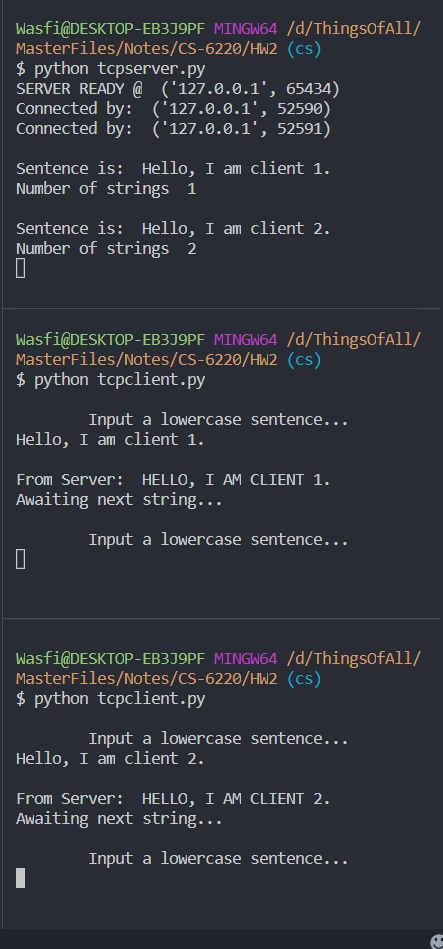
Wasfi Momen

9/22/18

1. TCP Server (Multithreaded)

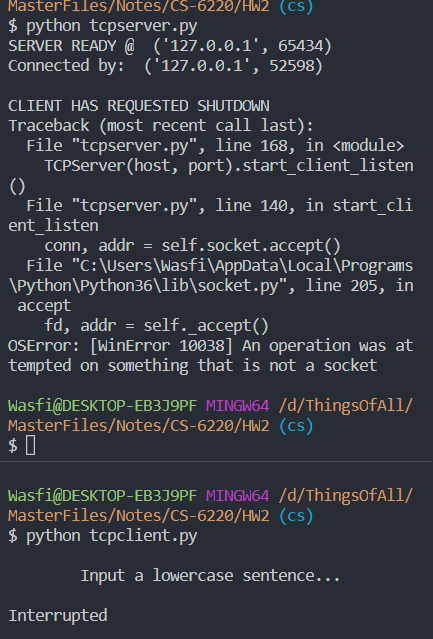
**Multithreaded Feature**

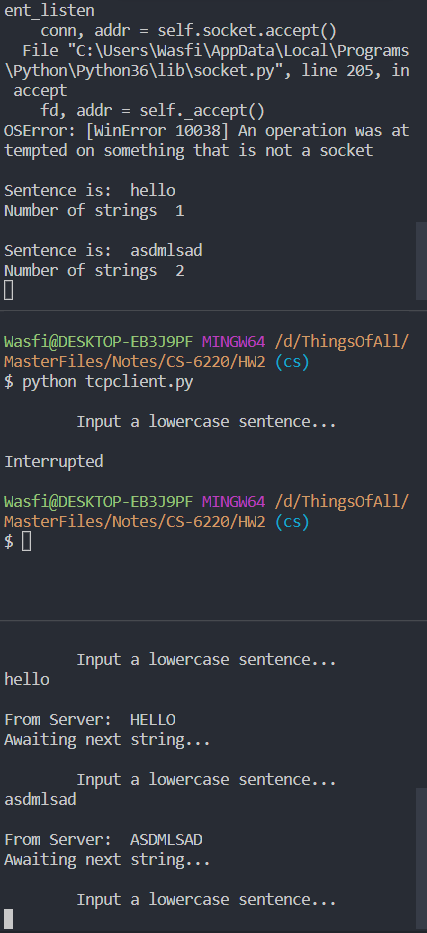


**Client shutdown feature**

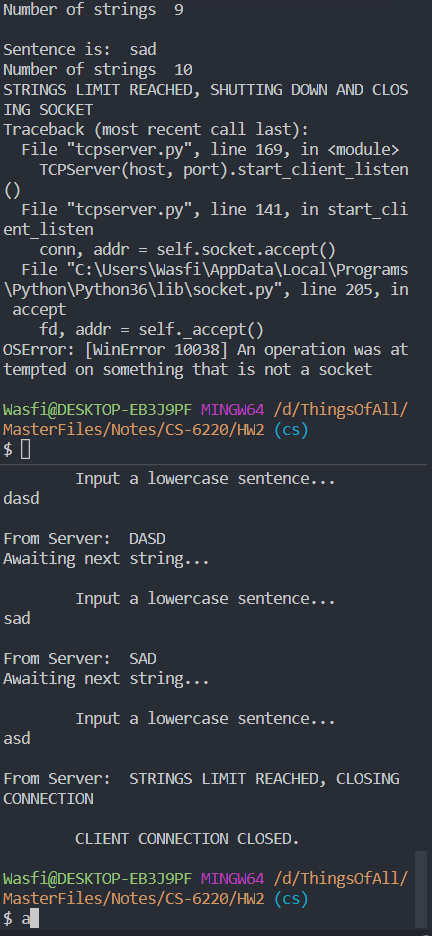
(Note the CLIENT HAS REQUESTED SHUTDOWN message. Server will continue to

run if more than two clients are connected. Exception is reported but does not halt execution.)

