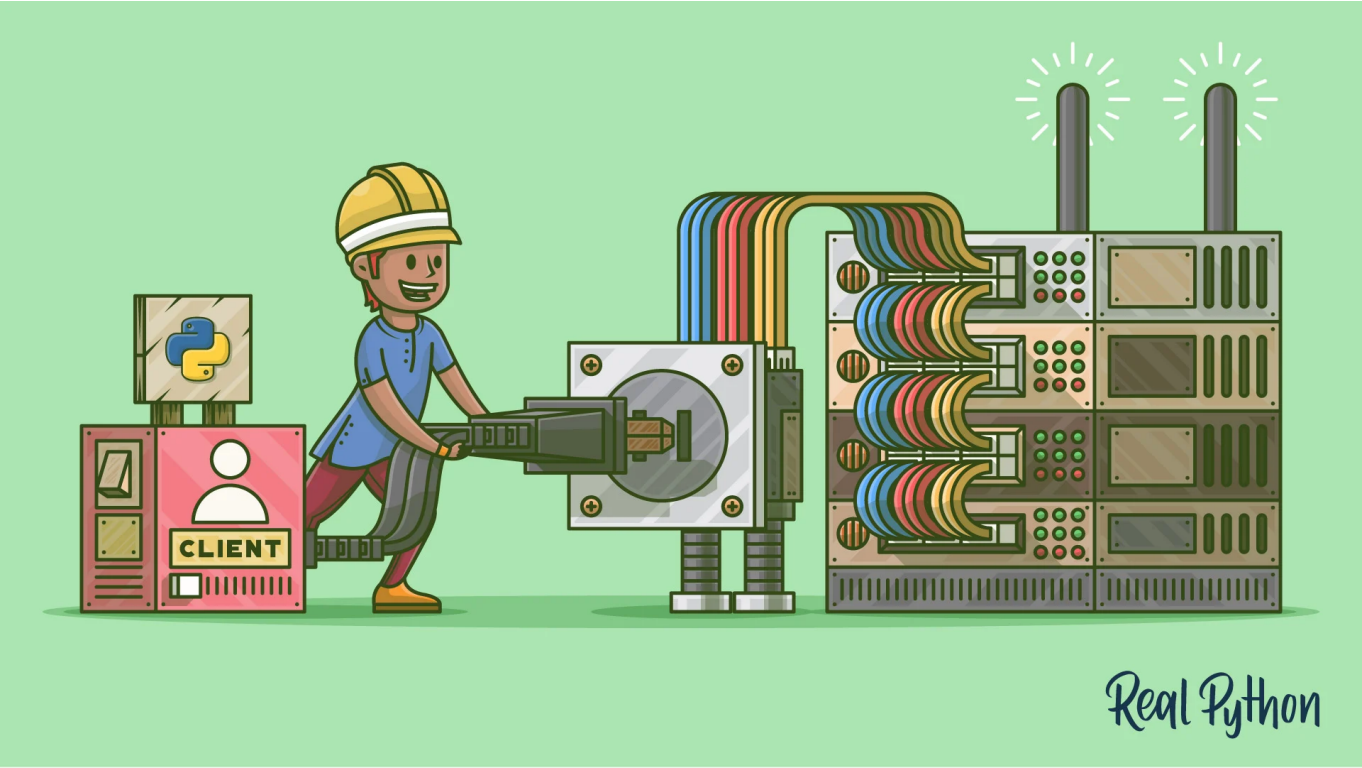


Python Socket

Bem vindo! O objetivo deste conteúdo é auxiliar a entender melhor o funcionamento de um Socket e como implementá-lo em Python.

❓ O que é este repositório?

Este repositório é fruto de uma atividade avaliativa da disciplina de Sistemas Distribuídos, ministrada pelo professor Eduardo Xavier, na Universidade Salvador (UNIFACS). A intenção deste é explicar, na prática, o que é um socket e ensinar como implementá-lo utilizando a linguagem Python

👤 Integrantes da equipe

- João Gabriel Santos Neves - matrícula UNIFACS: 042191010 - Github

📋 Requerimentos Mínimos de Software

O que você precisa fazer para executar este software.

📌 Instalação do Git

Caso você deseje baixar os arquivos deste respositório via git, você precisa ter o git instalado para clona-lo em um diretório local. Siga o passo a passo abaixo, a depender de qual sistema operacional você utilize:

Windows

Faça o download do instalador na página oficial do git e instale em sua máquina normalmente.

GNU/Linux

Siga este manual do site oficial do git

MacOS

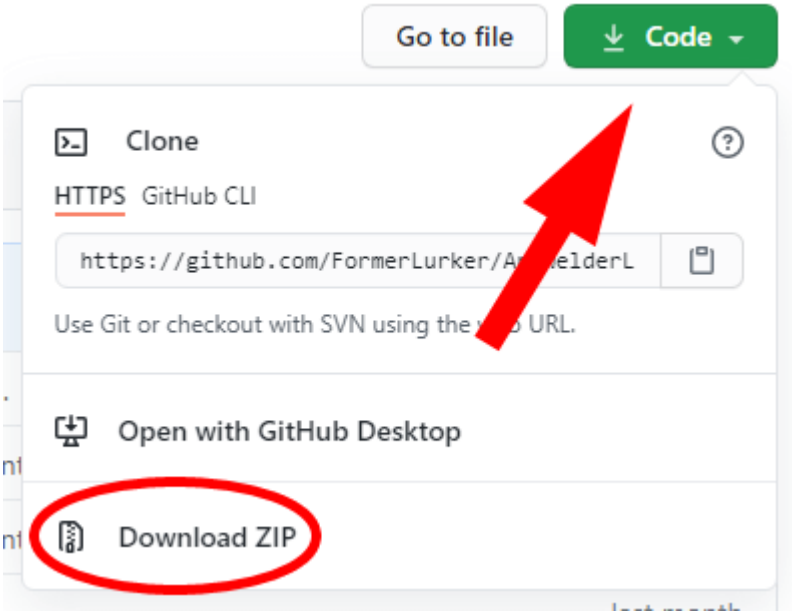
Siga este manual do site oficial do git

📌 Copiando este repositório para um diretório local

Há duas formas de fazer o download deste respositório:

