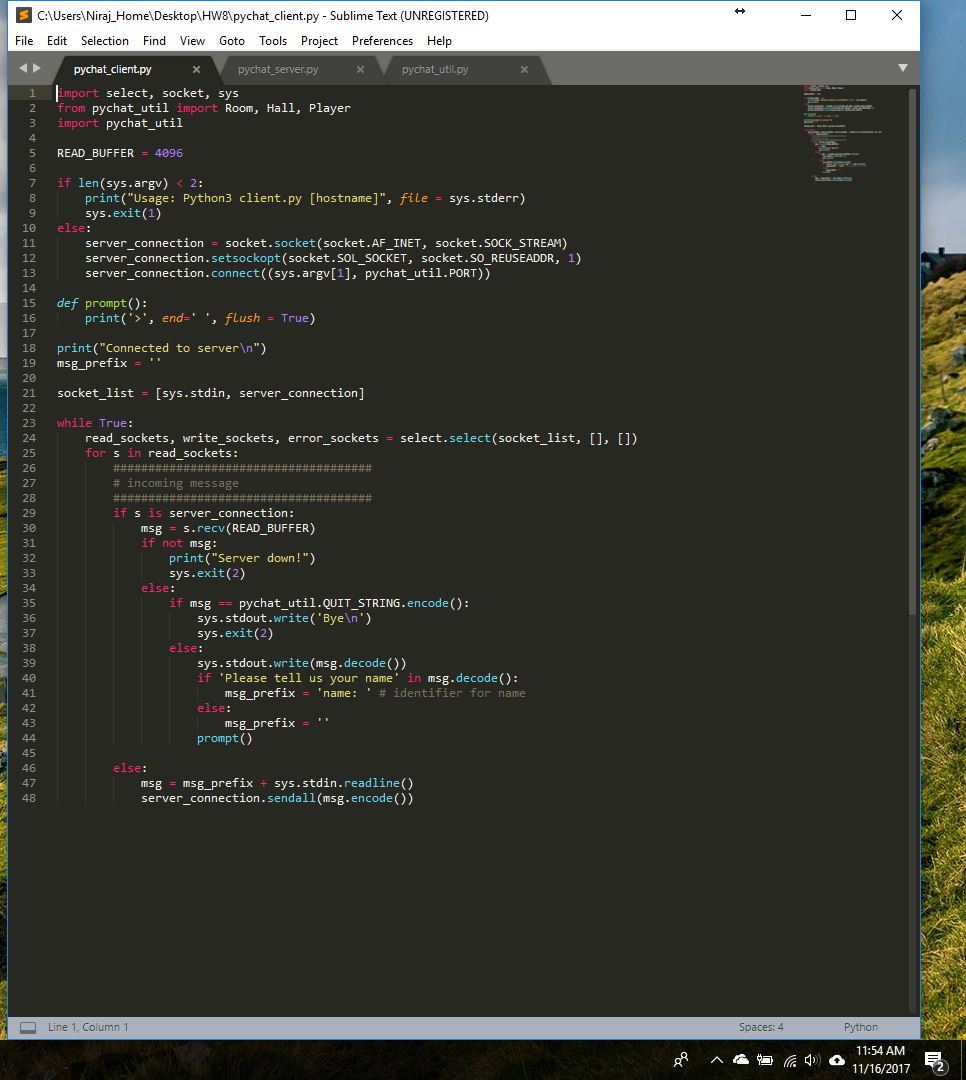
NAME : NIRAJ THANKI SID : 19376 CLASS: CS531



Source Code :

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:

#####################################

# incoming message

#####################################

if s is server\_connection:

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:

