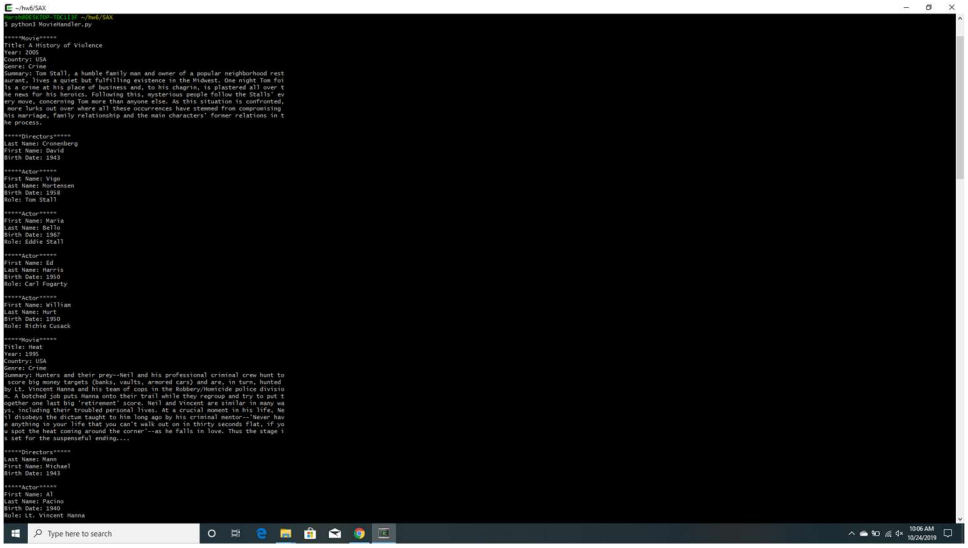
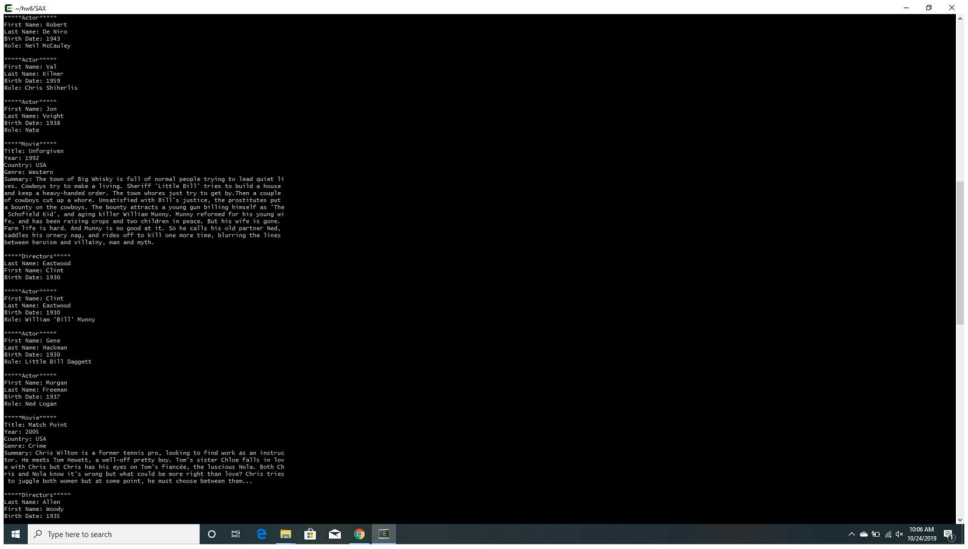