**1 client scenario 2 client scenario** (server continues to function)

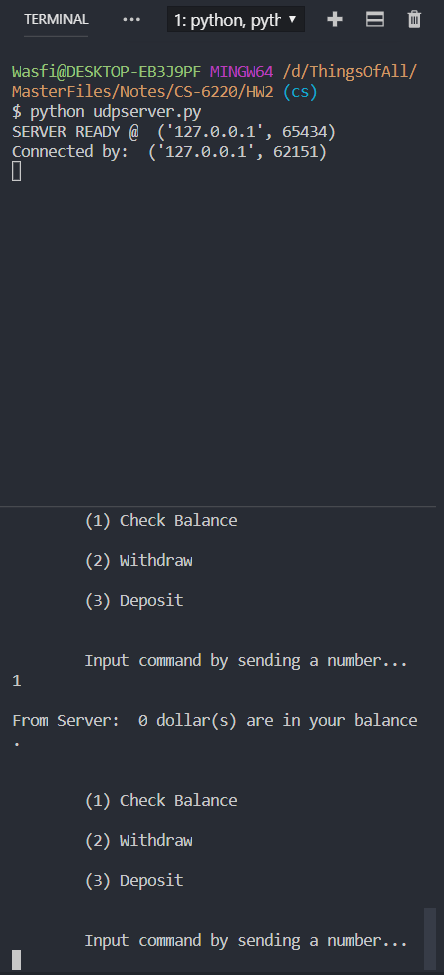


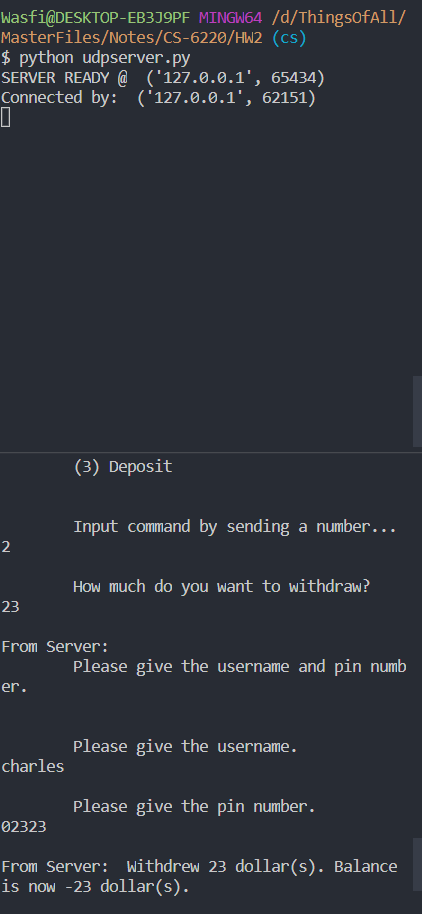