Download do ZIP

Dentro da página deste repositório no github tem a opção de fazer download de um arquivo zipado, com todos os módulos py dentro dele, conforme imagem:



Download via git

1. abra um terminal (powershell, cmd ou qualquer outro terminal do seu sistema operacional);
2. crie uma diretório (pasta) dentro do seu disco rígido através do comando: `mkdir <nome_da_pasta>`
3. clone este repositório para seu diretório local: Usando HTTPS: `git clone https://github.com/jgsneves/python-socket.git` Usando SSH: `git clone git@github.com:jgsneves/python-socket.git`

📌 Instalação do Python

Você precisa ter o Python instalado em sua máquina.

Windows

Caso você utilize o sistema operacional windows, dirija-se ao site oficial da linguagem, faça download do compilador do `python3`.

GNU/Linux

Siga as instruções contidas neste manual criado pela python brasil.

MacOS

Siga as instruções contidas neste manual criado pela python brasil.

⏰ Execução dos arquivos python

Depois do compilador Python instalado, executaremos cada arquivo do servidor e dos clientes, usando o seguinte comando:

```
python <path_do_arquivo>
```

Abra um terminal para cada socket (um para professor, outro para aluno e outro para servidor).

📪 O que é um socket?

O `socket` é um nó. É um ponto de comunicação entre dois sistemas distribuídos. Sistema é um software e sistema distribuído é uma aplicação que somente é completa com a utilização de mais de um programa localizado em máquinas diferentes (claro que podemos simular duas localizações distintas em uma mesma máquina, como é o caso dessa aplicação deste respositório).

É legal pensar o `socket` como um container, dentro de um navio. Este navio sai de um ponto A para o ponto B e o container contém uma mensagem que precisamos trafegar entre os pontos A e B. Geralmente utilizamos o `socket` em conjunto com o protocolo IP, que é responsável por endereçar toda máquina dentro de uma rede. O IP é o endereço numérico de toda máquina conectada a uma rede. Assim, cada `socket` possui, como `metadado`, o endereço de ip, a porta de comunicação e protocolo de comunicação utilizado dos pontos A e B.

> *OBS: metadado é um dado que não faz parte do conteúdo de uma informação, apenas representando uma informação complementar. Por exemplo: em uma carta, o seu conteúdo é um dado e as informações de endereço do remetente e do destinatário são metadados.*

⁉️ Onde utilizamos o socket?

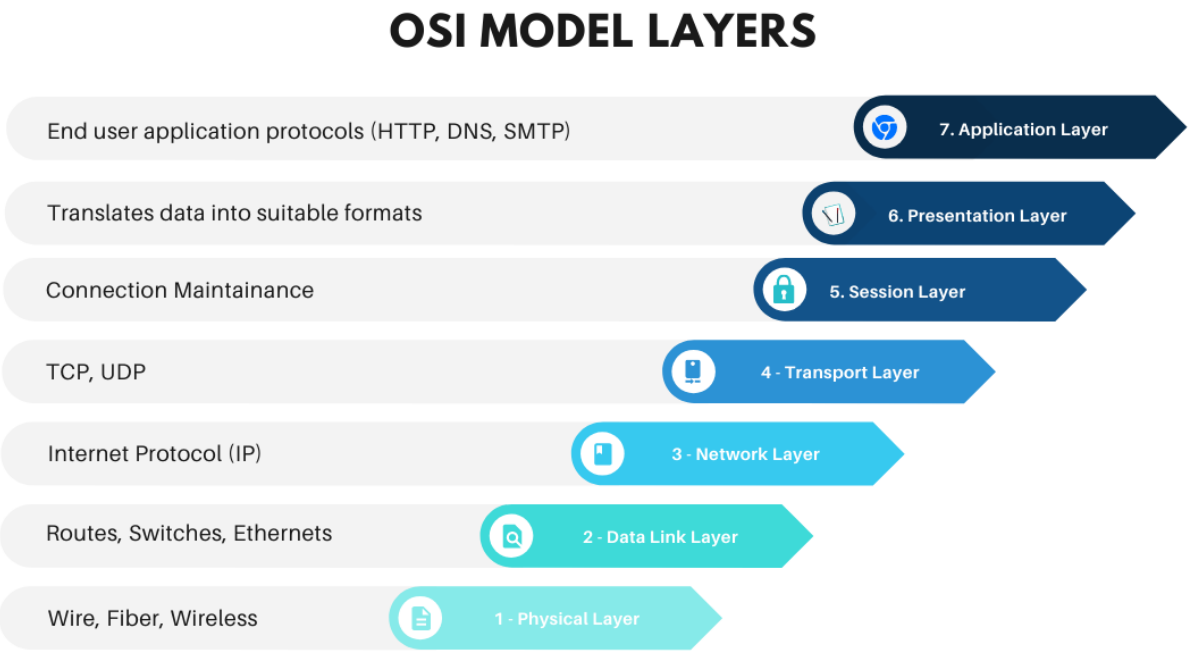
Bom, utilizamos o socket em praticamente toda a internet. Praticamente toda a internet é baseada no modelo cliente/servidor. Então, quando você entra em qualquer site de internet, por baixo dos panos, você está utilizando o web socket.

🚢 O que é TCP/IP e UDP?