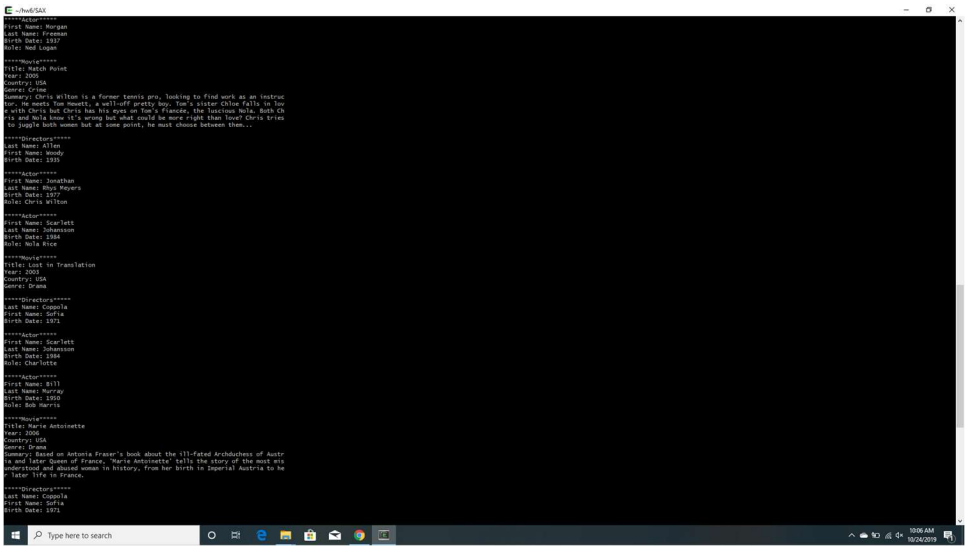
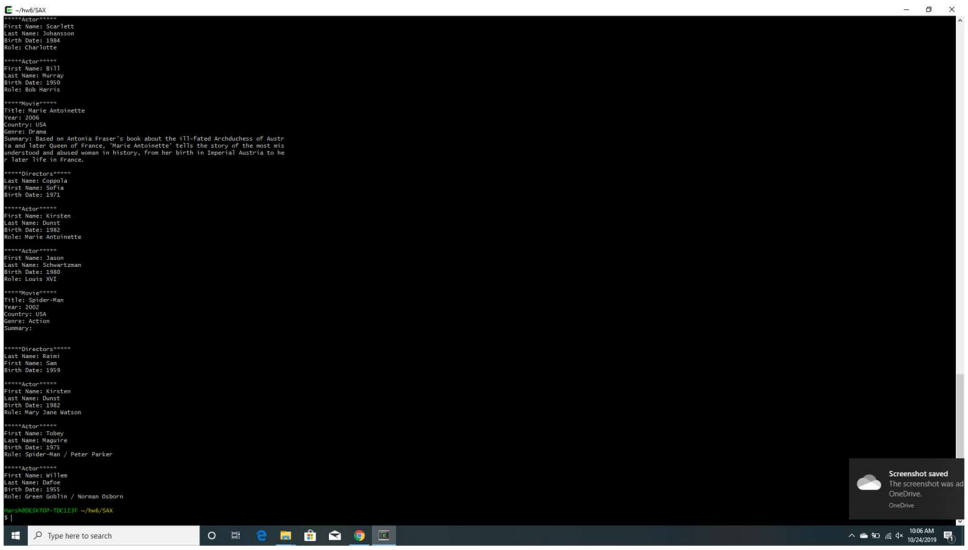
Q-2) Parsing with SAX

