**KARNATAK LAW SOCIETY’S**

**GOGTE INSTITUTE OF TECHNOLOGY**

**UDYAMBAG, BELAGAVI – 590008**

**(An Autonomous Institution under Visvesvaraya Technological University, Belagavi)**

**(Approved By AICTE, New Delhi)**

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

**COURSE PROJECT**

**OPERATING SYSTEM**

RACHANA KAMPLI 2GI18IS032

LAXMI NYAMAGOUD 2GI18IS020

HEMANTH I T 2GI18IS015

ROHAN KOKATANUR 2GI18IS066

Guided by:

**Prof. S.G. Sanu**

# Title: Simple Chat Room using Python

1. **Abstract:**

[Python](https://hackernoon.com/tagged/python?ref=hackernoon.com) is one of the most versatile programming languages and one can observe that through its various applications everywhere. Here is an example of how you can build a simple command-line based chat room using python which is easy to implement and understand. So let’s get started.

1. **Problem Statement:**

To create a simple chat room in Python using Network programming.

1. **Objectives:**

# Simple Chat Room System Project

The main objective of the Simple Chat Room project is to create a chat application which helps different users to communicate with each other through a server connected. This is a simple chat program with a server and can have many clients. The server needs to be started first and clients can be connected later. Simple Chat Room provides a bidirectional communication between client and server. It enables user to seamlessly communicate with each other. The user has an option to login to the chat room. The user should be able to give the ip address of the server and the port at which he is connecting. The user can chat using this chat application. The server shall poll for other users that are active in the chat room and make those users visible. If the user at the other end is active then they can start a chat session. The chat is recorded in the application. The user can save the chat transcript or clear it based on his requirement.

**//why**

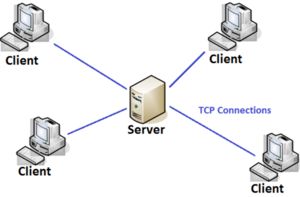
1. **Implementation:**

## ****What is a chat room?****

A chat room is a medium/interface that allows two or more people to chat and send messages to everyone. It can be interpersonal (one-one) and group chat too. In this [tutorial](https://hackernoon.com/tagged/tutorial?ref=hackernoon.com), we’re going to build a group chat room that can host more than two clients at a time.

## ****Architecture****

For chat room, we’re going to use the server-client architecture. It means multiple clients will be hosted by one server.



## **Getting Started:**

Before diving into code, the last thing we want to know is to follow this architecture, we need to write two scripts, one for the server-side and the other one for the client-side. One thing that needs to be very clear is that clients will talk only via server. There will be no direct communication between them. So let’s dive into the code.

//code

## ****Server-side code:****

Firstly, we’ll be importing two libraries named ‘socket’ and ‘threading’. Both are built-in libraries so there’s no need to pip install them. Just importing will work. Socket programming is a way of connecting two nodes on a network to communicate with each other whereas the threading module provides a very simple and intuitive API for spawning multiple threads in a program. Then we’ll move on to define our IP and port. One needs to know that only unreserved ports can be used as this is going to work on LocalHost and the computer may cause problems if you use the reserved ones.

https://hackernoon.com/emojis/heart.png

https://hackernoon.com/emojis/light.png

https://hackernoon.com/emojis/thumbs-down.png

https://hackernoon.com/emojis/money.png

While defining socket, two parameters named (AF\_INET) and (SOCK\_STREAM) will be used. First one indicates the usage of [internet](https://hackernoon.com/tagged/internet?ref=hackernoon.com) socket and the other one indicates the usage of TCP. Then we move on by defining the broadcast function. Its basic function is to send message to the clients in the clients list. This function is not retiring yet. We are going to use it ahead in other places too. Handling the clients is a pretty tough job, so is the function. It first tries if a message can be received from the client’s end, if yes, it is broadcasted.

https://hackernoon.com/emojis/heart.png

https://hackernoon.com/emojis/light.png

https://hackernoon.com/emojis/thumbs-down.png

https://hackernoon.com/emojis/money.png

But if there’s any kind of error/issue, the server keeps things simple. It simply removes the client. We’ve done a lot of work but, adding the clients still awaits. So let’s do that now. In the receive function, the keyword ‘NICKNAME’ is sent to the clients which means their nickname is requested. Later upon obtaining the nickname, it adds the client to the list.

https://hackernoon.com/emojis/heart.png

Well, this loop remains active and multiple clients can join the server. All that you need to have it the right IP and the port address.

Code:

#Coded by Yashraj Singh Chouhan

import socket, threading #Libraries import

host = '127.0.0.1' #LocalHost

port = 7976 #Choosing unreserved port

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) #socket initialization

server.bind((host, port)) #binding host and port to socket

server.listen()

clients = []

nicknames = []

def broadcast(message): #broadcast function declaration

for client in clients:

client.send(message)

def handle(client):

while True:

try: #recieving valid messages from client

message = client.recv(1024)

broadcast(message)

except: #removing clients

index = clients.index(client)

clients.remove(client)

client.close()

nickname = nicknames[index]

broadcast('{} left!'.format(nickname).encode('ascii'))

nicknames.remove(nickname)

break

def receive(): #accepting multiple clients

while True:

client, address = server.accept()

print("Connected with {}".format(str(address)))

client.send('NICKNAME'.encode('ascii'))

nickname = client.recv(1024).decode('ascii')

nicknames.append(nickname)

clients.append(client)

print("Nickname is {}".format(nickname))

broadcast("{} joined!".format(nickname).encode('ascii'))

client.send('Connected to server!'.encode('ascii'))

thread = threading.Thread(target=handle, args=(client,))

thread.start()

receive()

## Client-side code:

This is our second code where we’ll be writing the script for our clients. This code will be enough to get us multiple clients without any problem. So we begin with importing socket and threading. After initializing the socket, we need to connect it to the IP and the port. They need to be the same as the server in order to make it functional.

Now we need to connect to the server and if you remember ‘NICKNAME’ was the keyword sent by the server to ask for a nickname, if client receives that, it sends the nickname and enters the chatroom. But if it fails to receive that keyword, the connection gets lost. Now that we’re connected to the server, why are we waiting, let’s start messaging.

For that, we’ve got the write function that manages the sending of messages. If you wonder what about receiving messages, remember we imported threading. Here it comes into play, we need two threads to begin and that is done.

You’ve noticed that we’ve encoded messages into ASCII before sending, it is because we can send messages only in bytes form and not in string. That’s why always remember, encode before sending and decode after receiving.

code:

#Coded by Yashraj Singh Chouhan

import socket, threading

nickname = input("Choose your nickname: ")

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) #socket initialization

client.connect(('127.0.0.1', 7976)) #connecting client to server

def receive():

while True: #making valid connection

try:

message = client.recv(1024).decode('ascii')

if message == 'NICKNAME':

client.send(nickname.encode('ascii'))

else:

print(message)

except: #case on wrong ip/port details

print("An error occured!")

client.close()

break

def write():

while True: #message layout

message = '{}: {}'.format(nickname, input(''))

client.send(message.encode('ascii'))

receive\_thread = threading.Thread(target=receive) #receiving multiple messages

receive\_thread.start()

write\_thread = threading.Thread(target=write) #sending messages

write\_thread.start()

1. **Working Model Of Final Solution**

