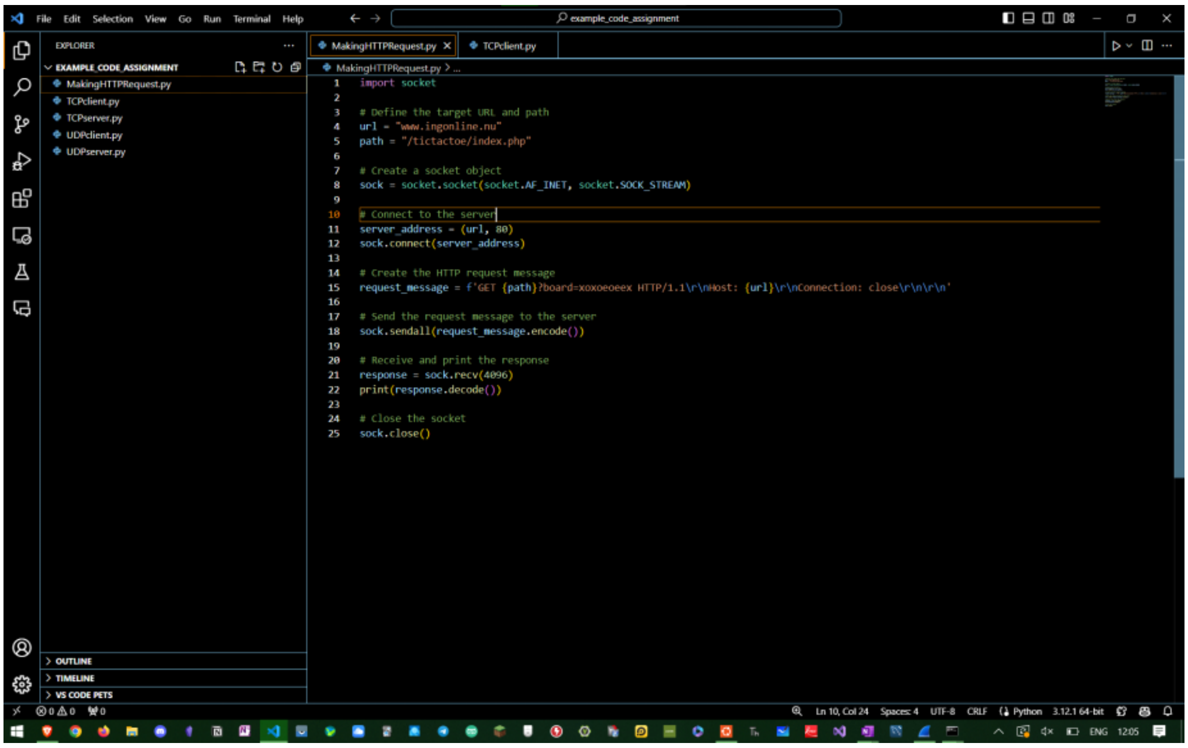
Here we did the reverse again, I ran TCP server and Mathilda ran TCP client. The message sends instantly its eerie to watch :)

A4 Brief summary

Getting the sample files for the connections was simple, running them with python as a breeze but seeing the connection being made so easily was till amazing to see for me. Feels a bit like how the first person to ever sent a internet message probably felt like :))
The code behind the program is surprisingly easy, I am surprised I just need the recievers adress and for them to also run a program.
Overall I was surprised how simple a connection can be nade with UDP or TCP and I will be using this newfound knowledge in the future to perhaps play pranks on my friends or to make more complex applications :)