Execution Result:-









MovieHandler.py:-

```
#!/usr/bin/python


import xml.sax


class MovieHandler( xml.sax.ContentHandler ):

   def __init__(self):

      self.CurrentData = ""
```

```
    self.title = ""
    self.year = ""
    self.country = ""
    self.genre = ""
    self.summary = ""
    self.director = ""
    self.last_name = ""
    self.first_name = ""
    self.birth_date = ""
    self.role = ""

# Call when an element starts
def startElement(self, tag, attributes):
    self.CurrentData = tag
    if tag == "movie":
        print ("\n*****Movie*****")
    elif tag == "director":
        print ("\n*****Directors*****")
    elif tag == "actor":
        print ("\n*****Actor*****")

# Call when an elements ends
def endElement(self, tag):
```

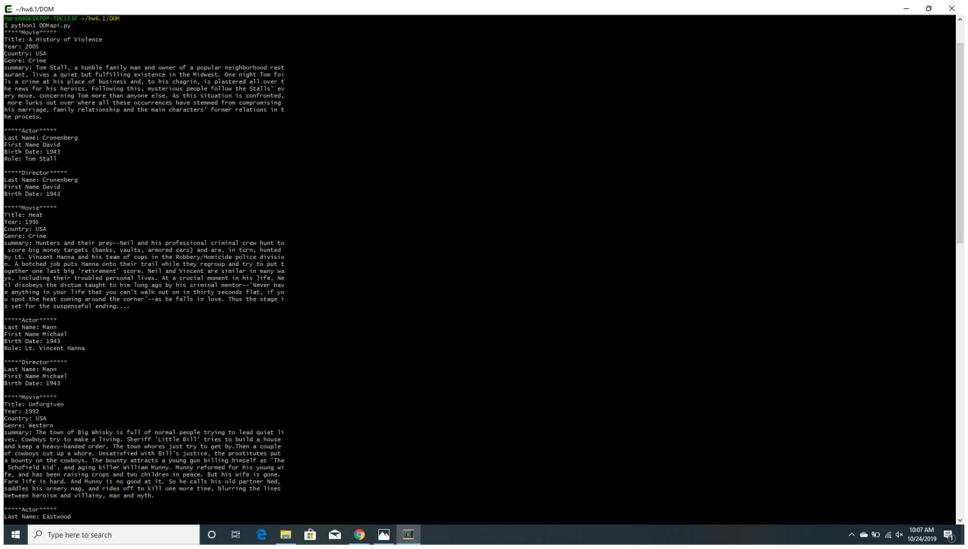
```
if self.CurrentData == "title":
  print ("Title:", self.title)
elif self.CurrentData == "year":
  print ("Year:", self.year)
elif self.CurrentData == "country":
  print ("Country:", self.country)
elif self.CurrentData == "genre":
  print ("Genre:", self.genre)
elif self.CurrentData == "summary":
  print ("Summary:", self.summary)
elif self.CurrentData == "director":
  print ("Director:", self.director)
elif self.CurrentData == "last_name":
  print ("Last Name:", self.last_name)
elif self.CurrentData == "first_name":
  print ("First Name:", self.first_name)
elif self.CurrentData == "birth_date":
  print ("Birth Date:", self.birth_date)
elif self.CurrentData == "role":
  print ("Role:", self.role)
self.CurrentData = ""
```

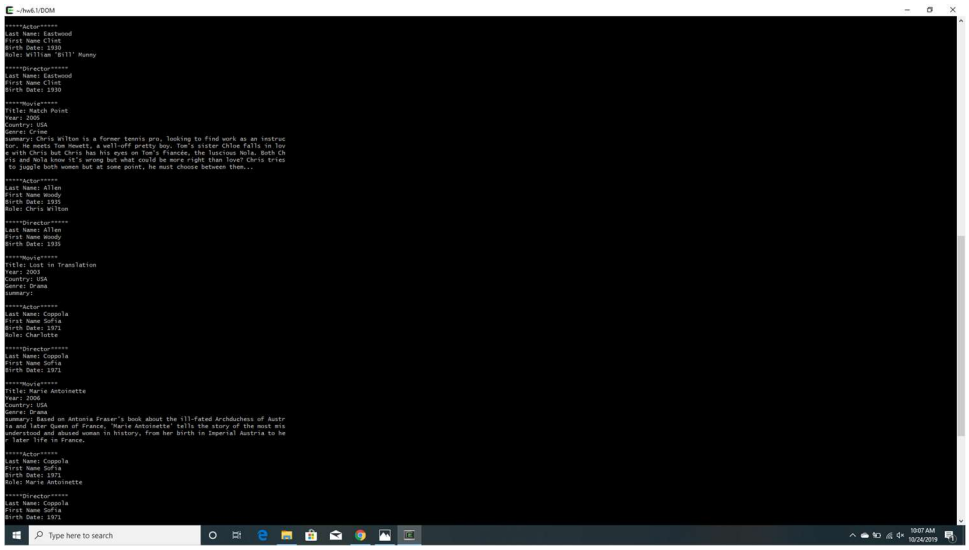
```
# Call when a character is read
def characters(self, content):
  if self.CurrentData == "title":
    self.title = content
  elif self.CurrentData == "year":
    self.year = content
  elif self.CurrentData == "country":
    self.country = content
  elif self.CurrentData == "genre":
    self.genre = content
  elif self.CurrentData == "summary":
    self.summary = content
  elif self.CurrentData == "director":
    self.director = content
  elif self.CurrentData == "last_name":
    self.last_name = content
  elif self.CurrentData == "first_name":
    self.first_name = content
  elif self.CurrentData == "birth_date":
    self.birth_date = content
  elif self.CurrentData == "role":
    self.role = content
```

