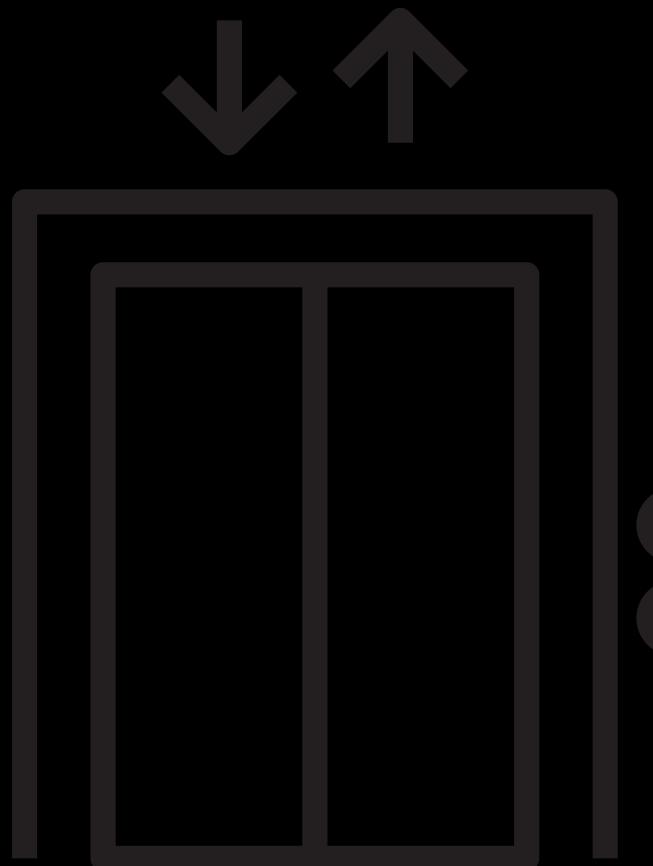

Elevador Multi Thread

O que é?

Uma simulação de elevadores utilizando threads para otimizar o transporte de passageiros em um prédio.

Objetivo

Desenvolver um sistema de gerenciamento de elevadores que minimize o tempo de espera dos passageiros e, consequentemente, reduza o consumo de energia,