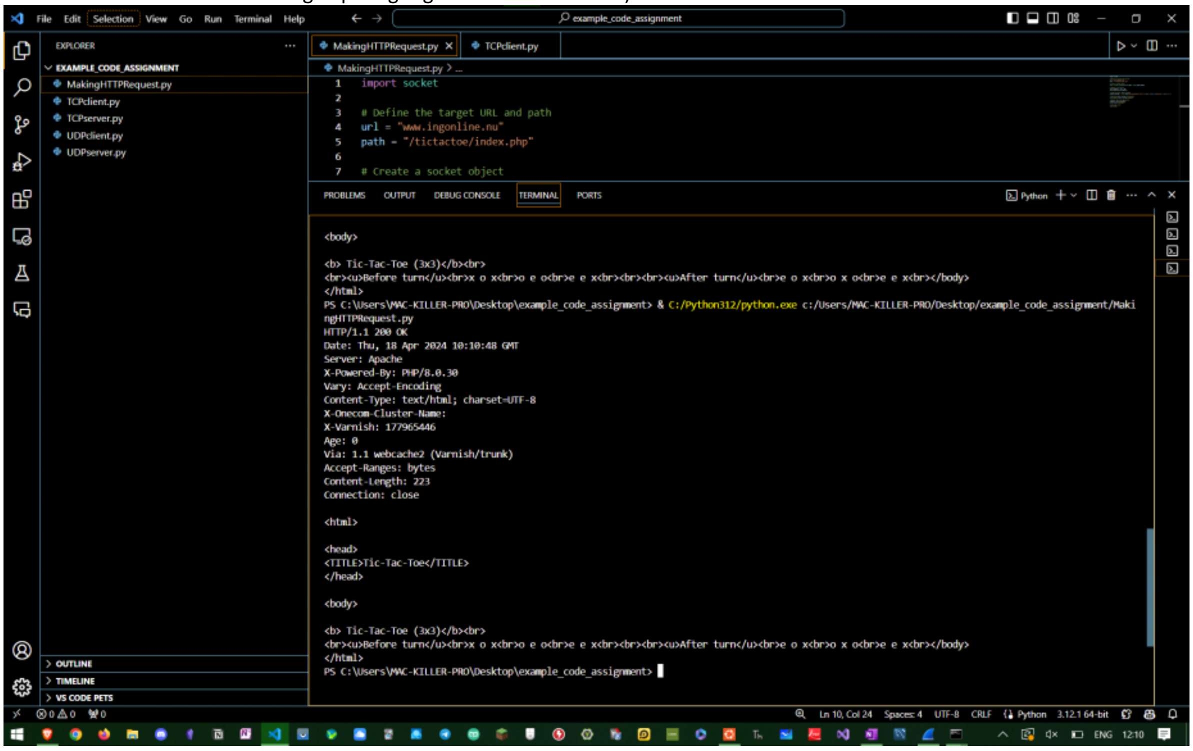
Part B

Making a HTTP request



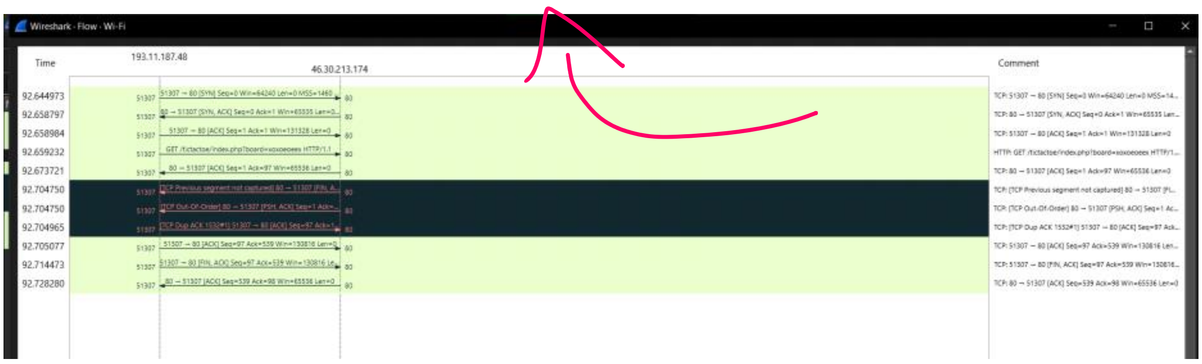
```python
import socket

# Define the target URL and path
url = "www.ingonline.nu"
path = "/tictactoe/index.php"

# Create a socket object
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to the server
server_address = (url, 80)
sock.connect(server_address)

# Create the HTTP request message
request_message = f'GET {path}?board=xoxoeoeex HTTP/1.1\r\nHost: {url}\r\nConnection: close\r\n\r\n'

# Send the request message to the server
sock.sendall(request_message.encode())

# Receive and print the response
response = sock.recv(4096)
print(response.decode())

# Close the socket
sock.close()
```

