

Documentação do código

client.py

```
import threading
import socket
```

Iniciamos o código importando os módulos acima. Fizemos uso do módulo threading para podermos criar e gerenciar da melhor forma as threads, enquanto o módulo socket foi utilizado para a comunicação via sockets.

```
def main():
```

Definimos a função main, que é responsável por toda a configuração inicial do cliente, também sendo o ponto de entrada do chat.

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

Esta linha de código deverá criar um objeto de socket chamado cliente.

Um socket é usado para enviar dados através de uma rede. Desta forma, esse objeto será usado para estabelecer uma conexão com o servidor.

O parâmetro socket.AF_INET indicará que o tipo de socket é para IPv4, que é a quarta e mais difundida versão do protocolo IP, com endereços no padrão 32 bits.

O socket.SOCK_STREAM indica que o socket será orientado a conexão TCP, que é um padrão de comunicação onde é permitido que dispositivos de comunicação, aplicativos e programas troquem mensagens em uma rede, usando informações de endereço.

```
try:
```

```
    cliente.connect(('localhost', 10101))
```

```
    print("Cliente conectado")
```

```
except:
```

```
    return print('\nNão foi possível conectar-se ao servidor.\n')
```

Neste trecho de código, o connect() permitirá que o cliente estabeleça conexão com o servidor. Desta forma, conectando-se ao servidor local (localhost), na porta 10101. Caso ocorra algum erro durante a tentativa de conexão, uma mensagem de erro será exibida.

```
username = input("Usuário: ")
```

```
print("\nConectado.")
```

O programa solicitará ao usuário que digite um nome para cadastrar, o mesmo será armazenado na variável username. Em seguida, exibirá uma mensagem informando que a conexão foi bem-sucedida.

```
thread1 = threading.Thread(target=recvMessages, args=[cliente])
```

```
thread2 = threading.Thread(target=sendMessages, args=[cliente, username])
```

```
thread1.start()
```

```
thread2.start()
```

Neste trecho duas threads são criadas usando o threading. A thread1 será responsável por receber mensagens do servidor, enquanto a thread2 fará o envio delas.

Atribuímos a função `recvMessages()` para a thread1, e o objeto de socket do (cliente) passará como argumento. A função `sendMessages()` é atribuída para a thread2, enquanto o objeto de socket do (cliente) e o (username) serão passados como argumentos.

O método `start()` iniciará as threads.

```
def recvMessages(cliente):  
    while True:  
        try:  
            msg = cliente.recv(2048).decode('utf-8')  
            print(msg+'\n')  
        except:  
            print('\nNão foi possível manter-se conectado.\n')  
            print('Pressione <ENTER> para continuar.')  
            cliente.close()  
            break
```

No trecho acima haverá a execução de uma thread separada para que possa receber mensagens do servidor.

No loop `while True`, o cliente deverá ficar aguardando a recepção de mensagens contínuas. Ainda dentro desse loop, o cliente receberá uma mensagem do servidor usando o método `recv()` do objeto de socket do cliente (`cliente.recv(2048)`).

O número 2048 indica o tamanho máximo dos dados que poderão ser recebidos pelo servidor.

O método `decode('utf-8')` deverá decodificar de bytes para string. Após decodificar, a função exibirá na tela e pulará uma linha. Caso haja alguma exceção durante a recepção de mensagens, significará que a conexão com o servidor foi perdida. Então uma mensagem de erro será exibida, o socket do cliente é fechado (`cliente.close()`), e o loop é interrompido com o `break`.

```
def sendMessages(cliente, username):  
    while True:  
        try:  
            msg = input('\n')  
            cliente.send(f'<{username}> <{msg}>'.encode('utf-8'))  
        except:  
            return
```

Assim como no trecho acima a thread será executada separadamente, mas com o intuito de enviar mensagens para o servidor.

O loop `while true` permitirá que o cliente envie mensagens. Dentro desse mesmo loop, o programa solicitará ao usuário que digite uma mensagem usando a função `input()`, que ficará salva na variável `msg`.

O cliente enviará a mensagem para o servidor usando o método `send()` do objeto de socket do cliente (`cliente.send()`). Antes da mensagem ser enviada, o (username) e a (msg) são formatados em string e codificados em bytes usando o método `encode('utf-8')`.

Se de algum modo ocorrer uma exceção, a função retorna, encerrando a thread.

```
main()
```

A função main() é chamada para iniciar o cliente.

