Individual Portfolio Assignment 1

Instructions to run the Python program:

1. Creating a “chat room”/server by typing in the correct command in the terminal. Example: “python3 server.py 1234”

   This will create a chat room in the port 1234, and other chat rooms can be created in other ports if that would be wanted.

   Typing the command: “python3 server.py -h” will show additional info such as needed positional arguments.

```
usage: server.py [-h] port

You can start a server and listen for incoming connections. Example of how:
"python3 system.py 1234"

positional arguments:
  port        Argument 1: Port number you want the server running on. Write it
              in numbers only!
```

2. Opening new terminal windows for the clients you want to connect to the server. The number of windows depends on how many clients you want to connect to the server.

   Connecting the clients by typing the correct command in the terminal. Example: “python3 client.py 1234 Enes”. Writing the name of one of the four bots (Alice, Bob, Dora or Chuck) will add a bot-client to the chat room. Writing any other name adds a host-client to the chat room, which means they won’t respond to you unless they type themselves.

   Similar to the TCP server, typing the command: “python3 client.py -h” will show additional info such as positional arguments for the TCP client.

```
usage: client.py [-h] port name

You can connect to a chat room. If you would like a bot client to join the
chat, write one of the bot names (Alice, Bob, Dora or Chuck) Example of how:
client.py 1234 Enes

positional arguments:
  port        Argument 1: Port number you wanna connect to. Write it in
              numbers only!
  name        Argument 2: The client name you would like to connect with.
              Write a bot name if you would like to connect a bot
```

3. The chat room should now be working. Suggest an action from a host-client such as: “Let’s cook some food”, and the connected bots should respond to that to the best of their abilities.
   If there are only host-clients in the chat room, it should be working as any other chat room. Also, if any of the bot-clients attempts to write a message, the other bots won’t respond to this as it’s not from a host-client

4. If you would like to leave the chat room, closing the terminal window would do the trick. The other connected clients, and the server window, will get a notification both when you first connected, and when you disconnected from the chat room.

Github link for the code:

For some unknown reason, I couldn’t login to my usual Github account and had to make a new one for this, which should explain the name “eneserginmidlertidig”
https://github.com/eneserginmidlertidig/portfolio1

Quick code explanation:

PS: the code itself has inline code-comments in the places I thought it was necessary but I figured I might explain it in case the code isn't clear enough

Creating a server socket which is listening, and running the function client_connecting from server side.

client_connecting accepts any incoming connections (is constantly listening) and sends a string 'not_connected' to the client that connects. On client side, client_handling is running and if the msg received from server is 'not_connected', client sends the name of the client back to server. Server then adds client to the client list, and notification gets sent to all the other connected clients (if there is any) telling them that this client has connected.

If a host-client has been connected to the server, host_sending_messages is running and any messages sent from the host gets sent to the server side, and handled by server_client_handling. From server_client_handling, the message gets broadcasted to all clients where it gets handled by client_handling on client side accordingly. If the message is an actual written message and not a notification of a client connecting/disconnecting, and if the message is written by a host, the bot-client functions are called upon and their output printed to all client windows.

Every time a host sends a message initiates "communication" between host_sending_message client side, client_handling client side and server_client_handling server side. The function server_client_handling on server side also takes care of telling all clients whenever a client gets disconnected from the server. It does this by broadcasting to the other clients the same way it does whenever a host-written message is being sent.

Demonstration of the chat room

```
Do you guys wanna grab something to eat?

Enes: Do you guys wanna grab something to eat?
Chuck: Such a good idea! Let´s go eating
Alice: eating! I absolutely love eating!

we can walk to the restaurant

Enes: we can walk to the restaurant
Chuck: YES! I want to walk too!

Alice: walking! I absolutely love walking!

And we could possibly steal something on the way

Enes: And we could possibly steal something on the way
Alice: I wanna avoid stealing

Chuck: I think we should do something else other than stealing
```

```
lets go murder someone

Enes: lets go murder someone
Dora: Yes definitely, let´s go murdering!

and then we can steal their belongings

Enes: and then we can steal their belongings
Dora: stealing? If you aren´t going to tell anyone I´m down. Sounds great to me...
```

```
Hello are you a host-client?

Enes: Hello are you a host-client?
Peter: Yes I am, you too?

Yes

Enes: Yes
Peter: Nice

Kevin has connected to the chat!
Peter: look another host-client

Kevin: hello guys
```

As you clearly can see by my demonstration, I've been very bored. I've made one of the bots, Dora, evil and an another of the bots Bob confused.