Python program was written, the connection as made, it was a bit confusing at first, didn't know that we should send the info in the url after ?board= but nothing a quick google search cant solve :)

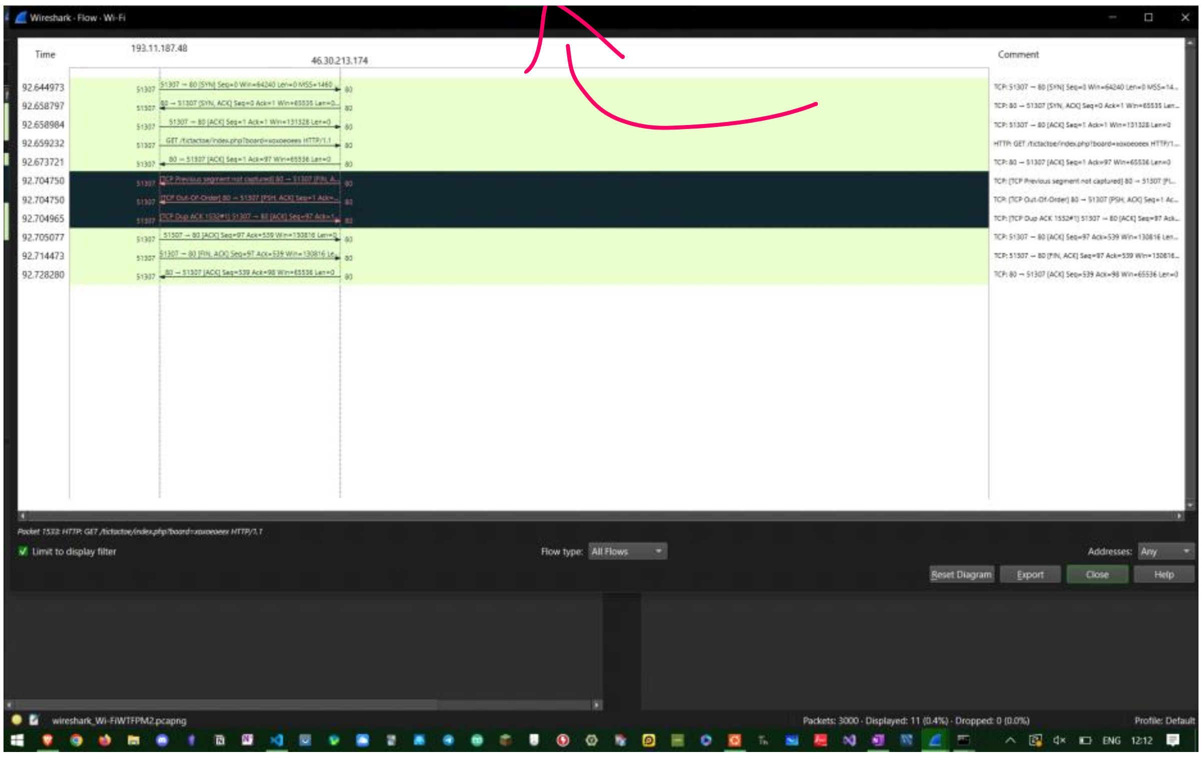


```
<body>

<b> Tic-Tac-Toe (3x3)</b><br>
<br><u>Before turn</u><br>x o x<br>o e o<br>e e x<br><br><br><u>After turn</u><br>e o x<br>o x o<br>e e x<br></body>
</html>
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment> & C:/Python312/python.exe c:/Users/MAC-KILLER-PRO/Desktop/example_code_assignment/Maki
ngHTTPRequest.py
HTTP/1.1 200 OK
Date: Thu, 18 Apr 2024 10:10:48 GMT
Server: Apache
X-Powered-By: PHP/8.0.30
Vary: Accept-Encoding
Content-Type: text/html; charset=UTF-8
X-Onecom-Cluster-Name:
X-Varnish: 177965446
Age: 0
Via: 1.1 webcache2 (Varnish/trunk)
Accept-Ranges: bytes
Content-Length: 223
Connection: close

<html>

<head>
<TITLE>Tic-Tac-Toe</TITLE>
</head>

<body>

<b> Tic-Tac-Toe (3x3)</b><br>
<br><u>Before turn</u><br>x o x<br>o e o<br>e e x<br><br><br><u>After turn</u><br>e o x<br>o x o<br>e e x<br></body>
</html>
PS C:\Users\MAC-KILLER-PRO\Desktop\example_code_assignment>
```

