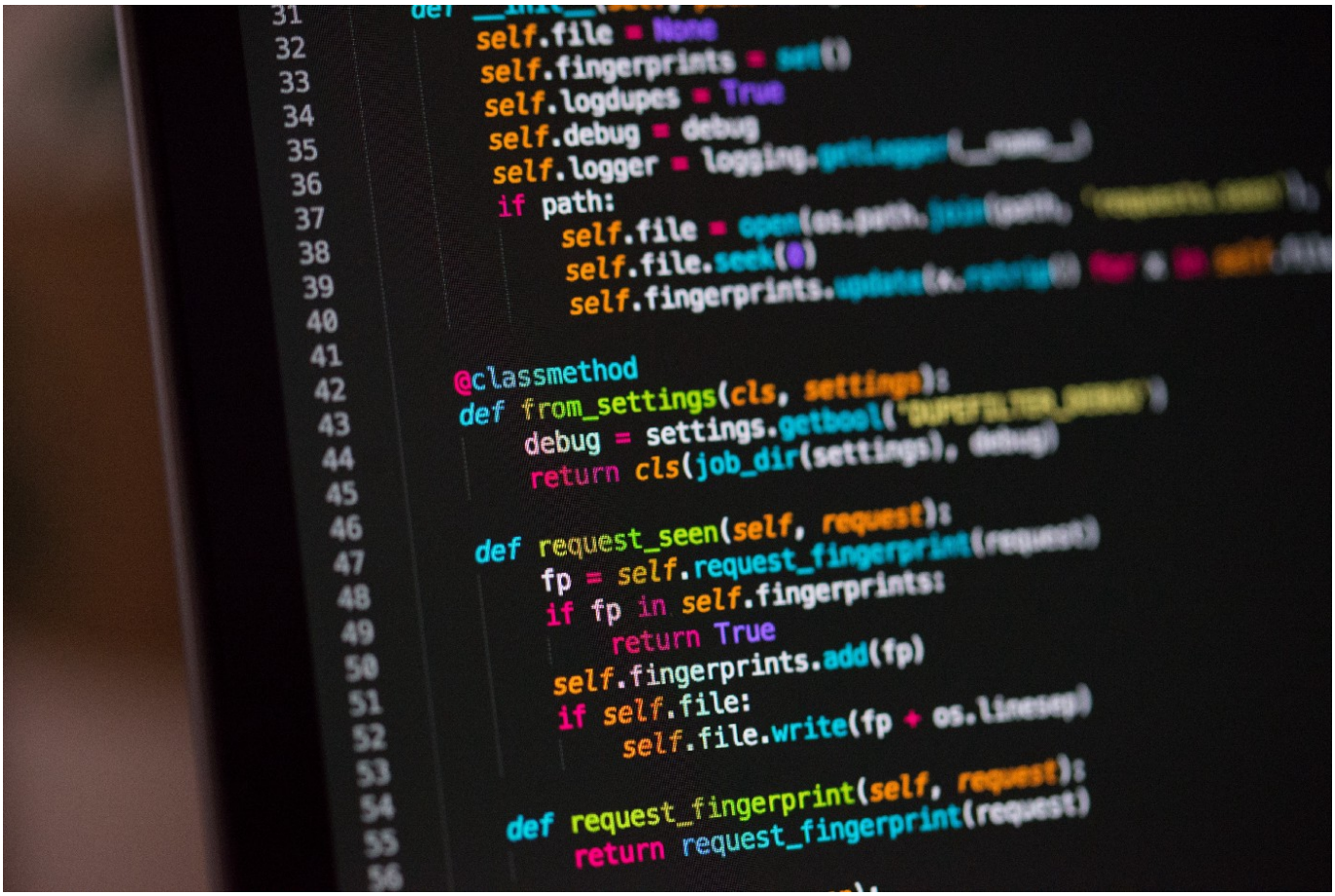
Create a Basic LAN Chat Room with Python3

Maxence LQ Follow

May 18, 2021 · 11 min read ★



Hello everyone, today we are going to create a really simple chat room in Python 3 with the help of the socket and threading built-in modules.

Our chat app will be composed of a server and multiple clients. The clients are going to establish a socket connection with the server. For a socket connection to work, both server and client need to be either on the same machine or in the same network. So are chatroom will work on LAN (Local Area Network).

Understanding Python Sockets

Before jumping into the code, we need to understand how do the sockets work. It is not necessary to know how sockets work internally, it is just interesting to know that the python socket module uses the C socket library to exchange data across the network. Primarily, a socket server needs to be initialized, then the client is going to connect directly to the server. Once the connection is established, a socket can both listen and send messages, if a socket is sending and the other is listening, we will be able to exchange data between them.