**Strings Limit Reached Feature**



1. UDP Server

**Check Balance Feature**

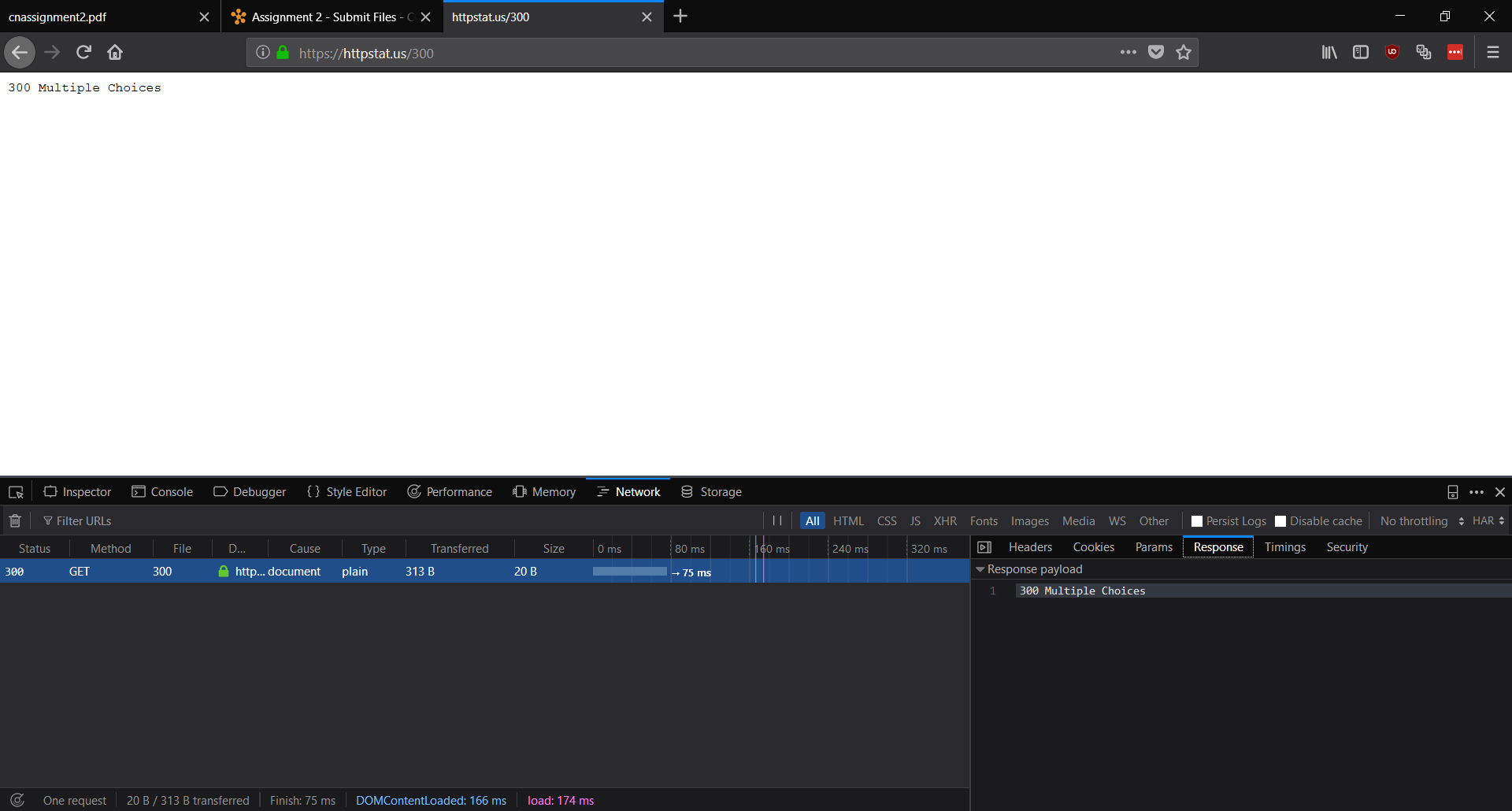