Basicamente, entendemos que o `socket` é um conteiner dentro de um navio. O mapa que diz a rota que o navio deve percorrer para chegar do ponto A ao B é o protocolo IP. O navio onde esse container está trafegando é um `pacote do tipo TCP ou UDP`. Basicamente, o navio é do tipo TCP ou UDP. Os protocolos

TCP e UDP são basicamente formas de trafegar pacotes, a maneira como carregamos a informação (com um pacote) através da rede. O protocolo TCP é um protocolo baseado na `confiança`, que exige a confirmação de chegada da informação, enquanto que o UDP é baseado na performance e `velocidade`, não se importando com perdas de informação no tráfego.

A imagem abaixo ilustra bem onde os protocolos TCP/UDP agem nesse fluxo de informação:



🔭 O que é concorrência (multithread)?

Falar de concorrência é falar de paralelismo. É a capacidade do computador processar mais de um dado ao mesmo tempo. Aqui não se fala apenas de velocidade de processamento (afinal, com o multithread, o computador conseguiria realizar mais de uma tarefa ao mesmo tempo, realizando aquilo que se propõe de forma mais célere), mas também na possibilidade de criar programas básicos que necessitem de processamento concorrente.

Em um programa muito básico, onde todas as instruções ocorrem de cima para baixo, em sequência, o multithread não faz falta. O computador executa todas as instruções e chega ao fim. Porém, quando precisamos criar programas um pouco mais complexos, como é o caso desse repositório, precisamos que o computador reserve atenção a mais de uma execução (aqui, nós temos um servidor e 2 clientes conectando a ele ao mesmo tempo).

Para escutar dois clientes e um servidor ao mesmo tempo, por exemplo, o processador deve reservar uma thread para cada requisição e outra para executar o servidor. Caso não houvesse o multithread, não seria possível conectar mais de um cliente no mesmo servidor, e ao mesmo tempo.

🔭 Diferença entre `thread` e `process`

Os dois referem-se a processamento. Porém, um `process` tem uma ou mais `threads`. O processo aloca recursos computacionais (processamento, memória, etc) para realizar uma ou mais `threads`, enquanto que esta última é uma unidade básica de processamento. Quando executamos um script python, por exemplo, utilizamos uma thread de um processo.

Nesse nosso repositório, criamos uma `thread` para processar o script e uma para cada conexão de cada cliente (professor e aluno). Caso não houvesse a possibilidade de fazer esse paralelismo, quando o professor se conectasse, seria necessário encerrar a conexão deste para que qualquer aluno pudesse se conectar e registrar sua presença. Não é esse o comportamento esperado.

🏃 Vamos pro código!

Agora que abordamos os conceitos básicos, vamos para o código!

📂 Estrutura de pastas (arquitetura)

```
src
|---clients
|       |--student.py      //implementação do cliente-estudante
|       |--teacher.py      //implementação do cliente-professor
|---models
|       |--classtype.py    //modelo de uma turma/classe
|---server
|       |--server.py       //implementação do servidor
|---services
|       |--service.py      //implementação do serviço de controle de chamada
|---index.py        //o script que deve ser executado para abrir o servidor
```

💡 Explicando a aplicação

Cada equipe deve criar um aplicativo de CHAMADA para uma turma de alunos utilizando sockets. O aplicativo deve funcionar da seguinte maneira:

```
• Há dois tipos de clientes: professor e aluno.
• Um professor pode iniciar ou finalizar o processo de chamada.
• Um aluno envia seu registro de presença em uma chamada ativa.
• O servidor hospeda a lista de alunos que estão presentes em cada chamada e
atende clientes (alunos e professores).
```

Não há necessidade de implementar uma interface gráfica. Uma interface textual, desde que compreensível, pode ser usada sem que isso prejudique a pontuação do trabalho.

A aplicação pode ser desenvolvida em C++, Java, GoLang, Dart ou Python. As equipes devem enviar ao professor um relatório com o seguinte conteúdo:

```
• Apresentação dos componentes da equipe
• Descrição dos requerimentos mínimos de software necessários para execução da
aplicação
(o que deve estar instalado no cliente e no servidor para a aplicação funcionar)
```

- Todos os códigos-fonte da aplicação
- Instruções de instalação da aplicação

📐 Detalhes das regras de negócio

Detalhes que devem estar presentes na implementação do projeto.

☎️ Cliente tipo “Professor”

• Dispara o início da chamada informando a identificação numérica da turma e recebe do servidor uma confirmação contendo a data e hora do início da chamada.

• Dispara o encerramento da chamada informando a identificação numérica da turma e recebe do servidor a data e a hora de encerramento da chamada, além de um vetor contendo todas as matrículas de alunos que responderam a chamada. Após o recebimento, deve exibir (na console da aplicação) uma lista contendo as matrículas (alunos presentes) que recebeu.

☎️ Cliente tipo “Aluno”

• Responde à chamada enviando sua matrícula e a identificação numérica de sua turma. Recebe como resposta a identificação numérica da turma, a data e a hora em que sua presença foi registrada pelo servidor.

• Se a turma a qual o aluno enviou seu registro de presença não estiver com a chamada iniciada pelo professor, o aluno recebe a mesma resposta, mas com a identificação da turma zerada (indicando assim, que o registro de presença foi recusado pelo servidor).