Por fim, concluímos que o código criará um cliente que estabelecerá uma conexão com o servidor, onde poderá digitar mensagens para enviar e exibir as recebidas também. Para garantir uma comunicação simultânea com o servidor, as operações de envio e recebimento de mensagens são feitas em threads separadas.

server.py

```
import threading
```

```
import socket
```

Estas linhas de código importarão os módulos threading e socket, que realizarão operações relacionadas a threads e estabelecerão comunicação por sockets.

```
clientes = []
```

Esta variável armazenará as conexões dos clientes que irão se conectar ao servidor.

Definindo a função principal main():

```
def main():
```

Nesta função ocorre a configuração inicial do servidor.

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

Esta linha criará um objeto (socket) chamado server, que aceitará conexões de clientes.

O socket.AF_INET indicará que o tipo de socket é para IPv4 que é a quarta e mais difundida versão do protocolo IP, com endereços no padrão 32 bits. Enquanto o socket.SOCK_STREAM indicará que o socket será orientado a conexão TCP.

```
try:
```

```
    server.bind(('localhost', 10101))
```

```
    server.listen(2)
```

```
    print("Servidor conectado")
```

```
except:
```

```
    return print("\nNão foi possível iniciar o servidor.\n")
```

Neste trecho o servidor tentará vincular o socket a um endereço IP e porta específica usando o método bind().

O servidor está vinculado ao endereço localhost e à porta 10101. Em seguida, o servidor recebe até 2 conexões simultâneas (server.listen(2)). Se ocorrer algum problema, uma mensagem de erro será exibida.

```
while True:
```

```
    cliente, addr = server.accept()
```

```
    clientes.append(cliente)
```

```
thread = threading.Thread(target=messagesTreatement, args=[cliente])
```

```
thread.start()
```

Neste pedaço de código, o servidor entrará em loop infinito para esperar conexões de clientes. Quando conectar, o método `accept()` é chamado no objeto de socket do servidor (`server.accept()`), retornando um novo objeto (cliente) e o (`addr`).

O objeto cliente será adicionado à lista de clientes (`clientes.append(cliente)`).

Sendo assim, uma nova thread é criada usando o módulo `threading`, que tratará as mensagens recebidas do cliente. A função `messagesTreatement()` será atribuída como o alvo da thread e o objeto cliente é passado como argumento (`args=[cliente]`). Por fim, uma thread é iniciada (`thread.start()`).

```
def messagesTreatement(cliente):  
    while True:  
        try:  
            msg = cliente.recv(2048)  
            broadcast(msg, cliente)  
        except:  
            deleteClient(cliente)  
            break
```

Neste trecho, cada thread será executada separadamente para cada cliente.

O loop `while True` permitirá que o servidor fique aguardando mensagens do cliente continuamente. Dentro do loop, o servidor tentará receber uma mensagem do cliente usando o `recv()` do objeto de socket do cliente (`cliente.recv(2048)`).

O número 2048 indica o tamanho máximo de dados que poderão ser recebidos pelo servidor.

Após receber uma mensagem, o `broadcast()` enviará uma mensagem recebida para todos os outros clientes conectados.

Se houver alguma restrição durante a recepção da mensagem, significará que a conexão foi perdida. Desta forma, a função `deleteClient()` removerá o cliente da lista de clientes e encerrará a thread.

```
def broadcast(msg, cliente):  
    for clienteItem in clientes:  
        if clienteItem != cliente:  
            try:  
                clienteItem.send(msg)  
            except:  
                deleteClient(clienteItem)
```

Essa função percorrerá a lista de clientes e enviará a mensagem (`msg`) para todos, com exceção daquele que enviará a mensagem. Para isso, o (`clienteItem`) verificará se o cliente atual é diferente do cliente que envia a mensagem (`cliente`). Caso seja diferente, a mensagem será enviada para esse cliente usando o `send()` do objeto de socket do cliente (`clienteItem.send(msg)`).

Caso ocorra alguma exceção durante o envio, significa que a conexão falhou, então será chamada a função `deleteClient()` para remover o cliente da lista de clientes.

```
def deleteClient(cliente):  
    clientes.remove(cliente)
```

Esta função removerá o cliente da lista de clientes usando o método `remove()`.

```
main()
```

A função `main()` é chamada para iniciar o servidor.

Diante disso, o código criará um servidor de chat básico que aceita conexões de clientes, onde tratará mensagens recebidas de clientes e enviará para todos os outros clientes conectados (broadcast). Cada cliente é tratado em uma thread separada para permitir a comunicação simultânea com vários clientes.