server.py code

```
import argparse
import socket
import threading
import time

# parser that checks if the user has written the needed arguments, which in
this case is the port number
parser = argparse.ArgumentParser(description='You can start a server and
listen for incoming connections.'
                                             '\nExample of how: "python3
system.py 1234"')
parser.add_argument('port', type=int,
                    help='Argument 1: Port number you want the server
running on. Write it in numbers only!')
args = parser.parse_args()
port = args.port

# creating a server socket, binding the ip address and the port given by
user and listening for connections
socketServer = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socketServer.bind(('localhost', port))
socketServer.listen()

# keeping track of clients and names
clients = []
clientsName = []


# broadcasting/sending message from host clients to the other clients
def broadcast(msg, client):
    for i in clients:
        # we don't want to send the message back to the original writer of
the message
        if i is not client:
            i.send(msg)


# Handling of all the clients sending messages or disconnecting
def server_client_handling(client):
    while True:
        try:
            msg = client.recv(1024)
            split = msg.decode().split(": ")

            if split[1] == "KICKALL":  # this does not work as intended,
                # but removing it also causes other problems that I don't
have the time to solve
                time.sleep(0.1)
                print("Disconnecting all the clients!")
                for i in clients:
                    i.close()

            else:
                time.sleep(0.5)
                broadcast(msg, client)
        except:
            # removing client and clientName from the lists
```

```
            index = clients.index(client)
            clients.remove(client)
            name = clientsName[index]
            clientsName.remove(name)
            client.close()
            broadcast(f'{name} has disconnected from the
chat!'.encode('utf-8'),
                      client)  # broadcasting to all the other connected
clients
            print(f'{name} has disconnected from the chat!')  # prints to
server window
            break


# connects the clients to the server.
def client_connecting():
    print('Server is running! Currently listening to connections...\n')
    while True:
        client, address = socketServer.accept()  # accepting connection
from the client
        client.send('not_connected'.encode('utf-8'))  # sending
'not_connected'
        # which is used for checking if a client is connected or not in
client side
        name = client.recv(1024).decode('utf-8')  # receives name from the
client
        clients.append(client)
        clientsName.append(name)
        print(f'{name} has connected to the chat!')  # prints to server
window
        broadcast(f'{name} has connected to the chat!'.encode('utf-8'),
                  client)  # broadcasts to all currently connected clients
        client.send(
            'You have connected to the chat!\nYou can disconnect from the
chat by closing the window.'.encode(
                'utf-8'))  # prints it to the original writer
        handling_thread = threading.Thread(target=server_client_handling,
args=(client,))
        handling_thread.start()


client_connecting()
```