💻 Servidor de Chamada

• Aguarda solicitações de clientes (escuta)

• Recebe solicitação e identifica o tipo:

```
• Se for um início de chamada se prepara para armazenar as matrículas de alunos
que responderão
a chamada da turma informada e retorna confirmação adequada (data e hora) ao
professor.
Lembre-se: pode haver mais de um professor fazendo chamada ao mesmo tempo para
turmas diferentes.
Cabe à equipe implementar uma solução para isso.

• Se for um encerramento de chamada retorna ao professor a confirmação adequada
(data, hora e vetor de matrículas) ao professor e apaga as informações referentes
a chamada da
turma que foi encerrada. Lembre-se que podem haver outras turmas ainda fazendo
chamada e isso não
pode sofrer interferência. Cabe à equipe implementar uma solução para isso.
```

```
• Se for um registro de presença, verifica se existe uma chamada ativa para a
turma informada pelo aluno.
```

• Caso a chamada esteja ativa (professor já iniciou, mas não encerrou), insere a matrícula do aluno no armazenamento de alunos presentes e devolve a confirmação adequada ao aluno (identificação da turma, a data e a hora em que a presença foi registrada).

• Caso a chamada não exista (professor não iniciou ou já encerrou a chamada da turma) apenas devolve a resposta adequada ao aluno (zero em lugar da identificação da turma, a data e a hora em que a presença foi negada).

Componentes da aplicação

index.py

É o script que deve ser executado para rodar o `server`. Rodar o comando na pasta root do projeto:

```
python src/index.py
```

No mais, o arquivo foi implementado da seguinte forma:

```
from server.server import Server
from services.service import Service

class_service = Service()

new_server = Server(class_service)

new_server.run()
```

Ele importa o serviço e passa sua instância no construtor do servidor. Após o instanciamento do server, executa o método `run()` para rodar o servidor.

src/clients/student.py

Implementação do cliente de estudante. Este cliente tenta registrar presença na lista de uma turma/classe.

```
import socket

PORT = 5050
HOST = socket.gethostbyname(socket.gethostname())
ADDRESS = (HOST, PORT)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDRESS)
```

```
print('''
--------------[BEM VINDO]-----------------

Olá Aluno,
Bem vindo ao sistema de registro de presença.

-----------------------------------------
''')

msg = ''
client_code = 'student'

def use_client_code(msg, client_code):
    return f'{msg},{client_code}'

while True:
    user_number = input('Informe seu número de matrícula: ')
    user_class = input('Informe o número da matéria
    em que deseja registrar presença: ')
    msg = str(user_number + '/' + user_class)
    encoded_package = str.encode(use_client_code(msg, client_code))
    client.send(encoded_package)
    response = client.recv(1024)
    decoded_response = response.decode()
    print(decoded_response)
```

Este módulo implementa um `socket` com as configurações do servidor (tem que ser um match perfeito entre porta, endereço e tipo de socket). O método `socket.gethostbyname(socket.gethostname())` busca o `IP address` da máquina quee está executando o script. Em outras palavras, é o mesmo que usar `localhost`.

O método `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` cria um `socket` do tipo `TCP/IP`.

Após a criação do `socket`, o script solicita dois `inputs` do usuário: **1) número de matrícula** e **2) número da matéria que ele deseja registrar a presença**. Depois os `inputs` são parseados em uma `string` que os separa por uma barra `/`.

Como o `socket` trafega apenas com dados no formato de `bytes`, o script codifica a `string` em `bytes` através do método `str.encode()`, o envia para o servidor e fica aguardando o feeedback (a resposta) do mesmo. Após a chegada da resposta, esta é decodificada de `bytes` para `string` e printada na tela.

Para executar este `client`, basta executar o seguinte comando da raiz do projeto:

```
python src/clients/student.py
```

A função `use_client_code()` basicamente parseia a mensagem que será enviada de acordo com o modelo combinado em contrato com o `servidor`. A mensagem enviada segue o modelo de ser uma string que possui o `input` do usuário, uma vírgula e o código do cliente:

```
"<input>,<client_code>"
```

src/clients/teacher.py

O módulo responsável por implementar o script do professor. Este usuário ativa e desativa lista de presença de aulas/turmas/classes.

```
import socket

PORT = 5050
HOST = socket.gethostbyname(socket.gethostname())
ADDRESS = (HOST, PORT)

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(ADDRESS)

print('''
---------------[BEM VINDO]------------------

Olá Professor,
Bem vindo ao sistema de registro de presença!

Neste sistema é possível:
- Abrir chamada de uma matéria;
- Encerrar a chamada da mesma;
- Consultar a lista de alunos presentes;

------------------------------------------
''')
print('Digite o número da turma para iniciar/encerrar sua chamada:')

msg = ''
client_code = 'teacher'

def use_client_code(msg, client_code):
    return f'{msg},{client_code}'

while True:
    userInput = input()
    msg = str(userInput)
    encodedPackage = str.encode(use_client_code(msg, client_code))
    client.send(encodedPackage)
    response = client.recv(1024)
    decoded_response = response.decode()
    print(decoded_response)
```

Mais uma vez, inicialmente setamos as informações do servidor e criamos um `socket` do tipo `TCP/IP` para realizar a conexão. Basicamente o script solicita que o usuário informe o número da turma e, caso a chamada

daquela turma já esteja iniciada, ela será encerrada. O loop de repetição faz a mesma coisa que o cliente `student`: solicita um input, codifica em `bytes` e envia para o servidor.

Após, aguardará o retorno da resposta pelo `server` e printará no terminal depois de decodificá-la, já que todo o dado que transita pelo `socket` está na forma de `byte`.