Screenshot of the response.

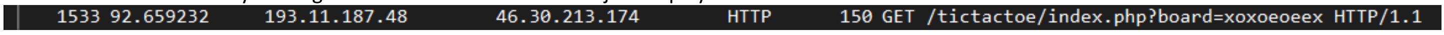


```
Wireshark · Flow · Wi-Fi

Time        193.11.187.48      46.30.213.174      Comment
92.644973   TCP: 51307 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=14…
92.658797   TCP: 80 → 51307 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len…
92.658984   TCP: 51307 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
92.659232   HTTP: GET /tictactoe/index.php?board=xoxoeoeex HTTP/1…
92.673721   TCP: 80 → 51307 [ACK] Seq=1 Ack=97 Win=65536 Len=0
92.704750   TCP: [TCP Previous segment not captured] 80 → 51307 [FI…
92.704750   TCP: [TCP Out-Of-Order] 80 → 51307 [PSH, ACK] Seq=1 Ac…
92.704965   TCP: [TCP Dup ACK 1552#1] 51307 → 80 [ACK] Seq=97 Ack…
92.705077   TCP: 51307 → 80 [ACK] Seq=97 Ack=539 Win=130816 Len…
92.714473   TCP: 51307 → 80 [FIN, ACK] Seq=97 Ack=539 Win=130816…
92.728280   TCP: 80 → 51307 [ACK] Seq=539 Ack=98 Win=65536 Len=0
```