Step 1. Coding the server

Once you have created your `server.py` file, we can start coding:

First, we will import the module we need:

```
import socket
```

Then let's create our socket object:

```
my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

`socket.AF_INET` and `socket.SOCK_STREAM` are just constants defined in the socket module to choose the type of socket we want, you don't need to understand what they mean to use them.

Now the socket is created, we can bind it to a port and optionally an address. We create two constants called `PORT` and `ADDRESS` to hold the port and the address we are going to use. We let the address to `"0.0.0.0"` for now, since we don't know what the client's address is going to be.

```
PORT = 8000
ADDRESS = "0.0.0.0"
my_socket.bind((ADDRESS, PORT))
```

Now our socket is ready, we can start listening for a connection. Then we need to accept the incoming connection.

```
my_socket.listen()
client, client_address = my_socket.accept()
```

The listen() method just allows incoming connections. The accept() method return a tuple, the first element is the connection, we will now use it to interact with the client and the second one is a tuple containing the IP address of the client as well as the port used (it can be different from the port you set up).

We are now ready to start listening to messages from the client!

```
message = client.recv(1024) #bytes
print(message.decode())
```

The recv() method takes only one non-optional argument, the buffer size, that is the max space available to receive the message. Don't forget that this method returns a byte object and so you need to decode it into a string.

OK, so now we should be done with the most basic server script possible, and the client script is even more simple you will see. Just before jumping to the client code, let's make sure we have the same server code, if you well followed this tutorial, your code should look something like that:

```
#server.py
import socket

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

PORT = 8000
ADDRESS = "0.0.0.0"
my_socket.bind((ADDRESS, PORT))

my_socket.listen()
client, client_address = my_socket.accept()
```

```
result = client.recv(1024)
print(result.decode())
```

Step 2. Coding the client

Client script is almost the same as the server script, except instead of listening for connection and message, we are actually going to connect and send messages.

First, import the module and create our socket object:

```
import socket

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Now we need to define the port and the address of the server, if you are using the same port as I do, it should be `8000`. For the address you just have to enter the address of the server, if you are running the server and the client on the same computer, the server address should be `localhost`, else just find the server's IP address (I'm not going to explain how to do it here but you should be able to find the information very easily).

```
PORT = 8000
ADDRESS = "localhost" # Same as "127.0.1.1"
```

Then we can connect to the server:

```
my_socket.connect((ADDRESS, PORT)
```

We are now ready to send messages! Just type in:

```
my_socket.send("The message you want to send".encode())
```

Don’t forget to encode the string into bytes, as we saw previously, sockets exchange bytes, not strings.

So now your client script should look something like that:

```
#client.py
import socket

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

host = "localhost" # "127.0.1.1"
port = 8000

my_socket.connect((host, port))

my_socket.send("hello".encode())
```

You should now be able to use the script, start the server script, and once it’s started launch the client script, if everything has gone well, you should see the message you chose, displayed in the console server’s console.

Improving our basic demo

Now you have probably tested our small demo, and maybe you're telling yourself that this doesn’t look like a chatroom. And as you are saying, this was just a demo to understand sockets, now we can really implement our little chatroom.

First, since this is a chat, we will probably need to send multiple messages, not just only one, so we are going to edit our server script to listen forever, and in the client, we are going to allow the user to enter some message to send to the server.

```
#server.py
import socket

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

PORT = 8000
ADDRESS = "0.0.0.0"
my_socket.bind((ADDRESS, PORT))
```

```
my_socket.listen()
client, client_address = my_socket.accept()

# Instead of receiving only one message, let's make an infinite loop
while True:
    result = client.recv(1024)
    print(result.decode())
```

As you can see, I have added an infinite loop to never stop listening to messages. Now let's work on the client script:

```
#client.py
import socket

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

host = "localhost" # "127.0.1.1"
port = 8000

my_socket.connect((host, port))

while True:
    message_to_send = input("Enter your message : ")
    my_socket.send(message_to_send.encode())
```

And here as you can see the user can now type in the message he wants to send and I have added an infinite loop here as well. I encourage you to try this code to see that it works.

Step 3. Allow our client to receive

OK, now we can send multiple messages to the server, but what about receiving messages from the server? This is what we are going to work on in this part.

So we will need to both listen for user input and listen for server messages at the same time, how could we do that?

One of the simplest solutions is to use the python threading built-in module, with that we will be able to run multiple threads at the same time, I'm not going to enter too much into the details here, just see the threads as workers that do their work in the

background but can interact with the python main program. If you don't know how threads work, I encourage you to find more information about them since it is something really useful in networking and programming in general.