client.py code

```
import argparse
import socket
import threading
import time
import random

bot_clients = ['alice', 'bob', 'dora', 'chuck']

# parser that checks if the user has written the needed arguments,
# which in this case is the port number and name of the client
parser = argparse.ArgumentParser(
    description='You can connect to a chat room. If you would like a bot
client to join the chat,'
                ' write one of the bot names (Alice, Bob, Dora or
Chuck)\nExample of how: client.py 1234 Enes')
parser.add_argument('port', type=int, help='Argument 1: Port number you
wanna connect to. Write it in numbers only!')
parser.add_argument('name', type=str, help='Argument 2: The client name you
would like to connect with.'
                                           ' Write a bot name if you would
like to connect a bot')
args = parser.parse_args()
port = args.port
name = args.name

# creating a client socket and then connecting to the user given port
socketClient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socketClient.connect(('localhost', port))

# list of positive and negative actions
positiveAction = ['eat', 'walk', 'laugh', 'kiss', 'paint', 'cook']
negativeAction = ['cry', 'fight', 'bully', 'shout', 'steal', 'murder']


# BOT 1 - alice: wants to do the positive actions, and not the negative
actions
def alice(action):
    if action in positiveAction:
        positiveAnswers = [
            '{}: I´d like {}.'
            .format(name, action + 'ing'),
            '{}: I think {} sounds great!'
            .format(name, action + 'ing'),
            '{}: {}! I absolutely love {}!'
            .format(name, action + 'ing', action + 'ing')]
        output = random.choice(positiveAnswers)

    elif action in negativeAction:
        negativeAnswers = [
            '{}: I wanna avoid {}'
            .format(name, action + 'ing'),
            '{}: {}? Don´t you have anything else to do?'
            .format(name, action + 'ing'),
            '{}: I think we should do something else other than {}'
            .format(name, action + 'ing')]
        output = random.choice(negativeAnswers)

    else:
        noActionAnswers = [
```

```
            '{}: What are you saying? Come on man...'.format(name),
            '{}: Nope, not happening.'.format(name),
            '{}: I´m not interested.'.format(name)]
        output = random.choice(noActionAnswers)
    return output + '\n'


# BOT 2 - bob: says no to every action and sometimes suggest something else
(bob is confused)
def bob(action):
    if action in negativeAction or action in positiveAction:
        answers = [
            '{}: I´m not really interested in {}. Can we do some {}
instead?'
            .format(name, action + 'ing', random.choice(negativeAction) +
'ing'),
            '{}: You want to {}? Doesnt {} sound better instead?'
            .format(name, action, random.choice(positiveAction) + 'ing'),
            '{}: No way I´m {}, Let´s just do something else...'
            .format(name, action + 'ing')]
        output = random.choice(answers)

    else:
        noActionAnswers = [
            '{}: What? Definitely no.'.format(name),
            '{}: Yeah, not doing that.'.format(name),
            '{}: Aren´t you supposed to say something
useful?'.format(name)]
        output = random.choice(noActionAnswers)
    return output + '\n'


# BOT 3 - dora: thinks the positive actions are boring, and wants to do the
negative actions (dora is evil)
def dora(action):
    if action in positiveAction:
        positiveAnswers = [
            '{}: {} is boring. I´m not interested in that.'
            .format(name, action + 'ing'),
            '{}: I don´t wanna {}. Not really for me.'
            .format(name, action),
            '{}: Really? {}? You can come up with something more exciting
than {}...'
            .format(name, action + 'ing', action + 'ing')]
        output = random.choice(positiveAnswers)

    elif action in negativeAction:
        negativeAnswers = [
            '{}: You wanna {}? I´m okay with that but maybe we should keep
it private.'
            .format(name, action),
            '{}: {}? If you aren´t going to tell anyone I´m down. Sounds
great to me...'
            .format(name, action + 'ing'),
            '{}: Yes definitely, let´s go {}!'
            .format(name, action + 'ing')]
        output = random.choice(negativeAnswers)

    else:
        noActionAnswers = [
            '{}: Come on man, suggest something to do!'.format(name),
```

```
            '{}: I don´t even know what I´m doing here.'.format(name)]
        output = random.choice(noActionAnswers)
    return output + '\n'


# BOT 4 - chuck: wants to do the positive actions, and not the negative
actions
def chuck(action):
    if action in positiveAction:
        positiveAnswers = [
            '{}: {}, sounds great to be honest!.'
            .format(name, action + 'ing'),
            '{}: Such a good idea! Let´s go {}'
            .format(name, action + 'ing'),
            '{}: YES! I want to {} too!'
            .format(name, action)]
        output = random.choice(positiveAnswers)

    elif action in negativeAction:
        negativeAnswers = [
            '{}: No chance I´m {}.'
            .format(name, action + 'ing'),
            '{}: What? {} sucks. I´m definitely not doing that!'
            .format(name, action + 'ing'),
            '{}: I think we should do something else other than {}'
            .format(name, action + 'ing')]
        output = random.choice(negativeAnswers)

    else:
        noActionAnswers = [
            '{}: I´m not doing that.'.format(name),
            '{}: That´s just boring. Suggest something
else...'.format(name),
            '{}: I´m not too sure to be honest.'.format(name)]
        output = random.choice(noActionAnswers)
    return output + '\n'


# this is the host sending messages, gets broadcasted to all the connected
clients
def host_sending_message():
    while True:
        msg = f'{name}: {input()}'  # formats the message
        split = msg.split(": ")
        if split[1] == "":
            print('Can´t be empty!')
            continue
        else:
            time.sleep(0.5)
            print('')
            print_send(msg)


def print_send(msg):
    print(msg)
    socketClient.send(msg.encode('utf-8'))


# this is used for handling of clients i.e. connecting not connected
clients to the server,
# and also making sure that clients receive messages from other clients
```

```
def client_handling():
    while True:
        msg = socketClient.recv(1024).decode('utf-8')

        if msg.lower() == 'not_connected':  # 'not_connected' is sent from
server if the client is not connected,
            # goes into this if-statement if the client isn't connected
            socketClient.send(name.encode('utf-8'))  # sends the clients
name back to the server

        else:
            if ":" in msg:  # goes in here if its a sent message,
                # and not a notification telling if a client has
connected/disconnected
                split = msg.split(": ")

                if split[0].lower() not in bot_clients:  # receiving
messages from host,
                    # will trigger bot functions later in this if-statement

                    message = ""
                    allActions = positiveAction + negativeAction
                    for i in allActions:  # loop that checks if the host-
client message includes any of the actions
                        if i in msg.lower():  # an action has been found in
the host-client message
                            message = i
                            break

                    output = ''
                    for i in bot_clients:  # loop that goes through the bot
names,
                        # and runs their assigned function using builtin
function eval
                        if i == name.lower():
                            output = eval(i + '(message)')
                            break

                    print(msg)  # message that is received from the host
                    print_send(output)  # message that is printed to the
current client window, and to the other clients

                else:  # receiving messages from bots, won't trigger bot
function
                    time.sleep(1)
                    print(msg)

            else:  # notifies that a client has connected/disconnected the
chat to the other clients
                print(msg)


handling_thread = threading.Thread(target=client_handling)
handling_thread.start()

if name not in bot_clients:
    sending_thread = threading.Thread(target=host_sending_message)
    sending_thread.start()
```