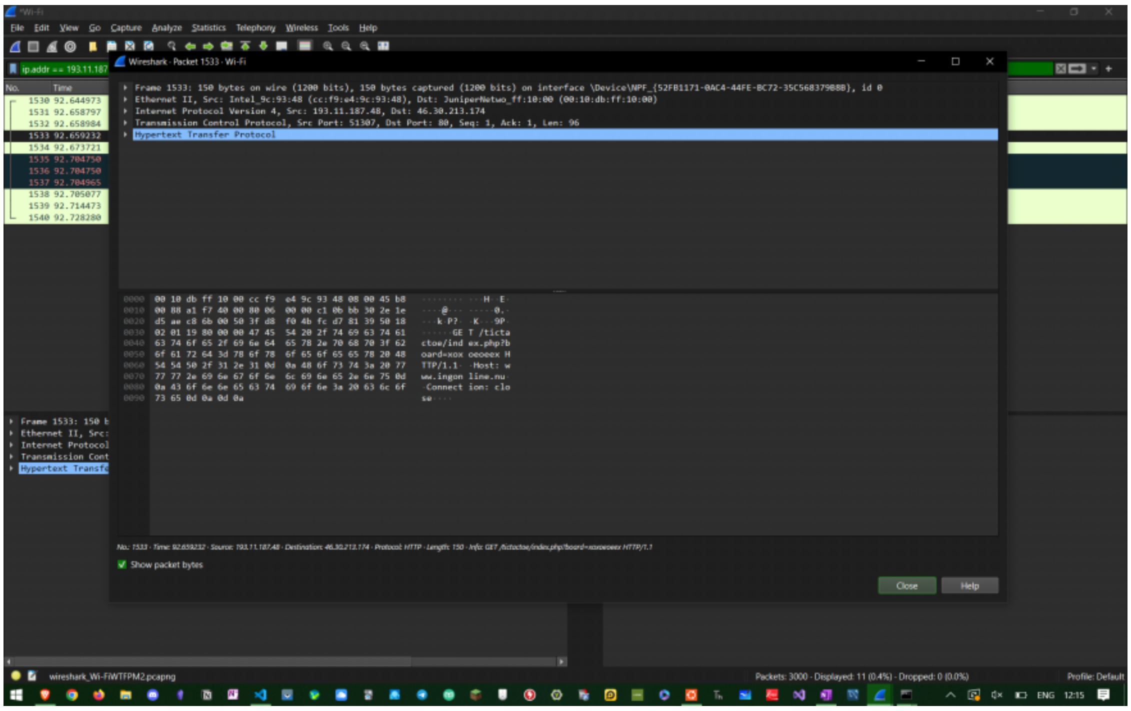


```
Wireshark · Flow · Wi-Fi

Time        193.11.187.48                                                   46.30.213.174     Comment
92.644973   51307  51307 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460          80   TCP: 51307 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=14…
92.658797   51307  80 → 51307 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0…       80   TCP: 80 → 51307 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len…
92.658984   51307  51307 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0            80   TCP: 51307 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
92.659232   51307  GET /tictactoe/index.php?board=xoxoeoeex HTTP/1.1        80   HTTP: GET /tictactoe/index.php?board=xoxoeoeex HTTP/1…
92.673721   51307  80 → 51307 [ACK] Seq=1 Ack=97 Win=65536 Len=0            80   TCP: 80 → 51307 [ACK] Seq=1 Ack=97 Win=65536 Len=0
92.704750   51307  [TCP Previous segment not captured] 80 → 51307 [FIN, A…  80   TCP: [TCP Previous segment not captured] 80 → 51307 [FI…
92.704750   51307  [TCP Out-Of-Order] 80 → 51307 [PSH, ACK] Seq=1 Ack=…     80   TCP: [TCP Out-Of-Order] 80 → 51307 [PSH, ACK] Seq=1 Ac…
92.704965   51307  [TCP Dup ACK 1532#1] 51307 → 80 [ACK] Seq=97 Ack=1…      80   TCP: [TCP Dup ACK 1532#1] 51307 → 80 [ACK] Seq=97 Ack…
92.705077   51307  51307 → 80 [ACK] Seq=97 Ack=539 Win=130816 Len=0         80   TCP: 51307 → 80 [ACK] Seq=97 Ack=539 Win=130816 Len…
92.714473   51307  51307 → 80 [FIN, ACK] Seq=97 Ack=539 Win=130816 Le…      80   TCP: 51307 → 80 [FIN, ACK] Seq=97 Ack=539 Win=130816…
92.728280   51307  80 → 51307 [ACK] Seq=539 Ack=98 Win=65536 Len=0          80   TCP: 80 → 51307 [ACK] Seq=539 Ack=98 Win=65536 Len=0

Packet 1532: HTTP: GET /tictactoe/index.php?board=xoxoeoeex HTTP/1.1
[✓] Limit to display filter          Flow type: All Flows          Addresses: Any

wireshark_Wi-FiWTFPM2.pcapng          Packets: 3000 · Displayed: 11 (0.4%) · Dropped: 0 (0.0%)          Profile: Default
```

Screenshot of the flow graph. Took a long time to figure out I can press "Limit to display filter" but with it enabled its easy to seew what happened when I ran the program. I filtered for anything that had my IP and the IP of the website that was given to us. I found the IPv4 IP adress of the website by looking at the URL in wireshark and it just displays the IP.