**Withdraw Feature**

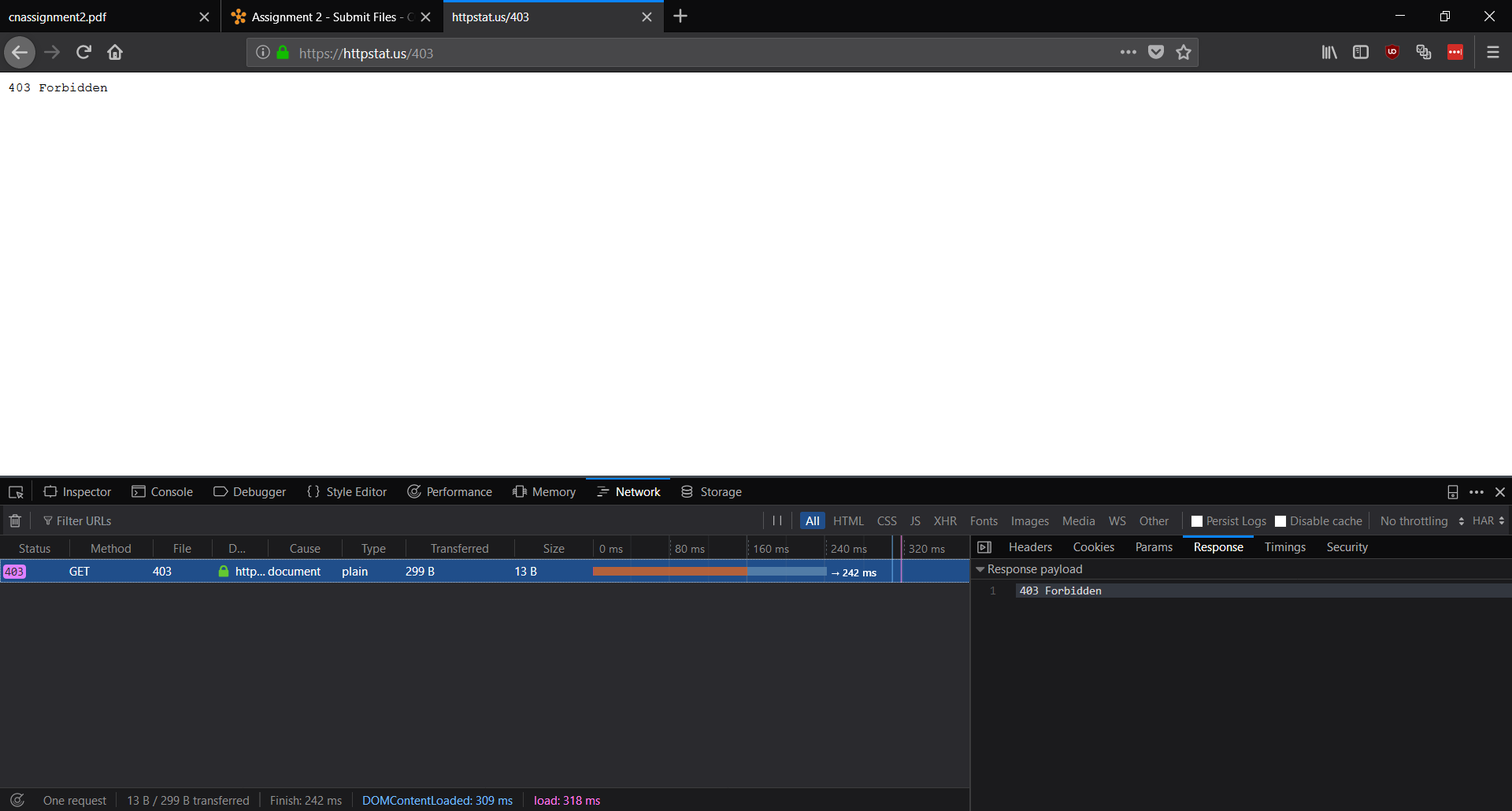
1. HTTP Responses

Went to httpstat.us and tried a couple of responses.