Linguagem

- Python

Bibliotecas

- Threading
- Random
- Time



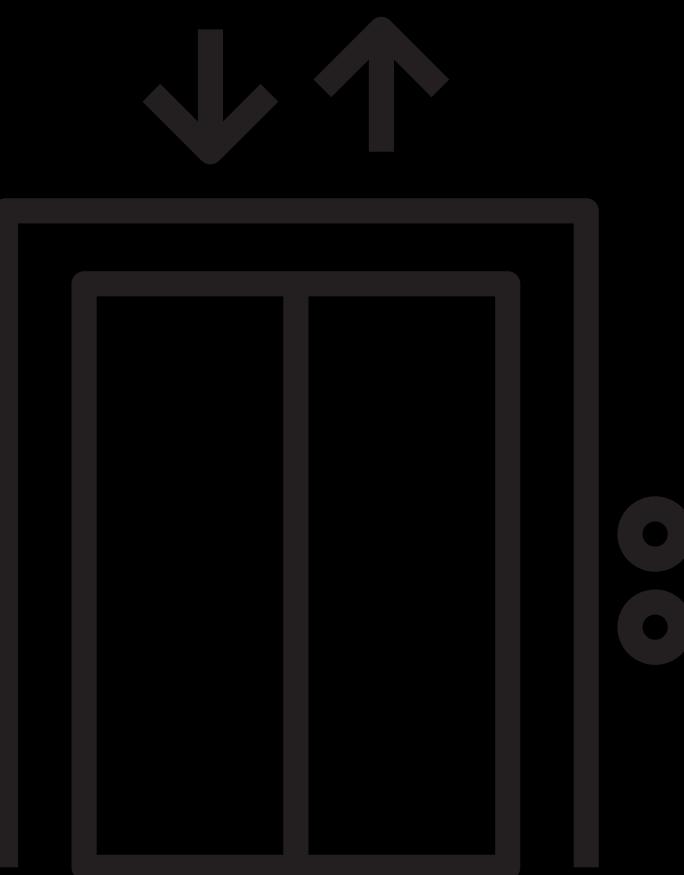
Estrutura Principal

Filas de Espera

- Cada andar possui uma fila de passageiros aguardando para entrar em um elevador. Essa estrutura é um dicionário onde as chaves são os andares (0 a 9) e os valores são listas de destinos dos passageiros.

Elevador (Elevador)

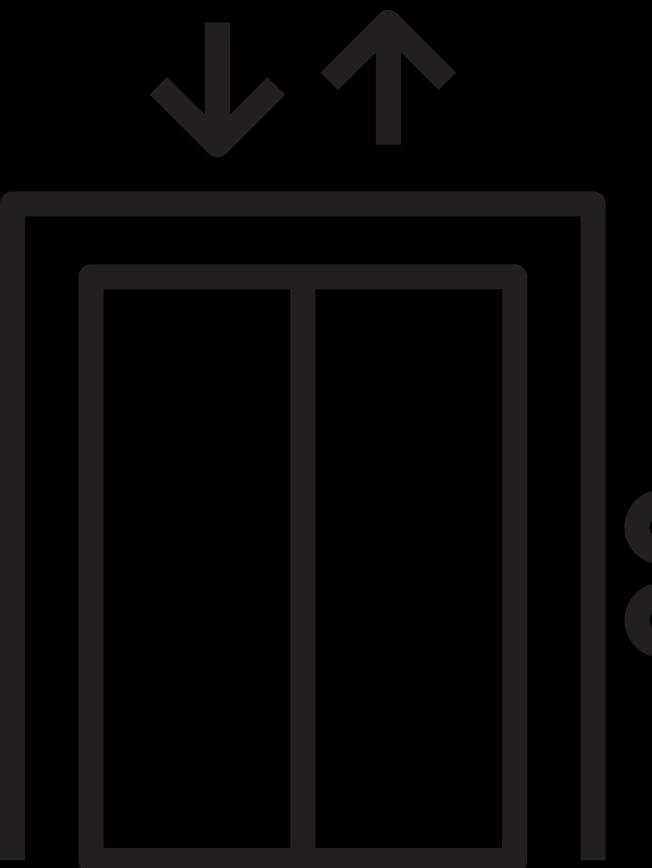
- Cada elevador é representado por uma thread, permitindo que ele se move e atenda passageiros de forma independente dos outros elevadores.
- Cada elevador tem:
- Um identificador.
- Um andar atual.
- Uma lista de passageiros que estão dentro do elevador.



Estrutura Principal

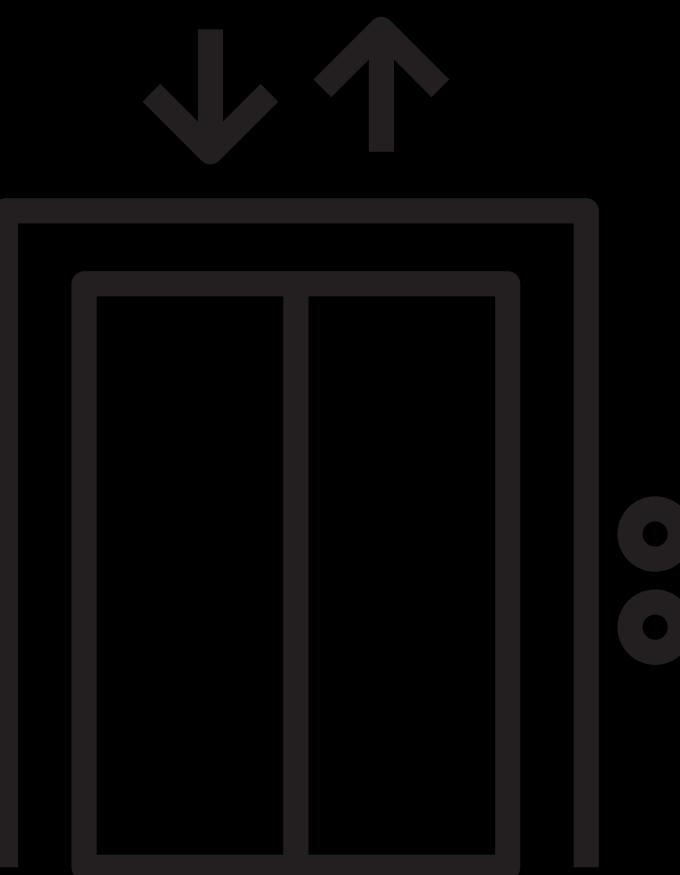
Passageiro (Passenger)