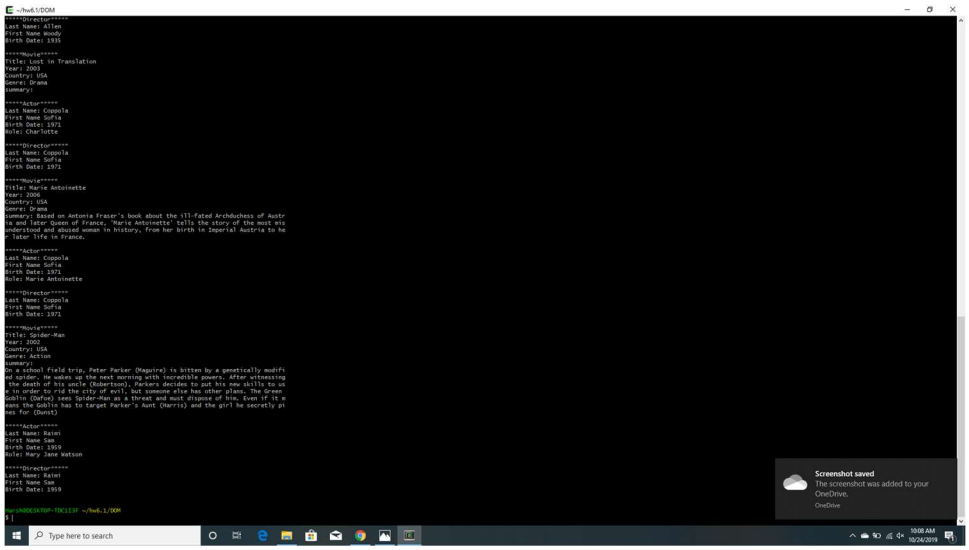
```
if ( __name__ == "__main__"):

  # create an XMLReader
  parser = xml.sax.make_parser()
  # turn off namepsaces
  parser.setFeature(xml.sax.handler.feature_namespaces, 0)

  # override the default ContextHandler
  Handler = MovieHandler()
  parser.setContentHandler( Handler )

  parser.parse("movies_long.xml")
```

Q-3) Parsing with DOMapi.py

Execution Result:-







DOMapi.py:-

#!/usr/bin/python

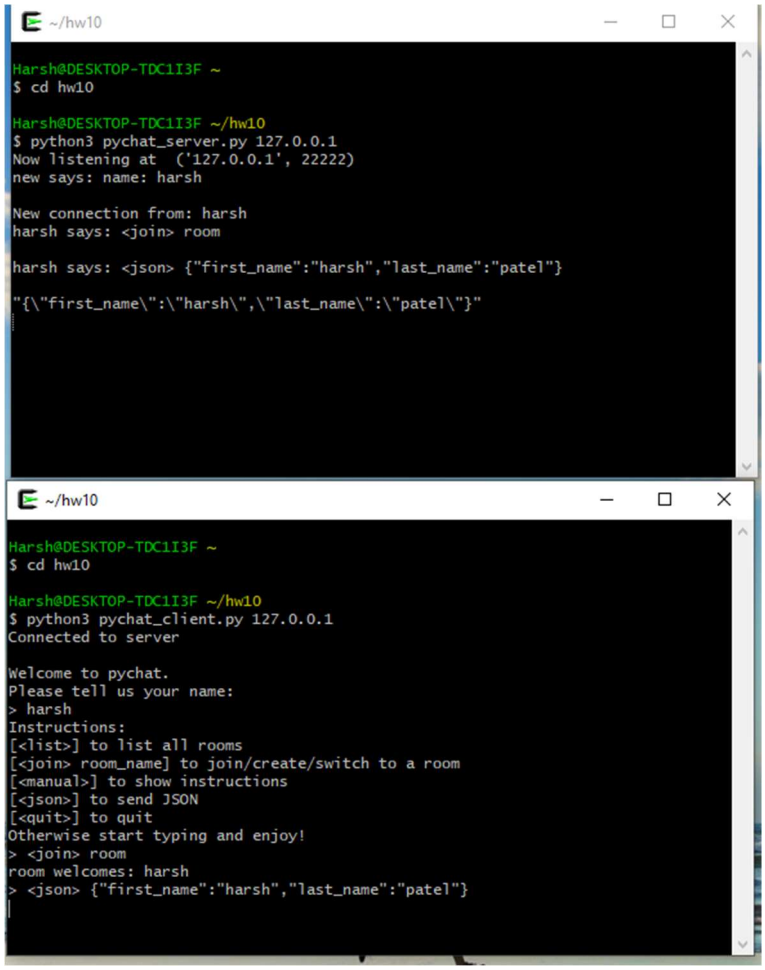
```
from xml.dom.minidom import parse
import xml.dom.minidom

# Open XML document using minidom parser
DOMTree = xml.dom.minidom.parse("movies_long.xml")
```

```
collection = DOMTree.documentElement

# Get all the movies in the collection
movies = collection.getElementsByTagName("movie")

# Print detail of each movie.
for movie in movies:
  print ("*****Movie*****")

  title = movie.getElementsByTagName('title')[0]
  print ("Title: %s" % title.childNodes[0].data)
  year = movie.getElementsByTagName('year')[0]
  print ("Year: %s" % year.childNodes[0].data)
  country = movie.getElementsByTagName('country')[0]
  print ("Country: %s" % country.childNodes[0].data)
  genre = movie.getElementsByTagName('genre')[0]
  print ("Genre: %s" % genre.childNodes[0].data)
  summary = movie.getElementsByTagName('summary')[0]
  print ("summary: %s" % summary.childNodes[0].data + "\n")

  if movie.getElementsByTagName("actor"):
    print ("*****Actor*****")
    last_name = movie.getElementsByTagName('last_name')[0]
```

```
    print ("Last Name: %s" % last_name.childNodes[0].data)
    first_name = movie.getElementsByTagName('first_name')[0]
    print ("First Name %s" % first_name.childNodes[0].data)
    birth_date = movie.getElementsByTagName('birth_date')[0]
    print ("Birth Date: %s" % birth_date.childNodes[0].data)
    role = movie.getElementsByTagName('role')[0]
    print ("Role: %s" % role.childNodes[0].data + "\n")

  if movie.getElementsByTagName("director"):
    print ("*****Director*****")
    last_name = movie.getElementsByTagName('last_name')[0]
    print ("Last Name: %s" % last_name.childNodes[0].data)
    first_name = movie.getElementsByTagName('first_name')[0]
    print ("First Name %s" % first_name.childNodes[0].data)
    birth_date = movie.getElementsByTagName('birth_date')[0]
    print ("Birth Date: %s" % birth_date.childNodes[0].data + "\n")
```