Para executar este script, execute este comando na raiz do projeto:

```
python src/clients/teacher.py
```

src/models/classtype.py

A primeira e única model do repositório. Ela representa uma aula (classe ou matéria, como queira se referir) e tem seus métodos.

```
class ClassType:
    def __init__(self, number: str) -> None:
        self.number = number
        self.is_opened = False
        self.present_students = []

    def open(self):
        self.is_opened = True

    def close(self):
        self.is_opened = False

    def add_student(self, student_number):
        self.present_students.append(student_number)

    def remove_student(self, student_number):
        self.present_students.remove(student_number)

    def is_student_present(self, student_number):
        for student in self.present_students:
            if student == student_number:
                return True
        return False
```

1) atributos

atributo	tipagem	descrição
number	string	o número que representa a turma. Número de identificação.
is_opened	boolean	indicação se a chamada está ativa ou não. Se a turma está aberta ou não
present_students	array	lista contendo todos os números de identificação dos estudantes

2) métodos

método	parâmetros	descrição
open	None	modifica o atributo is_openeded para True
close	None	modifica o atributo is_openened para False
add_student	student_number: número que representa o estudante (string)	adiciona novo estudante na lista
remove_student	student_number: número que representa o estudante (string)	remove um estudante da lista por número de identificação
is_student_present	student_number: número que representa o estudante (string)	retorna se um determinado estudante está presente na classe ou não

Para instanciá-la:

```
nova_turma = ClassType(<número_da_turma>)
```

src/server/server.py

Este módulo implementa, de fato, o servidor da nossa aplicação. Ele é instanciado no script do `index.py`.

```
import socket
import threading

class Server:
    def __init__(self, class_service) -> None:
        self.PORT = 5050
        self.HOST = socket.gethostbyname(socket.gethostname())
        self.ADDRESS = (self.HOST, self.PORT)
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.service = class_service

    def run(self):
        self.server.bind(self.ADDRESS)
        self.server.listen()
        print('Servidor está escutando...')
        while True:
            connection, client = self.server.accept()
            print('Cliente conectado: ', client)
            thread = threading.Thread(
                target=self.handle_connection,
                args=(connection, client)
            )
            thread.start()

    def handle_connection(self, connection: socket, _):
```

```
        while True:
            data: bytes = connection.recv(1024)
            decoded_data = data.decode()
            response: str = self.service.handle_message(decoded_data)
            print(response)
            encoded_response = response.encode()
            connection.send(encoded_response)
```

Ele é instanciado com atributos em valores fixos, que são os mesmos valores de porta, IP, e o socket do tipo `TCP/IP` dos clientes. Como informado anteriormente, essas informações devem ser idênticas aos sockets dos clientes, possibilitando a conexão entre eles. O único valor variável é do `service`, que é um objeto instanciado de um `Service`, que será explicado posteriormente.

```
novo_servidor = Server(<objeto_de_Service>)
```

1) atributos

atributos	tipagem	descrição
PORT	int	A porta utilizada para conexão entre sockets
HOST	string	O endereço de IP para conexão entre sockets
ADDRESS	tuple	tupla com os dados de HOST e PORT
server	socket	O socket do tipo SOCK_STREAM (TCP/IP)
service	instanceof Service	Objeto da classe Service()

2) métodos

métodos	parâmetros	descrição
run	None	Executa de fato o servidor. Em estado de escuta de requisições
handle_connection	connection: socket	Manejar cada requisição: criar uma nova thread para cada requisição e tratar com o Service

Como precisamos multithread para tratar várias requisições ao mesmo, e o Python não tem comportamento multithread por padrão, precisamos importar o pacote `threading` e criar uma `thread` através do método `threading.Thread(target=self.handle_connection, args=(connection, client))`. Basicamente, ele recebe o método que gerencia cada requisição, o `handle_connection()` como primeiro argumento e a tupla `(connection, client)` que é recebida como resposta da função `<socket>.recv()`.

src/services/service.py

O único serviço da aplicação. É o módulo que trata a mensagem que chega do cliente.

```
```

```
from datetime import datetime

from models.classtype import ClassType

class Service:
    def __init__(self) -> None:
        self.active_classes = []

    def add_class(self, class_obj):
        self.active_classes.append(class_obj)

    def remove_class(self, class_number):
        for item in self.active_classes:
            if item.number == class_number:
                self.active_classes.remove(item)

    def is_class_active(self, class_number):
        for item in self.active_classes:
            if item.number == class_number:
                return True
        return False

    def get_class_present_students(self, class_number):
        for item in self.active_classes:
            if item.number == class_number:
                return item.present_students


    def handle_teacher_message(self, message: str):
        currentDateAndTime = datetime.now().strftime("%d/%m/%Y às %H:%M")

        if self.is_class_active(message):
            present_students = self.get_class_present_students(message)
            self.remove_class(message)
            return f'A chamada da turma {message} foi encerrada em
{currentDateAndTime}! Os seguintes alunos registraram presença:
{present_students}'
        else:
            new_class = ClassType(message)
            self.add_class(new_class)
            return f'A chamada da turma {message} foi ativada em
{currentDateAndTime}!'

    def handle_student_message(self, message: str):
        currentDateAndTime = datetime.now().strftime("%d/%m/%Y às %H:%M")
        student_infos = message.split('/')
        student_number = student_infos[0]
        class_number = student_infos[1]

        if self.is_class_active(class_number):
            for item in self.active_classes:
                if item.number == class_number:
                    if item.is_student_present(student_number):
```

```
                        return f'Você já registrou sua presença nesta turma!'
                    else:
                        item.add_student(student_number)
            return f'Presença registrada com sucesso na turma {class_number} em
{currentDateAndTime}!'
        else:
            return f'Esta turma não está com presença ativa.
            Solicitação rejeitada em {currentDateAndTime}'

    def handle_message(self, decoded_data: str):
        message_list = decoded_data.split(',')
        client_message: str = message_list[0]
        client_code: str = message_list[1]

        if client_code == 'teacher':
            return self.handle_teacher_message(client_message)
        else:
            return self.handle_student_message(client_message)
```

Primeiro ele identifica o remetente tratando a mensagem que chega do cliente com o método `handle_message()`: se é um cliente `professor` ou `aluno`. Isso é feito analisando o conteúdo da mensagem. Todas as mensagens seguem o seguinte arquétipo, um contrato combinado com os `clients`:

```
"<input>,<client_code>"
```