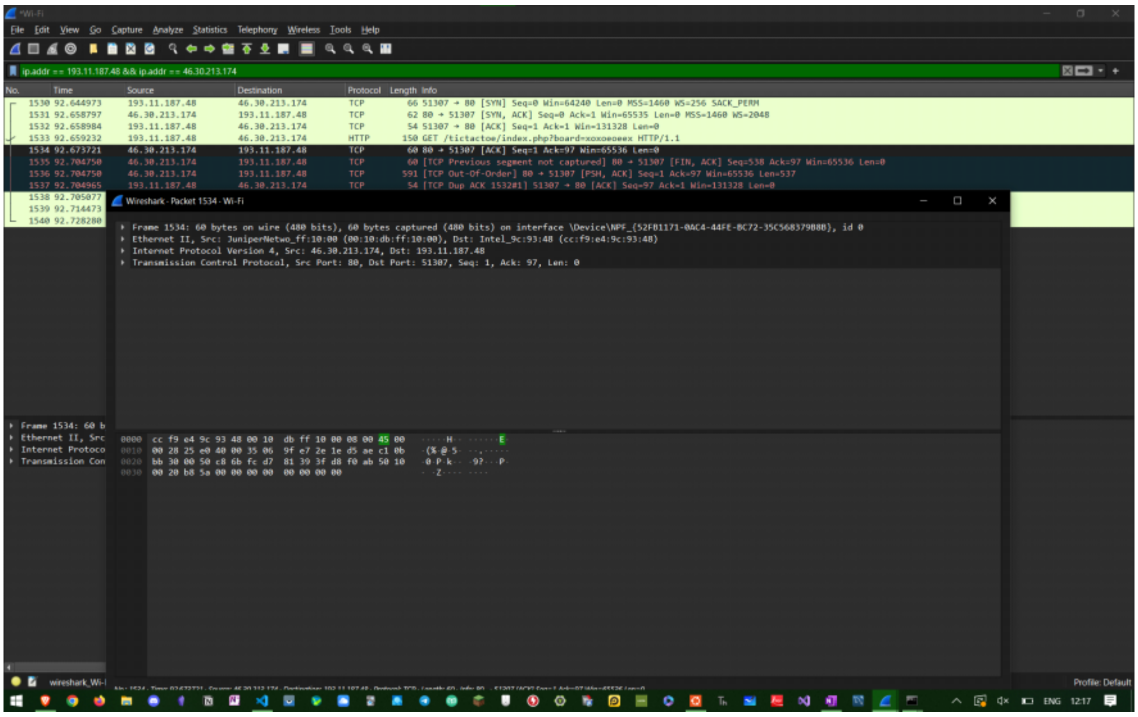


```
1533 92.659232    193.11.187.48       46.30.213.174       HTTP      150 GET /tictactoe/index.php?board=xoxoeoeex HTTP/1.1
```



```
ip.addr == 193.11.187

No.  Time
1530 92.644973
1531 92.658797
1532 92.658984
1533 92.659232
1534 92.673721
1535 92.704750
1536 92.704750
1537 92.704965
1538 92.705077
1539 92.714473
1540 92.728280

Wireshark · Packet 1533 · Wi-Fi

▸ Frame 1533: 150 bytes on wire (1200 bits), 150 bytes captured (1200 bits) on interface \Device\NPF_{52FB1171-0AC4-44FE-BC72-35C568379B8B}, id 0
▸ Ethernet II, Src: Intel_9c:93:48 (cc:f9:e4:9c:93:48), Dst: JuniperNetwo_ff:10:00 (00:10:db:ff:10:00)
▸ Internet Protocol Version 4, Src: 193.11.187.48, Dst: 46.30.213.174
▸ Transmission Control Protocol, Src Port: 51307, Dst Port: 80, Seq: 1, Ack: 1, Len: 96
▸ Hypertext Transfer Protocol

0000  00 10 db ff 10 00 cc f9  e4 9c 93 48 08 00 45 b8   ·········H··E·
0010  00 88 a1 f7 40 00 80 06  00 00 c1 0b bb 30 2e 1e   ····@·······0.·
0020  d5 ae c8 6b 00 50 3f d8  f0 4b fc d7 81 39 50 18   ···k·P?··K···9P·
0030  02 01 19 80 00 00 47 45  54 20 2f 74 69 63 74 61   ······GE T /ticta
0040  63 74 6f 65 2f 69 6e 64  65 78 2e 70 68 70 3f 62   ctoe/ind ex.php?b
0050  6f 61 72 64 3d 78 6f 78  6f 65 6f 65 65 78 20 48   oard=xox oeoeex H
0060  54 54 50 2f 31 2e 31 0d  0a 48 6f 73 74 3a 20 77   TTP/1.1· ·Host: w
0070  77 77 2e 69 6e 67 6f 6e  6c 69 6e 65 2e 6e 75 0d   ww.ingon line.nu·
0080  0a 43 6f 6e 6e 65 63 74  69 6f 6e 3a 20 63 6c 6f   ·Connect ion: clo
0090  73 65 0d 0a 0d 0a                                  se····

No.: 1533 · Time: 92.659232 · Source: 193.11.187.48 · Destination: 46.30.213.174 · Protocol: HTTP · Length: 150 · Info: GET /tictactoe/index.php?board=xoxoeoeex HTTP/1.1
[✓] Show packet bytes

wireshark_Wi-FiWTFPM2.pcapng          Packets: 3000 · Displayed: 11 (0.4%) · Dropped: 0 (0.0%)          Profile: Default
```