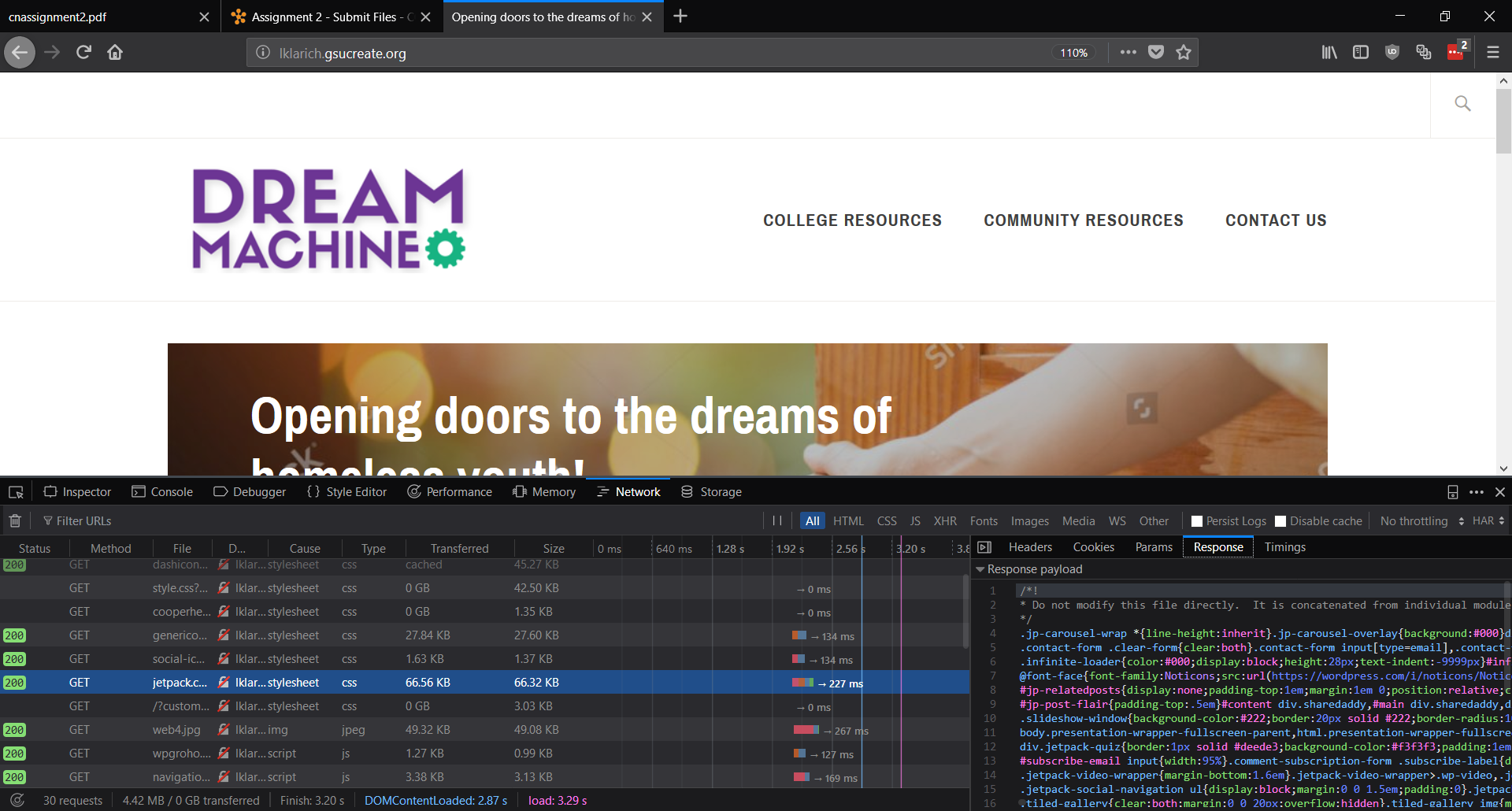
**300 multiple choices**



**403 Forbidden**



Also tried a GSU domain lklarich.gsucreate.org, but all it returned were 200s.



CODE

"""

udpserver.py

A udp server made in python.

Some variable and function names from socketserver.py in python 3.7 release.

Made by Wasfi Momen.

"""

"""

REQUIRMENTS

- ~~Maintains file with: name, pin, and balance~~

- ~~Prompts client for auth using name and pin~~

- ~~Allows client to deposit and withdraw AFTER auth~~

- ~~Messages client confirmation of record update~~

- ~~Allows client to ask for and receive current balance~~

ALL COMPLETE

"""

*import* socket

*import* sys

class UDPServer:

address\_family = socket.AF\_INET *# only IPv4 connections*

socket\_type = socket.SOCK\_DGRAM *# constant to use UDP socket type*

*# to be put into a file*

balance = 0

username = "charles"

pin = "02323"

client\_details = [] *# list to hold client tuples*

*# constructor from the socket.py module*

def \_\_init\_\_(*self*, *server\_address*, *server\_port*):

self.server\_address = server\_address

self.server\_port = server\_port

self.socket = socket.socket(self.address\_family, self.socket\_type)

*try*:

self.write\_to\_file() *# initialize and write to file*

self.bind()

*except*:

self.socket.close()

*raise* Exception('Error in setting up connection to host port.')

def add\_client\_details(*self*, *addr*):

"""

Add the tuples of our connected clients to a list.

This is used namely to connect back to clients for an auth check.

See auth\_challenge() for more info.