Q-4) pychat + json

Execution Result:-



Pychat_client.py:-

```
import select, socket, sys

from pychat_util import Room, Hall, Player

import pychat_util

READ_BUFFER = 4096
```

```
if len(sys.argv) < 2:

    print("Usage: Python3 client.py [hostname]", file = sys.stderr)

    sys.exit(1)

else:

    server_connection = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)

    server_connection.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)

    server_connection.connect((sys.argv[1], pychat_util.PORT))



def prompt():

    print('>', end=' ', flush = True)
```

```
print("Connected to server\n")

msg_prefix = ''



socket_list = [sys.stdin, server_connection]



while True:

    read_sockets, write_sockets, error_sockets =
select.select(socket_list, [], [])

    for s in read_sockets:

        if s is server_connection: # incoming message

            msg = s.recv(READ_BUFFER)

            if not msg:

                print("Server down!")

                sys.exit(2)
```

```
else:
    if msg == pychat_util.QUIT_STRING.encode():
        sys.stdout.write('Bye\n')
        sys.exit(2)
    else:
        sys.stdout.write(msg.decode())
        if 'Please tell us your name' in msg.decode():
            msg_prefix = 'name: ' # identifier for name
        else:
            msg_prefix = ''
        prompt()
```

```
        else:

            msg = msg_prefix + sys.stdin.readline()

            server_connection.sendall(msg.encode())
```

Pychat_server.py:-

```
# implementing 3-tier structure: Hall --> Room --> Clients;

# 14-Jun-2013



import select, socket, sys, pdb

from pychat_util import Hall, Room, Player

import pychat_util



READ_BUFFER = 4096



host = sys.argv[1] if len(sys.argv) >= 2 else ''
```

```
listen_sock = pychat_util.create_socket((host, pychat_util.PORT))



hall = Hall()

connection_list = []

connection_list.append(listen_sock)



while True:

    # Player.fileno()

    read_players, write_players, error_sockets =
select.select(connection_list, [], [])

    for player in read_players:

        if player is listen_sock: # new connection, player is a socket

            new_socket, add = player.accept()
```

```
    new_player = Player(new_socket)

    connection_list.append(new_player)

    hall.welcome_new(new_player)



else: # new message

    msg = player.socket.recv(READ_BUFFER)

    if msg:

        msg = msg.decode().lower()

        hall.handle_msg(player, msg)

    else:

        player.socket.close()

        connection_list.remove(player)
```

```
    for sock in error_sockets: # close error sockets

        sock.close()

        connection_list.remove(sock)
```

