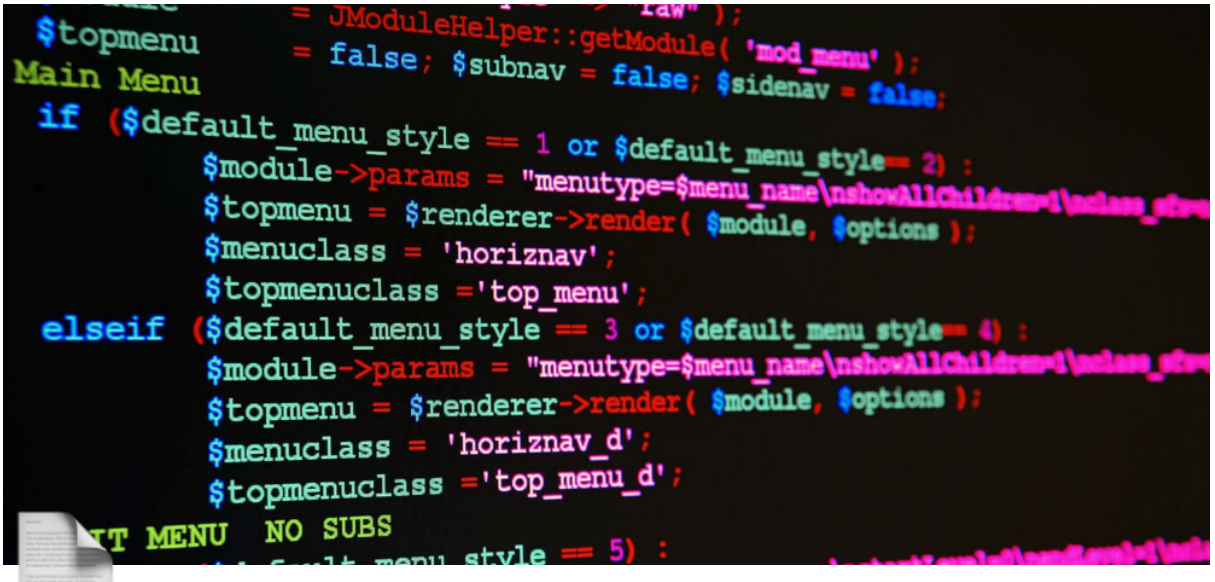

Códigos - Socket TCP e UDP

estabelecer uma conexão TCP com um servidor

- `endereço`: Este é um parâmetro obrigatório e deve ser uma tupla de dois elementos contendo o nome do host ou o endereço IP e o número da porta do serviço ao qual você deseja se conectar.
- `timeout`: Este é um parâmetro opcional que define o tempo limite em segundos para a tentativa de conexão. Se não for especificado, será utilizado o valor padrão global de tempo limite retornado por `socket.getdefaulttimeout()`.
- `endereço_fonte`: Este é um parâmetro opcional que especifica o endereço de origem a ser utilizado para o soquete antes de tentar estabelecer a conexão. Se não especificado, o comportamento padrão do sistema operacional será utilizado.
- `todos_erros`: Este é um parâmetro opcional que determina se todas as exceções que ocorrerem durante as tentativas de conexão serão coletadas e retornadas. Por padrão, apenas a exceção do último endereço da lista de endereços será retornada.

```
import socket

try:
    # Define o endereço do servidor e a porta
    endereco_servidor = ('localhost', 8080)

    # Estabelece a conexão com o servidor
    client_socket = socket.create_connection(endereco_servido

    # Se a conexão for bem-sucedida, envia uma mensagem ao se
    mensagem = "Olá, servidor!"
    client_socket.sendall(mensagem.encode())

    # Recebe a resposta do servidor
    resposta = client_socket.recv(1024)
    print("Resposta do servidor:", resposta.decode())

    # Fecha a conexão com o servidor
    client_socket.close()

except socket.error as err:
    # Trata possíveis erros de conexão
    print("Erro ao conectar ao servidor:", err)
```

Cliente e Server UDP

```
##---------CLIENTE---------##


from socket import *  # Importa todas as funções necessárias

serverName = 'localhost'  # Endereço do servidor
serverPort = 12000  # Porta do servidor
```

```
# Criação do socket do cliente para comunicação via UDP
clientSocket = socket(AF_INET, SOCK_DGRAM)

# Solicitação de entrada ao usuário
message = input("Input lowercase sentence:")  # Solicita ao u

# Envio da mensagem para o servidor
# A função sendto() é usada para enviar a mensagem para o end
clientSocket.sendto(message.encode("utf-8"), (serverName, ser

# Recebimento da mensagem modificada do servidor
# A função recvfrom() é usada para receber dados do servidor
# Retorna uma tupla contendo a mensagem e o endereço do servi
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)

# Exibição da mensagem modificada recebida do servidor
print(modifiedMessage.decode("utf-8"))

# Fechamento do socket do cliente
clientSocket.close()  # Libera os recursos do socket após a c
```

```
##---------SERVER---------##


from socket import *  # Importa todas as funções necessárias

serverPort = 12000  # Porta em que o servidor irá escutar por

# Criação do socket do servidor para comunicação via UDP
serverSocket = socket(AF_INET, SOCK_DGRAM)

# Associação do socket a um endereço e porta específicos
serverSocket.bind(('', serverPort))
```

```
print("The server is ready to receive")  # Indica que o servi

while True:  # Loop infinito para receber mensagens continuam
    # Recebe uma mensagem e o endereço do cliente que a envio
    message, clientAddress = serverSocket.recvfrom(2048)

    # Decodifica a mensagem recebida para UTF-8 e converte pa
    modifiedMessage = message.decode("utf-8").upper()

    # Envia a mensagem modificada de volta para o cliente
    # Utiliza o endereço do cliente recebido anteriormente pa
    serverSocket.sendto(modifiedMessage.encode("utf-8"), clie
```