A depender do tipo de cliente (`client_code`), adota-se um dos caminhos de tratamento de dados: se o cliente for um `professor`, a mensagem é tratada pela `handle_teacher_message()`, caso contrário, pelo método `handle_student_message()`. Os outros métodos são mais auxiliares. Vamos ao detalhamento da classe:

1) atributos

atributo	tipagem	descrição
active_classes	array	lista de classes ativas

2) métodos

métodos	parâmetros	descrição
add_class	class_obj: instanceof ClassType()	Adiciona nova classe à lista de classes ativas
remove_class	class_number: string. Número identificador da classe	Remove uma classe da lista de classes ativas pelo número
is_class_active	class_number: string. Número identificador da classe	Verifica se uma class está na lista de classes ativas

métodos	parâmetros	descrição
get_class_present_students	class_number: string. Número identificador da classe	Fornece a lista de estudantes presentes na classe
handle_teacher_message	message: string. Mensagem do cliente professor	Maneja a mensagem do professor
handle_student_message	message: string. Mensagem do cliente estudante	Maneja a mensagem do estudante
handle_message	decoded_data: string. Mensagem do socket	Maneja a mensagem do socket

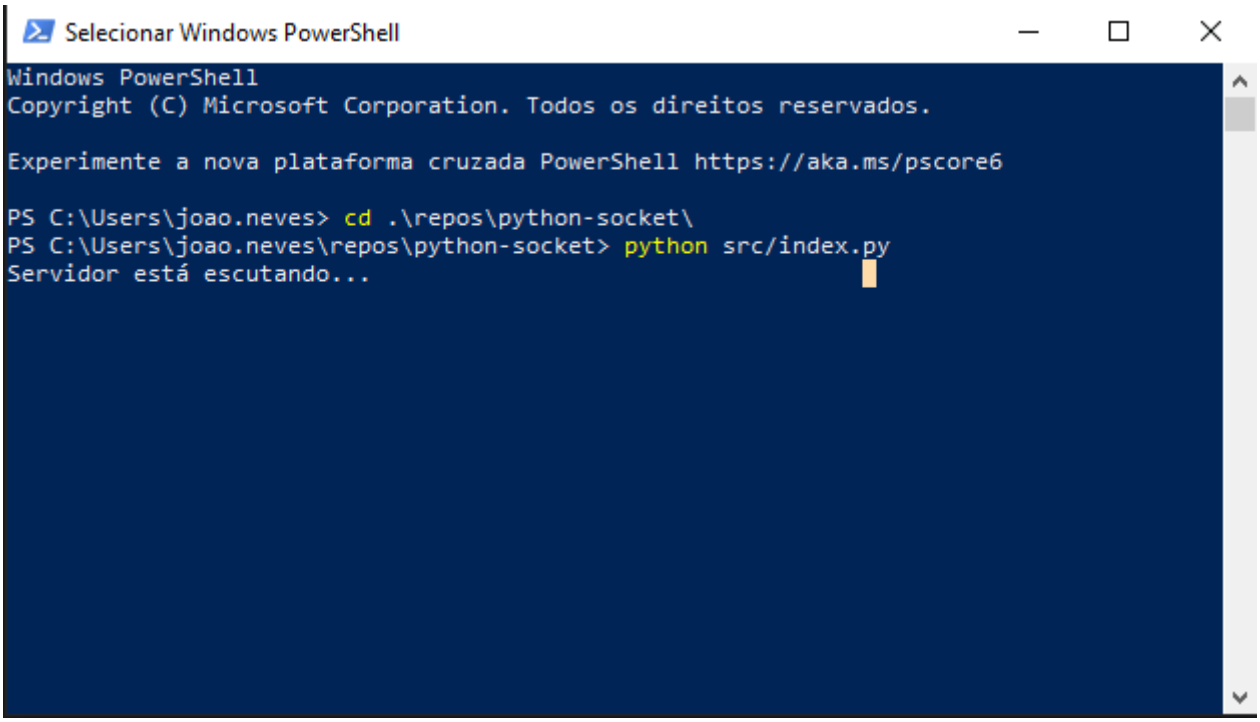
⏱️ Aplicação em execução

Com tudo pronto, podemos executar a aplicação:

1) Rodar o servidor

Em um terminal, digite o seguinte comando na raiz do projeto:

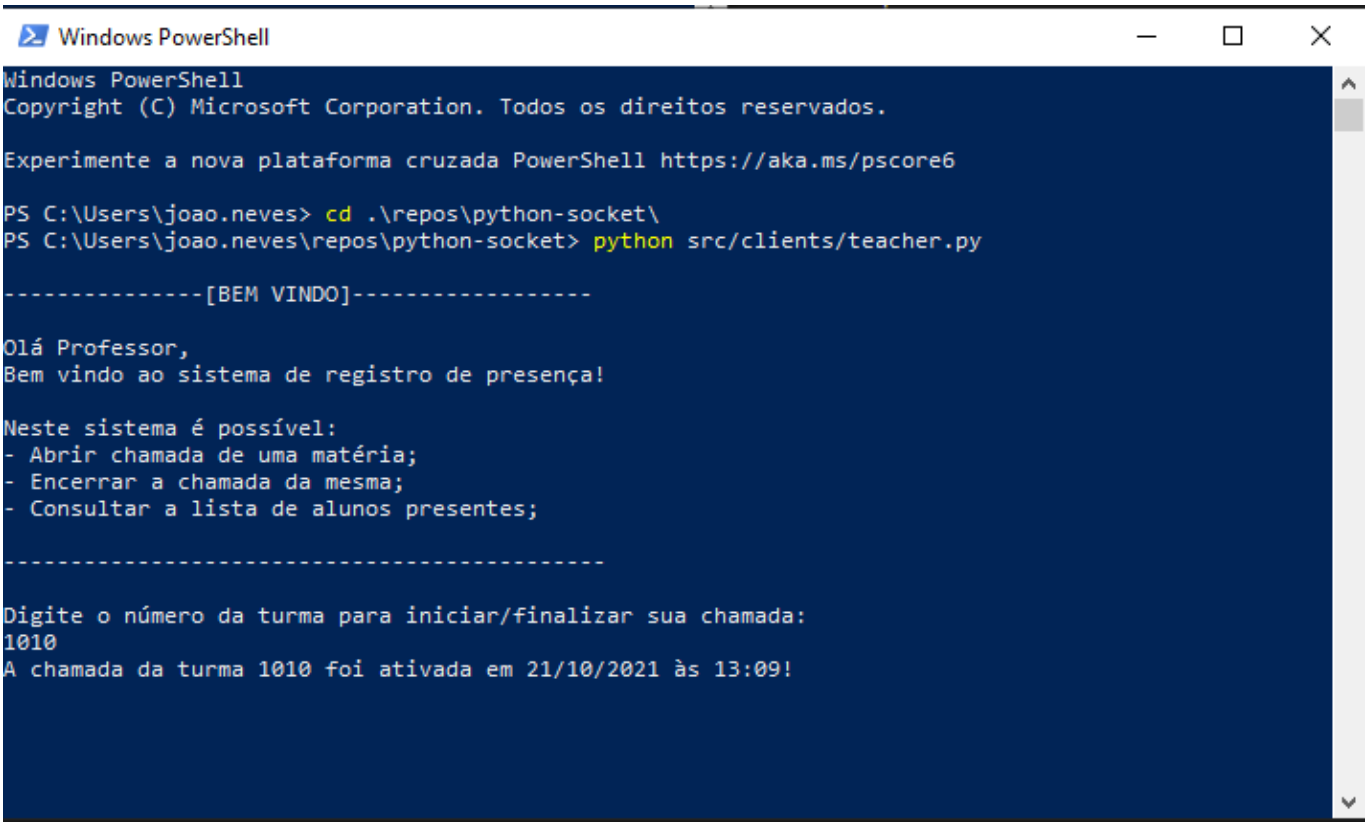
```
python src/index.py
```



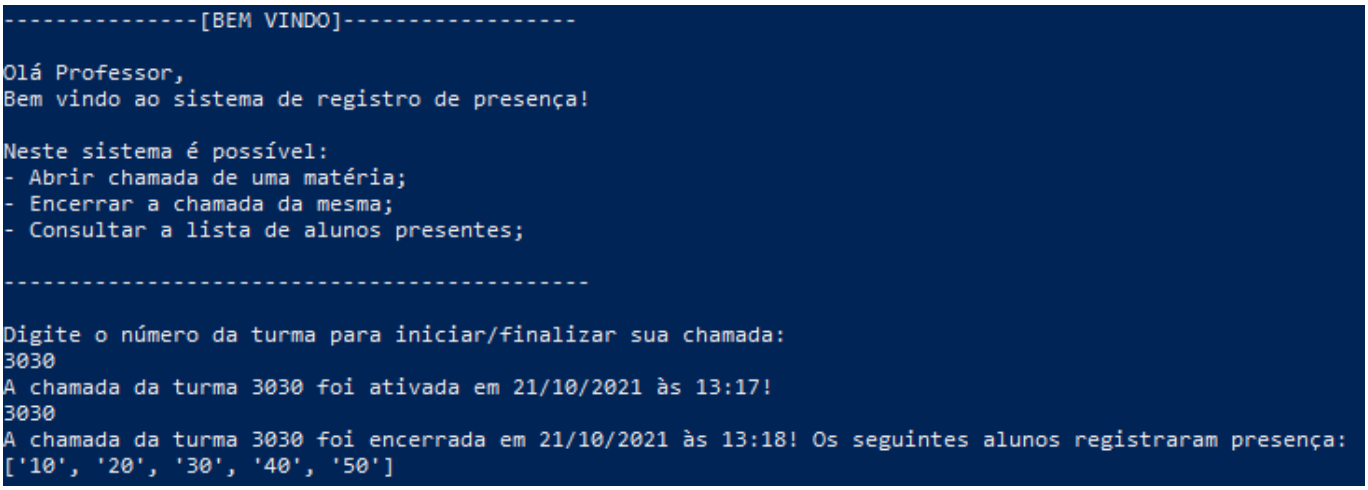
2) Rodar o teacher.py

Abra outro terminal e digite o seguinte comando na raiz do projeto:

```
python src/clients/teacher.py
```