Pychat_util.py:-

```
#################################################################
##
# implementing 3-tier structure:
#       Hall --> Room --> Clients;
# 14-Jun-2013
#################################################################
##
import socket, pdb, json

MAX_CLIENTS = 30
PORT = 22222
QUIT_STRING = '<$quit$>'


def create_socket(address):
```

```
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.setblocking(0)
    s.bind(address)
    s.listen(MAX_CLIENTS)
    print("Now listening at ", address)
    return s

class Hall:
    def __init__(self):
        self.rooms = {} # {room_name: Room}
        self.room_player_map = {} # {playerName: roomName}

    def welcome_new(self, new_player):
        new_player.socket.sendall(b'Welcome to pychat.\nPlease tell us
your name:\n')

    def list_rooms(self, player):

        if len(self.rooms) == 0:
            msg = 'Oops, no active rooms currently. Create your own!\n' \
                + 'Use [<join> room_name] to create a room.\n'
            player.socket.sendall(msg.encode())
```

```
        else:
            msg = 'Listing current rooms...\n'
            for room in self.rooms:
                msg += room + ": " + str(len(self.rooms[room].players)) + "
player(s)\n"
            player.socket.sendall(msg.encode())

    def handle_msg(self, player, msg):
        instructions = b'Instructions:\n'\
            + b'[<list>] to list all rooms\n'\
            + b'[<join> room_name] to join/create/switch to a room\n' \
            + b'[<manual>] to show instructions\n' \
            + b'[<json>] to send JSON\n' \
            + b'[<quit>] to quit\n' \
            + b'Otherwise start typing and enjoy!' \
            + b'\n'

        print(player.name + " says: " + msg)
        if "name:" in msg:
            name = msg.split()[1]
            player.name = name
            print("New connection from:", player.name)
            player.socket.sendall(instructions)
```

```
        elif "<join>" in msg:
            same_room = False
            if len(msg.split()) >= 2: # error check
                room_name = msg.split()[1]
                if player.name in self.room_player_map: # switching?
                    if self.room_player_map[player.name] == room_name:
                        player.socket.sendall(b'You are already in room: ' +
room_name.encode())
                        same_room = True
                    else: # switch
                        old_room = self.room_player_map[player.name]
                        self.rooms[old_room].remove_player(player)
                if not same_room:
                    if not room_name in self.rooms: # new room:
                        new_room = Room(room_name)
                        self.rooms[room_name] = new_room
                    self.rooms[room_name].players.append(player)
                    self.rooms[room_name].welcome_new(player)
                    self.room_player_map[player.name] = room_name
            else:
                player.socket.sendall(instructions)
```

```
elif "<list>" in msg:
    self.list_rooms(player)

elif "<manual>" in msg:
    player.socket.sendall(instructions)

elif "<json>" in msg:
    if len(msg.split()) >= 2: # error check
        json_string = msg.split()[1]
        print(json.dumps(json_string))
    else:
        player.socket.sendall(instructions)

elif "<quit>" in msg:
    player.socket.sendall(QUIT_STRING.encode())
    self.remove_player(player)

else:
    ##################################
    # check if in a room or not first
    ##################################
    if player.name in self.room_player_map:
```

```
self.rooms[self.room_player_map[player.name]].broadcast(player,
msg.encode())
        else:
            msg = 'You are currently not in any room! \n' \
                + 'Use [<list>] to see available rooms! \n' \
                + 'Use [<join> room_name] to join a room! \n'
            player.socket.sendall(msg.encode())

    def remove_player(self, player):
        if player.name in self.room_player_map:

self.rooms[self.room_player_map[player.name]].remove_player(player
)
        del self.room_player_map[player.name]
        print("Player: " + player.name + " has left\n")
class Room:
    def __init__(self, name):
        self.players = [] # a list of sockets
        self.name = name

    def welcome_new(self, from_player):
        msg = self.name + " welcomes: " + from_player.name + '\n'
        for player in self.players:
```

```
        player.socket.sendall(msg.encode())

    def broadcast(self, from_player, msg):
        msg = from_player.name.encode() + b":" + msg
        for player in self.players:
            player.socket.sendall(msg)

    def remove_player(self, player):
        self.players.remove(player)
        leave_msg = player.name.encode() + b"has left the room\n"
        self.broadcast(player, leave_msg)

class Player:
    def __init__(self, socket, name = "new"):
        socket.setblocking(0)
        self.socket = socket
        self.name = name

    def fileno(self):
        return self.socket.fileno()
```