Now we know how to solve the problem, we need the threading module, so let's import it:

```
import threading
```

We need now to define two functions, one to listen, and one to send. For the sending part, we just need to put our while loop into a function, something like this:

```
def thread_sending():
    while True:
        message_to_send = input()
        my_socket.send(message_to_send.encode())
```

And the listen function is not going to be more complicated:

```
def thread_receiving():
    while True:
        message = my_socket.recv(1024).decode()
        print(message)
```

Once we have our two functions, we can now create two threads, one for each function, and start them. Threads are really simple to use, we just need to create two threads objects and specify what function they will execute:

```
thread_send = threading.Thread(target=thread_sending)
thread_receive = threading.Thread(target=thread_receiving)
```

We just need to start them and that's all for the client for now:

```
thread_send.start()
thread_receive.start()
```

You can try to run the client script and you won't see any difference, it's because we haven't made the server send any message, so let's work on that. For simplicity, we are just going to make the server send something each time a message is received, something like that:

```
while True:
    result = client.recv(1024)
    print(result.decode())
    client.send("Message received !".encode())
```

You should now be able to send a message and receive a response from the server. As always, before starting the next step, let's make sure we work with the same code:

```
#server.py
import socket

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

PORT = 8000
ADDRESS = "0.0.0.0"

my_socket.bind((ADDRESS, PORT))

my_socket.listen()
client, client_address = my_socket.accept()

# Instead of receiving only one message, let's make an infinite loop
while True:
    result = client.recv(1024)
    print(result.decode())
    client.send("Message received !".encode())
```

And the client:

```
#client.py
import socket
import threading

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = "localhost" # "127.0.1.1"
port = 8000
my_socket.connect((host, port))

def thread_sending():
    while True:
        message_to_send = input()
        my_socket.send(message_to_send.encode())

def thread_receiving():
    while True:
        message = my_socket.recv(1024).decode()
        print(message)

thread_send = threading.Thread(target=thread_sending)
thread_receive = threading.Thread(target=thread_receiving)
thread_send.start()
thread_receive.start()
```

Step 4: Adding more than one client

OK, so now we can connect to a server using a client, we can send and receive messages from the server, we are almost done, but, isn't a chat composed of several people? Yeah of course it is, we need now to be able to connect more than one client to a chat, and that's what we are going to work on in this part.

So at first, we could just think that to create a chat, we just need to start multiple clients, but after testing this method, you will understand that in a chat, the client needs to interact with other clients, not just only interact with the server. To solve this issue, we will need to keep a list of all the clients connected, and every time someone sends a message, we will send this message to all the clients in the list. Our server will transmit messages from one client to others.

The first step is to be able to handle multiple connections, as you can see in the code, accept() method is only called once, which mean we can only receive one connection, let's change that, and we are going to write the code in functions to be able to use threads:

```
def thread_accept():
    while True:
        my_socket.listen()
        client, client_address = my_socket.accept()
```

Now we can add the client to a broadcast list, we need first to create the list (out of the function):

```
broadcast_list = []
```

And append the client to the list (in the loop):

```
broadcast_list.append(client)
```

And the last thing, we need to set up a thread that is going to listen for messages from the client:

```
start_listenning_thread(client)
```

As you can see we have not coded this function yet, we are going to do that in one minute, let's before make sure we have the same function:

```
def accept_loop():
    while True:
        my_socket.listen()
        client, client_address = my_socket.accept()
        broadcast.append(client)
        start_listenning_thread(client)
```

OK, now we are sure to have the same code, so let's create the `start_listenning_thread` function. This function needs to create a dedicated thread for this client that is going to listen for message and broadcast them:

```
def start_listenning_thread(client):
    client_thread = threading.Thread(
            target = listen_thread,
            args = (client,) #the list of argument for the function
        )
    client_thread.start()
```

We need now to implement our `listen_thread` function:

```
def listen_thread(client):
    while True:
        message = client.recv(1024).decode()
        print(f"Received message : {message}")
        broadcast(message)
```

And the last function to write, `broadcast` will just send the message to every client in the broadcast list:

```
def broadcast(message):
    for client in broadcast:
        client.send(message.encode())
```

We can now just run a function to start accepting connections. Now your code should look like that:

```
#server.py
import socket

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

PORT = 8000
ADDRESS = "0.0.0.0"

broadcast_list = []

my_socket.bind((ADDRESS, PORT))
```

```
def accept_loop():
    while True:
        my_socket.listen()
        client, client_address = my_socket.accept()
        broadcast_list.append(client)
        start_listenning_thread(client)

def start_listenning_thread(client):
    client_thread = threading.Thread(
            target=listen_thread,
            args=(client,) #the list of argument for the function
        )
    client_thread.start()

def listen_thread(client):
    while True:
        message = client.recv(1024).decode()
        print(f"Received message : {message}")
        broadcast(message)

def broadcast(message):
    for client in broadcast_list:
        client.send(message.encode())

accept_loop()
```

If you have coded everything right, you should now be able to start the server and connect several clients to it, send messages, etc...