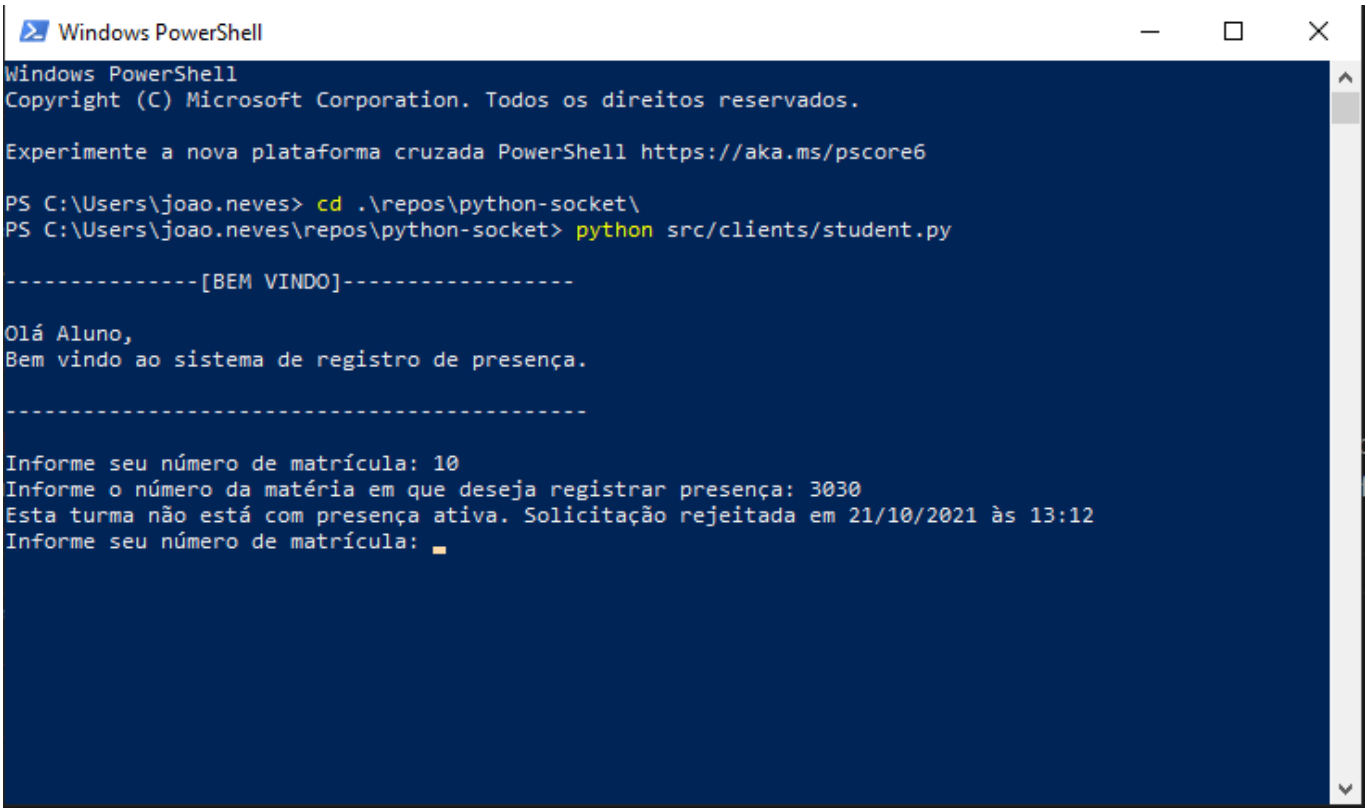
Pode repetir o processo quantas vezes forem necessárias. O sistema suporta múltiplos professores conectados ao mesmo tempo. Basta informar o número da turma que deseja iniciar a chamada. Um único professor pode também abrir mais de uma chamada ao mesmo tempo.



3) Rodar o student.py

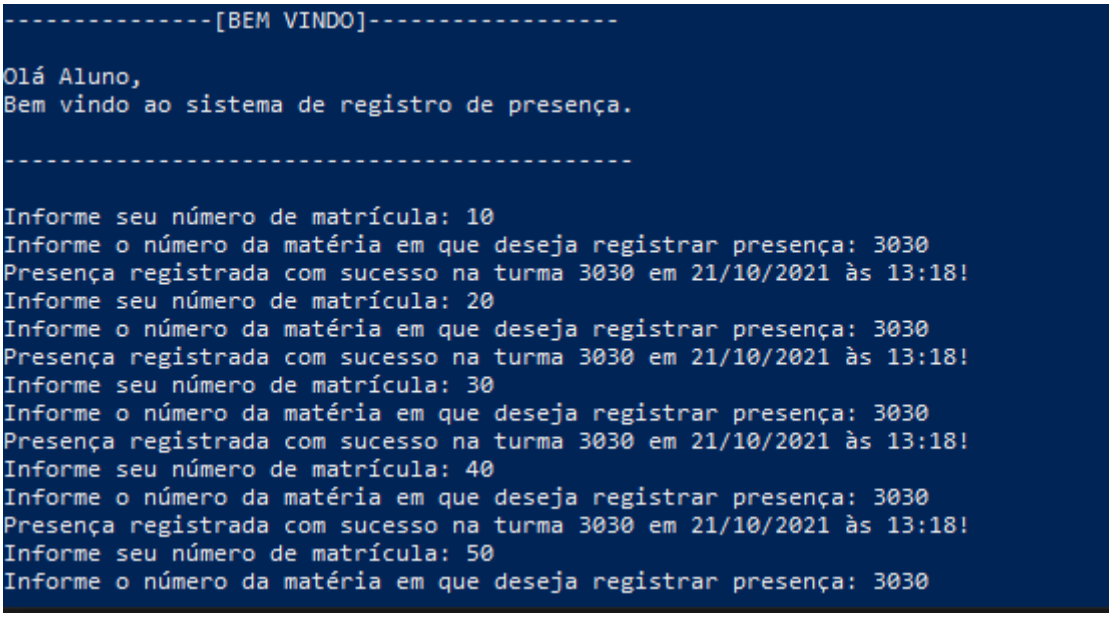
Mais uma vez, abra outro terminal e digite o seguinte comando na raiz do projeto:

```
python src/clients/student.py
```



Pode repetir o processo quantas vezes forem necessárias. O sistema suporta múltiplos alunos conectados ao mesmo tempo. Basta informar o número da matrícula e o número da matéria que seja registrar presença. Caso o número da turma informado não coincidir com nenhuma turma aberta, o sistema informará o erro. Caso positivo, registrará a presença do aluno.

Múltiplos alunos podem registrar presença no mesmo script, sem necessidade de instanciar um único cliente para cada aluno.



📌 Referências

- Diferença entre thread e process
- Como usar processamento concorrente (multithread) com Python
- Básico de Socket - wiki

- Criar um socket básico em python - @bosontreina
- HOWTO: socket em python
- Python Socket Programming Tutorial - @TechWithTimm
- Whats the OSI model?
- What's SOA