CODIFICAÇÃO DADOS

- Python lidam com dados como sequências de bytes, não como strings diretamente
- Quando você envia dados por meio de um socket, eles precisam ser convertidos em uma representação de bytes para serem transmitidos pela rede.

```
import socket

# Criando um socket TCP/IP
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STR

# Endereço e porta do servidor
server_address = ('localhost', 8080)

# Conectando ao servidor
client_socket.connect(server_address)

# Valores de x e y que queremos enviar
x = 42
y = 77
```

```
# Construindo a mensagem como uma string
message = f"{x} {y}"

# Codificando a mensagem em bytes usando a codificação ASCII
encoded_message = message.encode('ascii')

# Enviando a mensagem codificada para o servidor
client_socket.sendall(encoded_message)

# Fechando o socket
client_socket.close()
```

Problema no armazenamento de endereços IP e portas como inteiros nas estruturas de dados usadas em programação de rede

- Por exemplo, em uma estrutura `sockaddr_in`, a porta é armazenada como um inteiro de 16 bits (`u_short sin_port`) e o endereço IP é armazenado como um inteiro de 32 bits (`in_addr sin_addr`).
- Alguns sistemas usam a ordem de bytes big-endian, onde os bytes mais significativos (mais à esquerda) são armazenados primeiro, enquanto outros sistemas usam a ordem de bytes little-endian, onde os bytes menos significativos (mais à esquerda) são armazenados primeiro.
- Isso pode criar problemas quando duas máquinas com diferentes ordens de bytes tentam se comunicar pela rede. Por exemplo, se um endereço IP for representado como `128.119.40.12` (big-endian) e for interpretado por uma máquina little-endian, ele pode ser interpretado incorretamente como `12.40.119.128`. Isso pode causar falhas na comunicação.

```
import socket

# Criando um socket UDP para o cliente
client_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM

# Configurando a porta do cliente
```

```
client_port = 12345  # Exemplo de porta

# Configurando a mensagem a ser enviada
message = b"Hello, server!"  # Mensagem em bytes

# Convertendo a porta do cliente para a ordem de bytes da red
client_port = socket.htons(client_port)

# Enviando a mensagem para o servidor
server_address = ('localhost', 1111)  # Endereço e porta do s
client_sock.sendto(message, (server_address[0], server_addres

# Fechando o socket do cliente
client_sock.close()
```

```
import socket

# Criando um socket UDP para o servidor
server_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM

# Configurando a porta do servidor
server_port = 1111  # Porta do servidor

# Configurando o endereço do servidor
server_address = ('', server_port)  # Endereço do servidor (v

# Vinculando o socket do servidor ao endereço e porta
server_sock.bind(server_address)

# Esperando por uma mensagem do cliente
message, client_address = server_sock.recvfrom(1024)  # Taman

# Convertendo a porta do cliente de volta para a ordem de byt
client_port = socket.ntohs(client_address[1])

# Exibindo a porta do cliente
print(f"Porta do cliente: {client_port}")
```

```
# Fechando o socket do servidor
server_sock.close()
```

Lendo um Arquivo de Texto:

- Para abrir e ler um arquivo de texto em Python, usamos a função `open()` e o método `read()`.
- Se o arquivo não for encontrado, capturamos a exceção `IOError`.
- Aqui está um exemplo:

```
filename = 'foo.txt'
try:
    inputfile = open(filename, 'r')
except IOError:
    print('Arquivo não encontrado')
contents = inputfile.read()
print(contents)
```

Dividindo Linhas de uma Mensagem:

- Quando recebemos dados de um soquete, recebemos um grande bloco de bytes.
- Precisamos converter esses bytes em uma string e, em seguida, dividi-los em linhas.

```
message = 'hello there'
parts = message.split(' ')
print('Primeira parte: ' + parts[0])
print('Segunda parte: ' + parts[1])
```

Além disso, algumas bibliotecas úteis para o projeto incluem `socket` (essencial), `time` ou `datetime` (para datas), `re` (expressões regulares),

`os.path` e/ou `os.stat` (para informações de arquivo). Certifique-se de obter aprovação para usar outras bibliotecas ou utilitários além desses.