Step 5. A few issues to solve

If you have tested the chat, you should have noticed a few issues, first, when a message is sent, we have no way to know who sent the message, and second, (this one is less visible) but we have created no way handle the stop of the connection between a client and the server, we will dive into it later.

Add nicknames to our app

So we have two options to implement nicknames, we can either implement this on the server-side or on the client-side. For simplicity here we are going to choose the second option. (Keep in mind our chatroom is not secured at all, there are many exploits but we don't care about that.)

The client script will need to ask the user to choose a nickname, and then send the nickname with the message. This is not very complicated:

```
nickname = input("Choose your nickname : ").strip()
```

And we can verify the nickname is not empty:

```
while not nickname:
    nickname = input("Your nickname should not be empty : ").strip()
```

And now we have the user's nickname, we can send it with the message, our `thread_sending` function should now look like this:

```
def thread_sending():
    while True:
        message_to_send = input()
        message_with_nickname = nickname + " : " + message_to_send
        my_socket.send(message_with_nickname.encode())
```

And now, users have a nickname, it was not that complicated. Your `client.py` should look like this:

```
#client.py
import socket
import threading

nickname = input("Choose your nickname : ").strip()
while not nickname:
    nickname = input("Your nickname should not be empty : ").strip()

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = "localhost" # "127.0.1.1"
port = 8000
my_socket.connect((host, port))

def thread_sending():
    while True:
        message_to_send = input()
        message_with_nickname = nickname + " : " + message_to_send
        my_socket.send(message_with_nickname.encode())
```

```
def thread_receiving():
    while True:
        message = my_socket.recv(1024).decode()
        print(message)

thread_send = threading.Thread(target=thread_sending)
thread_receive = threading.Thread(target=thread_receiving)
thread_send.start()
thread_receive.start()
```

The server should now handle if the client disconnect

You may have noticed that when a client disconnect (using Ctrl+C), the server get a very strange behavior where it never stops printing `received message :`.

So first, let's make sure the client send the message only if it not empty:

```
message_to_send = input()
if message_to_send:
      message_with_nickname = nickname + " : " + message_to_send
      my_socket.send(message_with_nickname.encode())
```

And that the server only prints the message if it is not empty, if it is, it means that the client has been disconnected, so we can stop the listening thread (to stop the thread we can just exit out of the function):

```
message = client.recv(1024).decode()
if message:
    print(f"Received message : {message}")
    broadcast(message)
else:
    return
```

Now your code should look like this:

```
#server.py
import socket
import threading
```

```
my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

PORT = 8000
ADDRESS = "0.0.0.0"

broadcast_list = []

my_socket.bind((ADDRESS, PORT))

def accept_loop():
    while True:
        my_socket.listen()
        client, client_address = my_socket.accept()
        broadcast_list.append(client)
        start_listenning_thread(client)

def start_listenning_thread(client):
    client_thread = threading.Thread(
            target=listen_thread,
            args=(client,) #the list of argument for the function
        )
    client_thread.start()

def listen_thread(client):
    while True:
        message = client.recv(1024).decode()
        if message:
            print(f"Received message : {message}")
            broadcast(message)
        else:
            print(f"client has been disconnected : {client}")
            return

def broadcast(message):
    for client in broadcast_list:
        try:
            client.send(message.encode())
        except:
            broadcast_list.remove(client)
            print(f"Client removed : {client}")

accept_loop()
```

And the client script:

```
#client.py
import socket
```

```
import threading

nickname = input("Choose your nickname : ").strip()
while not nickname:
    nickname = input("Your nickname should not be empty : ").strip()

my_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = "localhost" # "127.0.1.1"
port = 8000
my_socket.connect((host, port))

def thread_sending():
    while True:
        message_to_send = input()
        if message_to_send:
            message_with_nickname = nickname + " : " +
message_to_send
            my_socket.send(message_with_nickname.encode())

def thread_receiving():
    while True:
        message = my_socket.recv(1024).decode()
        print(message)

thread_send = threading.Thread(target=thread_sending)
thread_receive = threading.Thread(target=thread_receiving)
thread_send.start()
thread_receive.start()
```

If your code looks like that, the chatroom shall be functional. There is obviously plenty of things you can add to it but this tutorial was just about building a very simple chat.

More content at ***plainenglish.io***

Python Programming Software Development Python Programming Coding