- Cada passageiro é um objeto que contém:
- Origem: o andar de onde ele começa.
- Destino: o andar para onde ele vai.



Execução e Multithreading

- O programa cria “n” elevadores, e cada um é iniciado como uma thread.
- A função gerar_passageiros() também roda em uma thread separada para criar “n” passageiros de forma contínua e independente dos elevadores.
- Obs: Tanto o número de elevadores, quanto o número de passageiros é configurado no código de acordo com o limite máximo desejado



```
def simular_elevadores(num_elevadores):
    global fila_espera, passageiros_atendidos, inicio_simulacao, fim_simulacao, passageiro_counter
    # Reinicia variáveis globais para cada simulação
    fila_espera = {i: [] for i in range(NUM_ANDARES)}
    passageiros_atendidos = 0
    passageiro_counter = 1
    inicio_simulacao = time.time()
    fim_simulacao = None

    # Define a semente fixa para que a geração dos passageiros seja a mesma em todas as simulações
    random.seed(42)

    print(f"\n== Iniciando simulação com {num_elevadores} elevador(es) ==")
    elevadores = [Elevador(i) for i in range(num_elevadores)]
    for elevador in elevadores:
        elevador.start()

    gerador = threading.Thread(target=gerar_passageiros)
    gerador.start()

    for elevador in elevadores:
        elevador.join()
    gerador.join()

    tempo_total = fim_simulacao - inicio_simulacao if fim_simulacao else 0
    print(f"== Simulação finalizada em {tempo_total:.2f} segundos ==\n")
    return tempo_total

# Simulação com diferentes números de elevadores para comparação
for num in [1, 2, 3]:
    print(f">>>> SIMULAÇÃO COM {num} ELEVADOR(ES) <<<")
    simular_elevadores(num)
```

Função gerar_passageiros()

```
def gerar_passageiros():
    global passageiro_counter
    gerados = 0
    while gerados < TOTAL_PASSEIROS:
        origem = random.randint(0, NUM_ANDARES - 1)
        destino = random.randint(0, NUM_ANDARES - 1)
        if origem != destino:
            novo_passageiro = Passageiro(passageiro_counter, origem, destino)
            passageiro_counter += 1
            fila_espera[origem].append(novo_passageiro)
            print(f"[GERADOR] Novo passageiro criado: ID {novo_passageiro.id} | Origem {origem}, Destino {destino}")
            gerados += 1
            time.sleep(random.uniform(0.3, 1.0))
```

```
NUM_ANDARES = 10
NUM_ELEVADORES = 3
CAPACIDADE_ELEVADOR = 5
TOTAL_PASSEIROS = 10
```

Como Utilizamos Threads

. No uso das threads, o código simula o funcionamento simultâneo de múltiplos elevadores de forma paralela.

Benefícios do uso de Threads

. Execução simultânea das tarefas, sem bloqueios.

. Maior eficiência no uso dos elevadores e menor tempo de espera.

. Simulação realista do funcionamento de um elevador com processos paralelos.