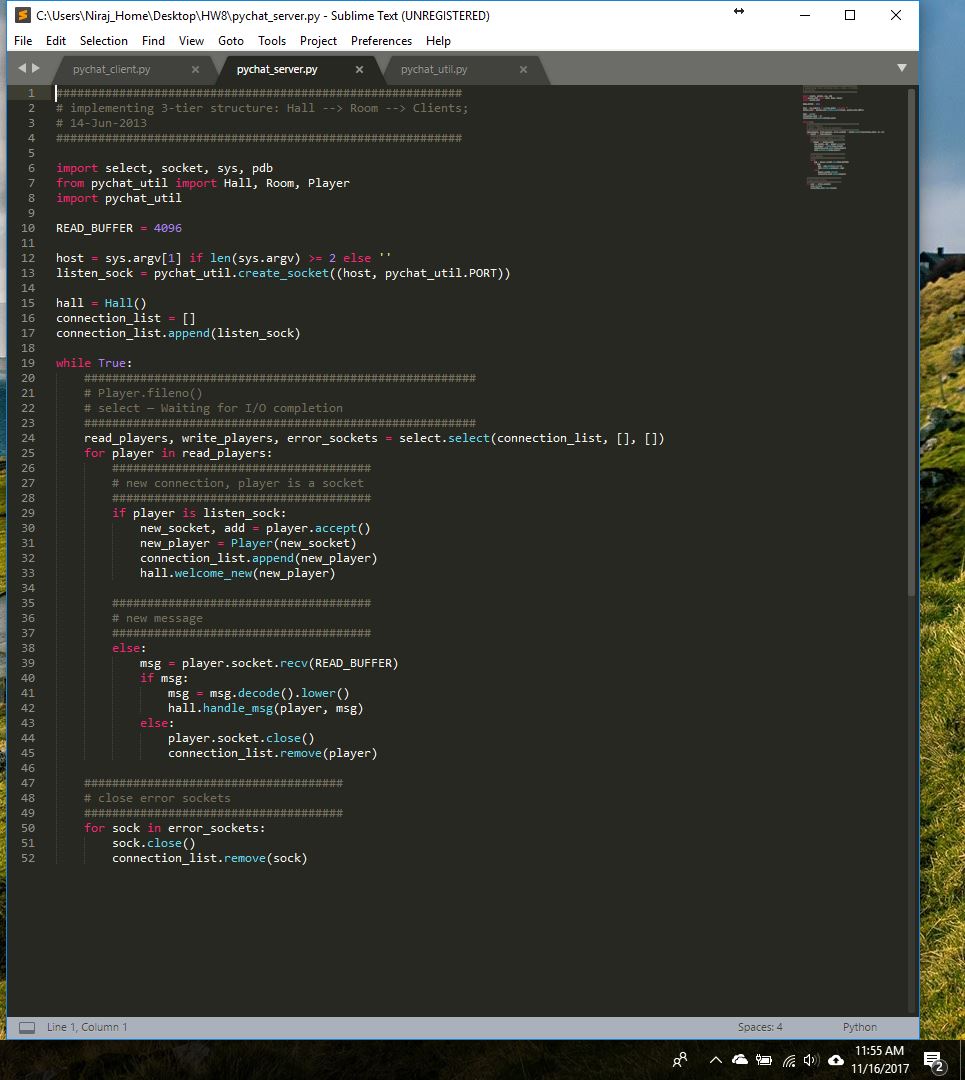
msg\_prefix = ''

prompt()

else:

msg = msg\_prefix + sys.stdin.readline()

server\_connection.sendall(msg.encode())



Source Code:

##########################################################

# 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()

# select — Waiting for I/O completion

########################################################

read\_players, write\_players, error\_sockets = select.select(connection\_list, [], [])

for player in read\_players:

#####################################

# new connection, player is a socket

#####################################

if player is listen\_sock:

new\_socket, add = player.accept()

new\_player = Player(new\_socket)

connection\_list.append(new\_player)

hall.welcome\_new(new\_player)

#####################################

# new message

#####################################

else:

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)