Content of the packet from my computer containing the HTTP request



```
ip.addr == 193.11.187.48 && ip.addr == 46.30.213.174

No.   Time       Source          Destination     Protocol Length Info
1530 92.644973  193.11.187.48   46.30.213.174   TCP      66 51307 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
1531 92.658797  46.30.213.174   193.11.187.48   TCP      62 80 → 51307 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=2048
1532 92.658984  193.11.187.48   46.30.213.174   TCP      54 51307 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0
1533 92.659232  193.11.187.48   46.30.213.174   HTTP    150 GET /tictactoe/index.php?board=xoxoeoeex HTTP/1.1
1534 92.673721  46.30.213.174   193.11.187.48   TCP      60 80 → 51307 [ACK] Seq=1 Ack=97 Win=65536 Len=0
1535 92.704750  46.30.213.174   193.11.187.48   TCP      60 [TCP Previous segment not captured] 80 → 51307 [FIN, ACK] Seq=538 Ack=97 Win=65536 Len=0
1536 92.704750  46.30.213.174   193.11.187.48   TCP     591 [TCP Out-Of-Order] 80 → 51307 [PSH, ACK] Seq=1 Ack=97 Win=65536 Len=537
1537 92.704965  193.11.187.48   46.30.213.174   TCP      54 [TCP Dup ACK 1532#1] 51307 → 80 [ACK] Seq=97 Ack=1 Win=131328 Len=0
1538 92.705077
1539 92.714473
1540 92.728280

Wireshark · Packet 1534 · Wi-Fi
› Frame 1534: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF_{52FB1171-0AC4-44FE-8C72-35C568379888}, id 0
› Ethernet II, Src: JuniperNetwo_ff:10:00 (00:10:db:ff:10:00), Dst: Intel_9c:93:48 (cc:f9:e4:9c:93:48)
› Internet Protocol Version 4, Src: 46.30.213.174, Dst: 193.11.187.48
› Transmission Control Protocol, Src Port: 80, Dst Port: 51307, Seq: 1, Ack: 97, Len: 0

0000  cc f9 e4 9c 93 48 00 10  db ff 10 00 08 00 45 00   ·····H·· ······E·
0010  00 28 25 e0 40 00 35 06  9f e7 2e 1e d5 ae c1 0b   ·(%·@·5· ··.·····
0020  bb 30 00 50 c8 6b fc d7  81 39 3f d8 f0 ab 50 10   ·0·P·k·· ·9?···P·
0030  00 20 b8 5a 00 00 00 00  00 00 00 00               · ·Z···· ····
```

- The contents of the packet from the server containing the HTTP response.

Part B conclusion

To conclude, it was nice to make a HTTP connection and reqlized that its easier then I thought.
Wireshark was also a very useful tool to this project.
Overall, while there were some difficulties along the way, the process of creating the web browser was ultimately successful, and I gained valuable experience in web development through this project.