"""

*if* addr in self.client\_details:

*return*

self.client\_details.append(addr)

print("Connected by: ", addr)

def bind(*self*):

"""Binds the socket to any available port"""

*try*:

*# a tuple of the address and port is used for the bind function*

self.socket.bind((self.server\_address, self.server\_port))

*except*:

self.socket.close()

*raise* Exception(

'Error in binding socket to address and port specified.')

def check\_auth(*self*, *username*, *pin*):

*# forgot list comprehension, so we separate the first chars here*

username = username[1:]

pin = pin[1:]

*if* (username == self.username and pin == self.pin):

*return* True

*else*:

*return* False

def auth\_challege(*self*):

"""

Challenges the client to provide credentials.

Credentials are returned as a tuple that we can pass to

check\_auth individually.

Returns true if credentials are correct and false otherwise.

"""

message = "\n\tPlease give the username and pin number.\n"

*# send a message back to the client asking for auth info*

self.socket.sendto(message.encode(), (self.client\_details[0]))

*# initate a new frame to get UDP data. (Probably can just do self.socket.recvfrom(2048))*

conn, addr = self.socket.recvfrom(2048)

credentials = conn.decode()

credentials = credentials.split()

credentials = [

word *for* line in credentials *for* word in line.split()] *# separate the username and password strings*

*if* (self.check\_auth(credentials[0], credentials[1]) == True):

*return* True

*else*:

*return* False

def check\_balance(*self*):

*return* str(self.balance) + " dollar(s) are in your balance."

def deposit(*self*, *amount*):

amount = amount[0:]

self.balance = self.balance + int(amount)

self.write\_to\_file()

def withdraw(*self*, *amount*):

amount = amount[0:]

self.balance = self.balance - int(amount)

self.write\_to\_file()

def process\_command(*self*, *message*):

"""

Takes in a message of 3 formats -- either b, w{int number}, or d{int number}

The first character in the message string represent the check\_balance(), deposit(),

and withdraw() functions, respectively.

For withdraw and deposit, the auth\_challenge() validates the user permission to

execute the functions.

Returns strings to send to clients, either a confirmation or invalid message, to

be then sent back to the client.

"""

*if* (len(message) >= 1):

*if* (message[0] == "b"):

*return* self.check\_balance()

*elif* (message[0] == "w"):

*if* (self.auth\_challege() != False):

self.withdraw(message[1:])

*return* "Withdrew " + message[1:] + " dollar(s). Balance is now " + str(self.balance) + " dollar(s)."

*else*:

*return* "Invalid credentials. Please try the withdraw command again."

*elif* (message[0] == "d"):

*if* (self.auth\_challege() != False):

self.deposit(message[1:])

*return* "Deposited " + message[1:] + " dollar(s). Balance is now " + str(self.balance) + " dollar(s)."

*else*:

*return* "Invalid credentails. Please try the deposit command again."

*else*:

*return* "Improper format, please send command again."

def print\_details(*self*):

"""Print out the server details"""

print('SERVER READY @ ', self.socket.getsockname())

def write\_to\_file(*self*):

"""

Writes to the user.txt file. This function is called

once every time a UDPServer is instantiated.

"""

string = str(self.balance) + "\n" + \

self.username + "\n" + self.pin + "\n"

file = open("user.txt", "w")

*try*:

file.write(string)

file.close()

*except* Exception *as* e:

file.close() *# just in case*

sys.exit(1) *# to force stop execution*

print(e)

host = "127.0.0.1"

*# port = 0 # the OS should choose an open port for us*

port = 65434

def main():

*try*:

server\_sock = UDPServer(host, port)

server\_sock.socket.setblocking(1)

server\_sock.print\_details()

*while* True:

conn, addr = server\_sock.socket.recvfrom(2048)

server\_sock.add\_client\_details(addr)

message = conn.decode()

message = server\_sock.process\_command(message)

server\_sock.socket.sendto(message.encode(), addr)

server\_sock.socket.close()

*except* KeyboardInterrupt:

print("\nExited by Ctrl+C.")

server\_sock.socket.close()

sys.exit(1)

main()

"""

udpclient.py

A udp client made in python.

Some variable and function names from socketserver.py in python 3.7 release.

Made by Wasfi Momen.

"""

*import* socket

*import* sys

class UDPClient:

address\_family = socket.AF\_INET *# only IPv4 connections*

socket\_type = socket.SOCK\_DGRAM *# constant to use TCP socket type*

*# constructor from the socket.py module*

def \_\_init\_\_(*self*, *server\_address*, *server\_port*):

self.server\_address = server\_address

self.server\_port = server\_port

self.socket = socket.socket(self.address\_family, self.socket\_type)

def connect(*self*):

"""Attempts to connect to the server socket, throws exception and closes socket if fails."""