#####################################

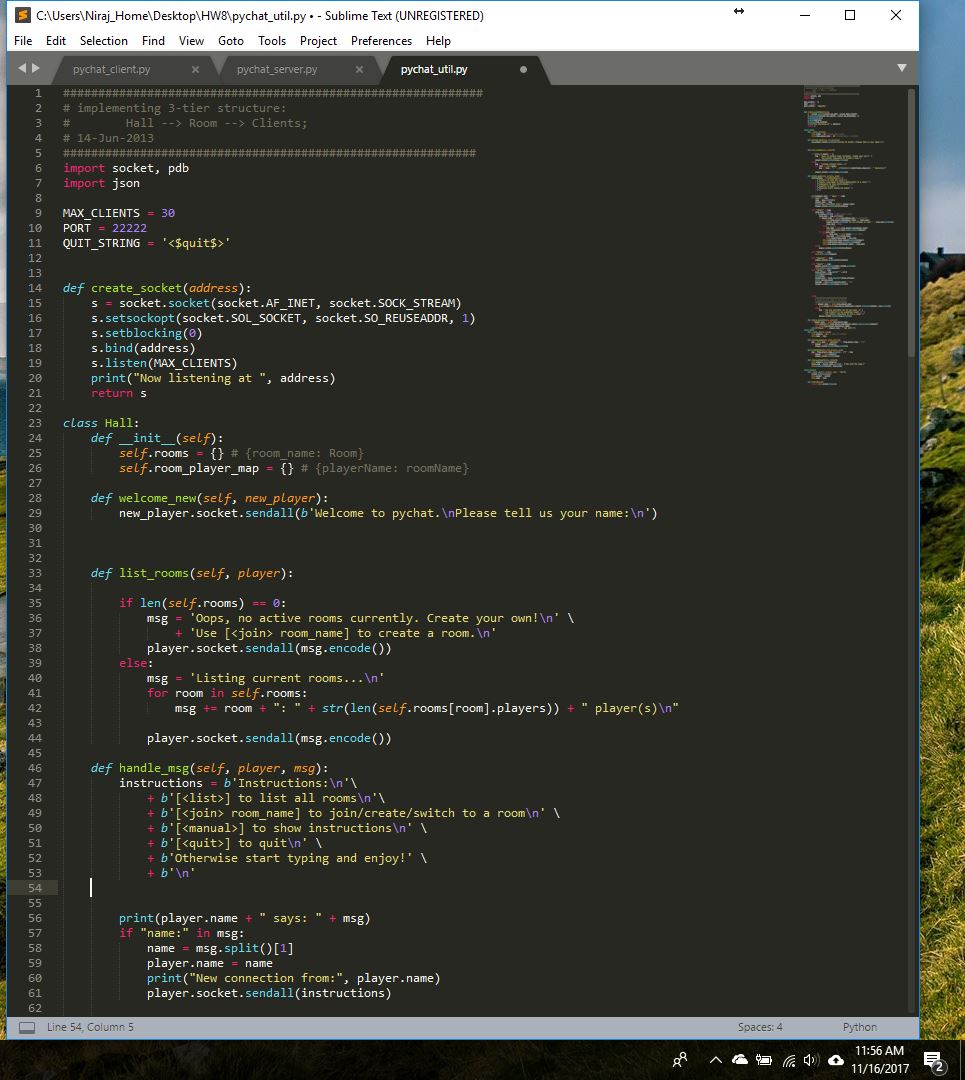
# close error sockets

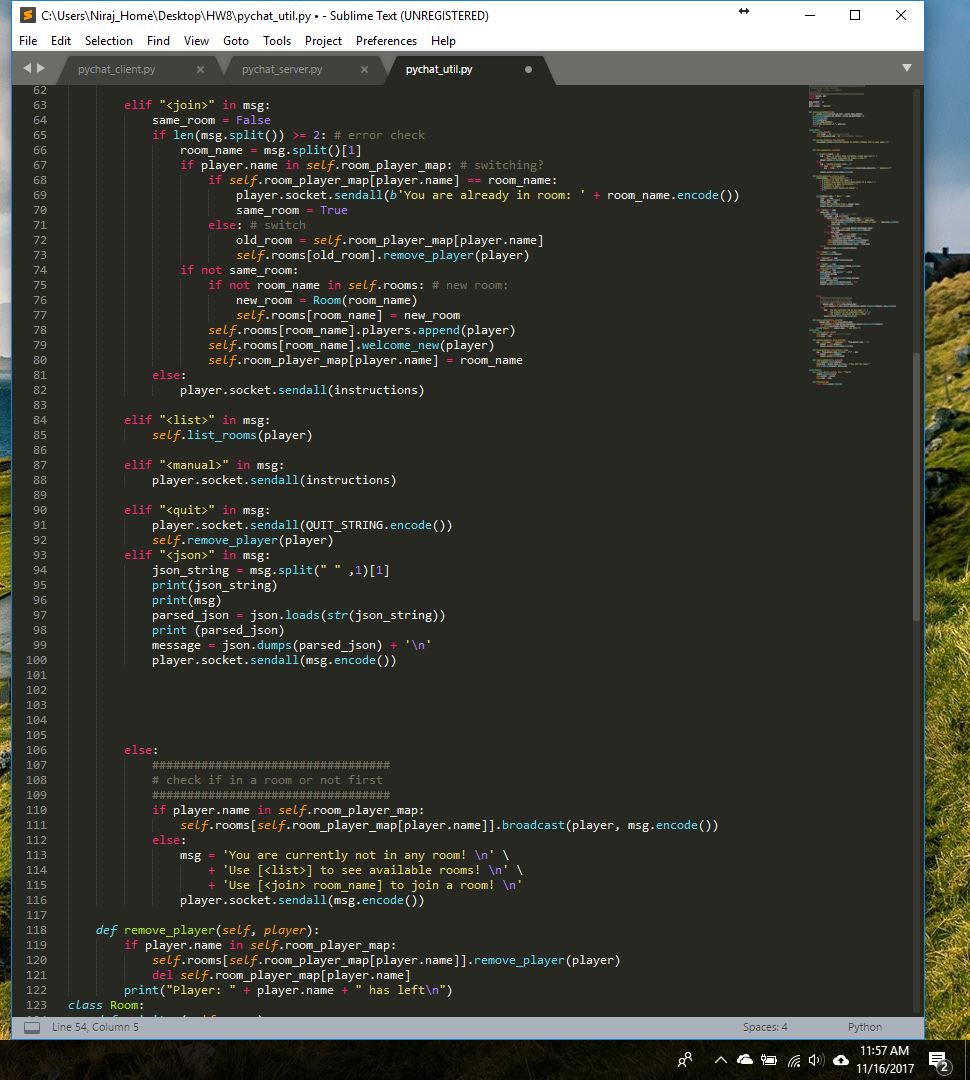
#####################################

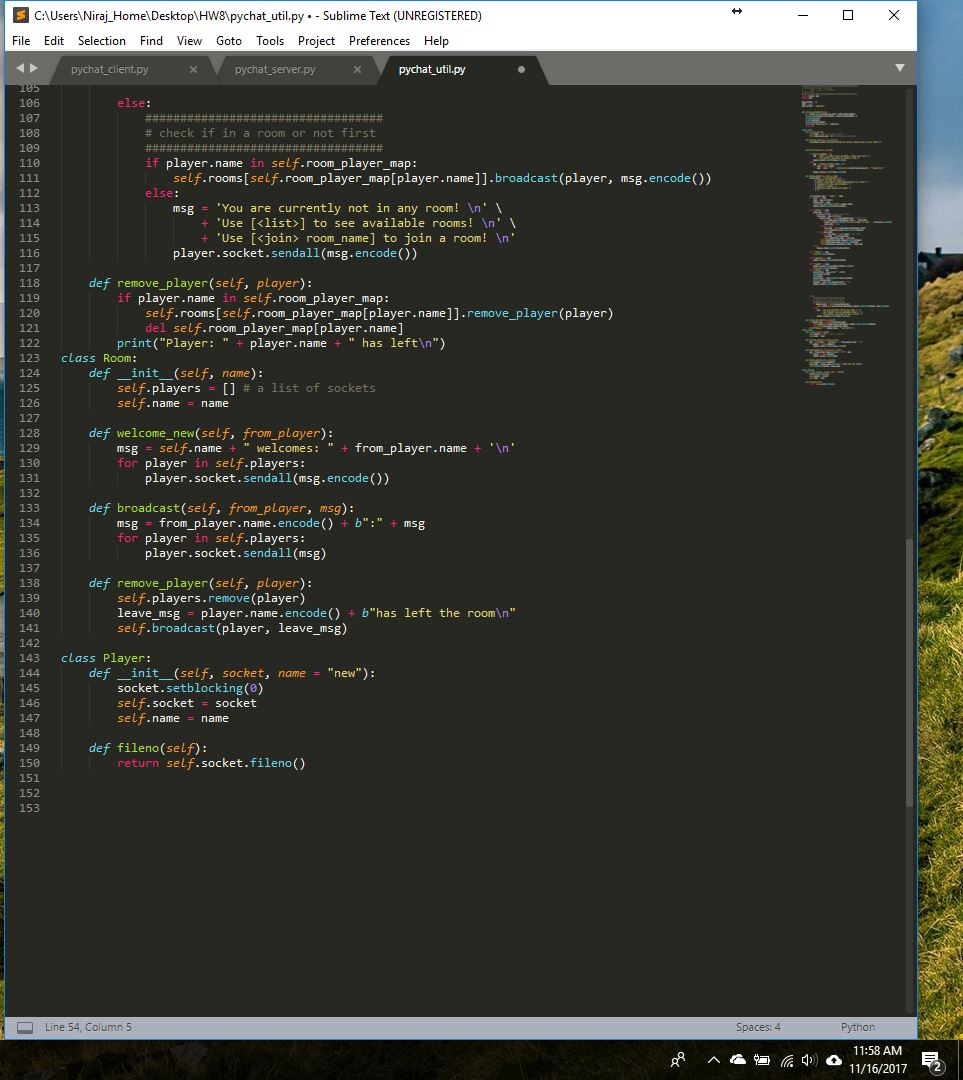
for sock in error\_sockets:

sock.close()

connection\_list.remove(sock)







Source Code:

############################################################

# implementing 3-tier structure:

# Hall --> Room --> Clients;

# 14-Jun-2013

###########################################################

import socket, pdb

import 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'[<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 "<quit>" in msg:

player.socket.sendall(QUIT\_STRING.encode())

self.remove\_player(player)

elif "<json>" in msg:

json\_string = msg.split(" " ,1)[1]

print(json\_string)

print(msg)

parsed\_json = json.loads(str(json\_string))

print (parsed\_json)

message = json.dumps(parsed\_json) + '\n'

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

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()