*try*:

self.socket.connect((self.server\_address, self.server\_port))

*except*:

self.socket.close()

*raise* Exception('Failed to connect to server socket.')

def process\_command(*self*, *number*):

"""Formats a message to be received and interpreted server-side."""

amount = 0

*if* (number == "1"):

*return* "b"

*elif* (number == "2"):

amount = input("\n\tHow much do you want to withdraw?\n")

*return* ("w" + amount)

*elif* (number == "3"):

amount = input("\n\tHow much do you want to deposit?\n")

*return* ("d" + amount)

*else*:

*return* "\nIllegal Command."

host = '127.0.0.1'

*# port = 0 # the OS should choose an open port for us*

port = 65434

def main():

*try*:

client\_sock = UDPClient(host, port)

client\_sock.connect()

*while* True: *# from python 3.7 docs examples*

print("\n")

print("\t(1) Check Balance\n")

print("\t(2) Withdraw\n")

print("\t(3) Deposit\n")

command = input("\n\tInput command by sending a number...\n")

*# get a message to send to the server*

message = client\_sock.process\_command(command)

*if* (message != "\nIllegal Command."):

*# send message to server.*

client\_sock.socket.sendto(

message.encode(), (host, port))

conn, addr = client\_sock.socket.recvfrom(2048)

message\_from\_server = conn.decode()

print("\nFrom Server: ", message\_from\_server)

*if* (message\_from\_server == "\n\tPlease give the username and pin number.\n"):

username = input("\n\tPlease give the username.\n")

pin = input("\n\tPlease give the pin number.\n")

formatted = "u" + username + " " + "p" + pin

client\_sock.socket.sendto(formatted.encode(), (host, port))

*# we make another conn, addr here to get another frame of the UDP packet.*

*# dunno how to fix except keeping track of packet data sent and received.*

conn, addr = client\_sock.socket.recvfrom(2048)

message\_from\_server = conn.decode()

print("\nFrom Server: ", message\_from\_server)

client\_sock.socket.close()

*except* KeyboardInterrupt:

print("Interrupted")

client\_sock.socket.close()

sys.exit(1)

main()

"""

tcpclient.py

A tcp client made in python.

Some variable and function names from socketserver.py in python 3.7 release.

Made by Wasfi Momen.

"""

*import* socket

*import* sys

class TCPClient:

"""TCPClient class that only binds the

socket using the specified ports in the

constructor.

All other socket operations should

be used by the .socket member access.

"""

address\_family = socket.AF\_INET *# only IPv4 connections*

socket\_type = socket.SOCK\_STREAM *# constant to use TCP socket type*

*# constructor from the socket.py module*

def \_\_init\_\_(*self*, *server\_address*, *server\_port*):

self.server\_address = server\_address

self.server\_port = server\_port

self.socket = socket.socket(self.address\_family, self.socket\_type)

def connect(*self*):

"""Attempts to connect to the server socket, throws exception and closes socket if fails."""

*try*:

self.socket.connect((self.server\_address, self.server\_port))

*except*:

self.socket.close()

*raise* Exception('Failed to connect to server socket.')

host = '127.0.0.1'

*# port = 0 # the OS should choose an open port for us*

port = 65434

def main():

*try*:

client\_sock = TCPClient(host, port)

client\_sock.connect()

*while* True: *# from python 3.7 docs examples*

sentence = input("\n\tInput a lowercase sentence...\n")

client\_sock.socket.sendall(sentence.encode())

data = client\_sock.socket.recv(1024)

print("\nFrom Server: ", data.decode())

*# server has terminated connection, clean up and close socket*

*if* (data == "STRINGS LIMIT REACHED, CLOSING CONNECTION".encode()):

print("\n\tCLIENT CONNECTION CLOSED.")

client\_sock.socket.shutdown(2)

*break*

client\_sock.socket.close()

*except* KeyboardInterrupt:

print("Interrupted")

client\_sock.socket.sendall("CLIENT REQUESTS SHUTDOWN".encode())

client\_sock.socket.shutdown(2)

client\_sock.socket.close()

sys.exit(1)

main()

"""

tcpserver.py

A tcp server made in python.

Some variable and function names from socketserver.py in python 3.7 release.

Made by Wasfi Momen.

"""

"""

REQUIRMENTS

- ~~Capitalizes strings~~

- ~~Max 10 strings~~

- ~~Client ask for termination~~ \*\*"Appropiate message" to terminate is Ctrl+C\*\*

- ~~Limit reached, send message to client~~

- ~~Multithreaded~~

ALL COMPLETE

"""

"""

ISSUES

- Non-graceful shutdown of more than two clients. Exception will be raised

but server will continue to function till the last client disconnects.

- KeyboardInterrupt not working correctly.

"""

*import* socket

*import* sys

*import* threading

class TCPServer:

"""

TCPServer class contains the necessary functions

for the application by keeping track of the strings

and only binding the socket to the specified ports in

the constructor.

All other socket operations should

be used by the .socket member access.

"""

address\_family = socket.AF\_INET *# only IPv4 connections*

socket\_type = socket.SOCK\_STREAM *# constant to use TCP socket type*

*# request\_queue\_size = 1 # we only take one connection*

num\_strings = 0 *# number of strings received*

client\_details = [] *# list of clients connected*

*# constructor from the socket.py module*

def \_\_init\_\_(*self*, *server\_address*, *server\_port*):

self.server\_address = server\_address

self.server\_port = server\_port

self.socket = socket.socket(self.address\_family, self.socket\_type)

*try*:

self.bind()

*except*:

self.socket.close()

*raise* Exception('Error in setting up connection to host port.')

def add\_client\_details(*self*, *addr*):

"""

Add the tuples of our connected clients to a list.

This is not currently used, but would be great to

gracefully except during shutdowns of client sockets.

"""

*if* addr in self.client\_details:

*return*

self.client\_details.append(addr)

print("Connected by: ", addr)

def capitalize\_string(*self*, *client\_string*):

"""Capitalizes string that is received from the client."""

self.received\_string()

*return* client\_string.upper()

def bind(*self*):

"""Binds the socket to a port"""

*try*:

*# option to reuse addresses for sockets*

self.socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

*# a tuple of the address and port is used for the bind function*

self.socket.bind((self.server\_address, self.server\_port))

*except*:

self.socket.close()

*raise* Exception(

'Error in binding socket to address and port specified.')

def process\_clients(*self*, *connection*, *address*):

"""

Main logic of the program. Takes care of the shutdow of

client sockets and termination of server when strings limit

is reached.

"""

*while* True:

sentence = connection.recv(1024).decode()

*# break out of while loop if shutdown request is given*

*if* (sentence == "CLIENT REQUESTS SHUTDOWN"):

print("\nCLIENT HAS REQUESTED SHUTDOWN")

*# 2 means SHUT\_RDWR or disable read and write, Windows only takes numbers*

connection.shutdown(2)

connection.close()

*break*

*# break out of while loop if number of strings exceeds 10*

*if* (self.get\_num\_of\_strings() >= 10):

print(

"STRINGS LIMIT REACHED, SHUTTING DOWN AND CLOSING SOCKET")

connection.sendall(

"STRINGS LIMIT REACHED, CLOSING CONNECTION".encode())

connection.shutdown(2)

connection.close()

*break*

print("\nSentence is: ", sentence)

connection.sendall(

self.capitalize\_string(sentence).encode() + "\nAwaiting next string...".encode())

print("Number of strings ",

self.get\_num\_of\_strings())

self.socket.close()

def start\_client\_listen(*self*):

"""

Starts the mulithreaded listen for the

clients. Also adds clients to a list of

connected clients.

Every client added will go through

here to be processed by process\_clients()

"""

*try*:

self.socket.listen(4) *# listen to 4 clients max*

self.print\_details()

*while* True:

conn, addr = self.socket.accept()

self.add\_client\_details(addr)

threading.Thread(*target*=self.process\_clients,

*args*=(conn, addr)).start()

self.socket.close()

*except* KeyboardInterrupt:

print("\nExited by Ctrl+C.")

self.socket.close()

sys.exit(1)

def print\_details(*self*):

"""Print out the server details"""

print('SERVER READY @ ', self.socket.getsockname())

def received\_string(*self*):

"""Increments the number of string received by the server."""

self.num\_strings += 1

def get\_num\_of\_strings(*self*):

"""Returns the number of strings the server has received."""

*return* self.num\_strings

host = "127.0.0.1"

*# port = 0 # the OS should choose an open port for us*

port = 65434

TCPServer(host, port).start\_client\_listen()