Geek University

Evolua seu lado geek!

www.geekuniversity.com.br

Entendendo o funcionamento das Threads

Entendendo o funcionamento das Threads

Vamos voltar ao programa que vimos ao conhecer a síntaxe básica de uma thread...

```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

Entendendo o funcionamento das Threads

```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

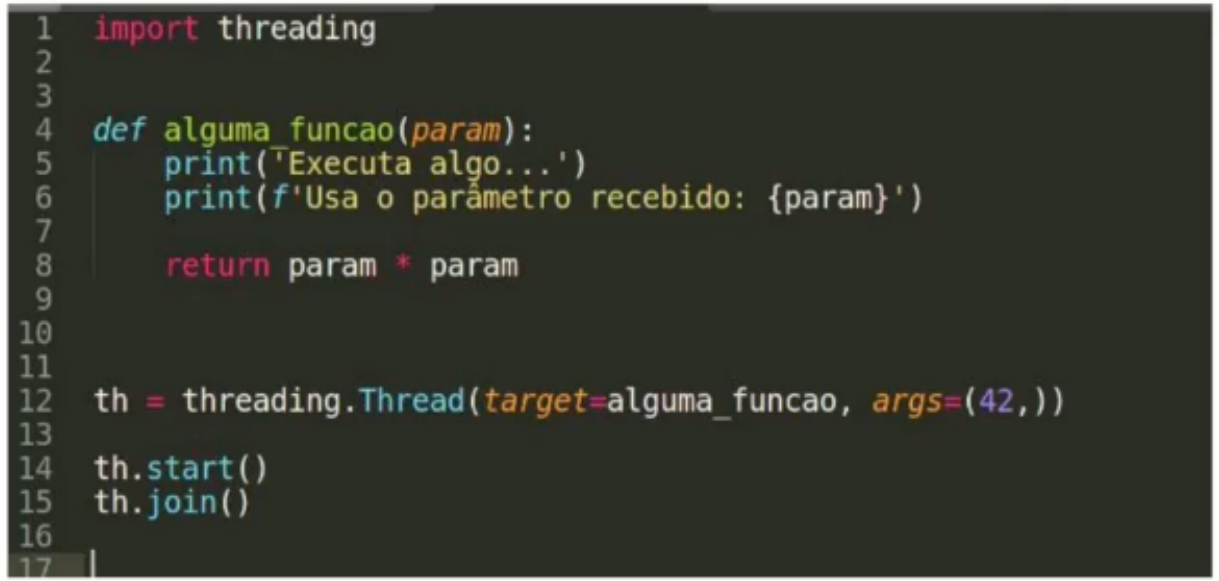
Quando o programa é executado, é criado um processo Python e uma thread (linha de execução) é iniciada para executar o programa.

Entendendo o funcionamento das Threads

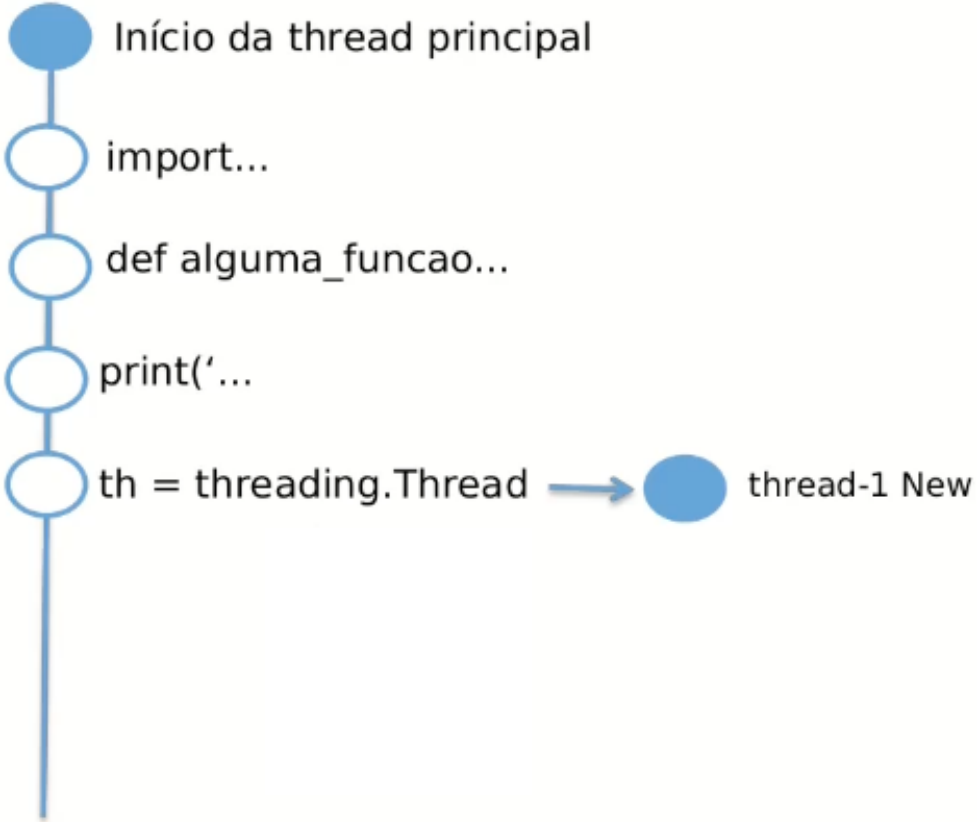
```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

Quando o programa é executado, é criado um processo Python e uma thread (linha de execução) é iniciada para executar o programa. Esta thread é a principal. Qualquer outra thread criada neste processo será filha desta thread principal. Desta forma uma estrutura hierárquica de threads é criada.

Entendendo o funcionamento das Threads



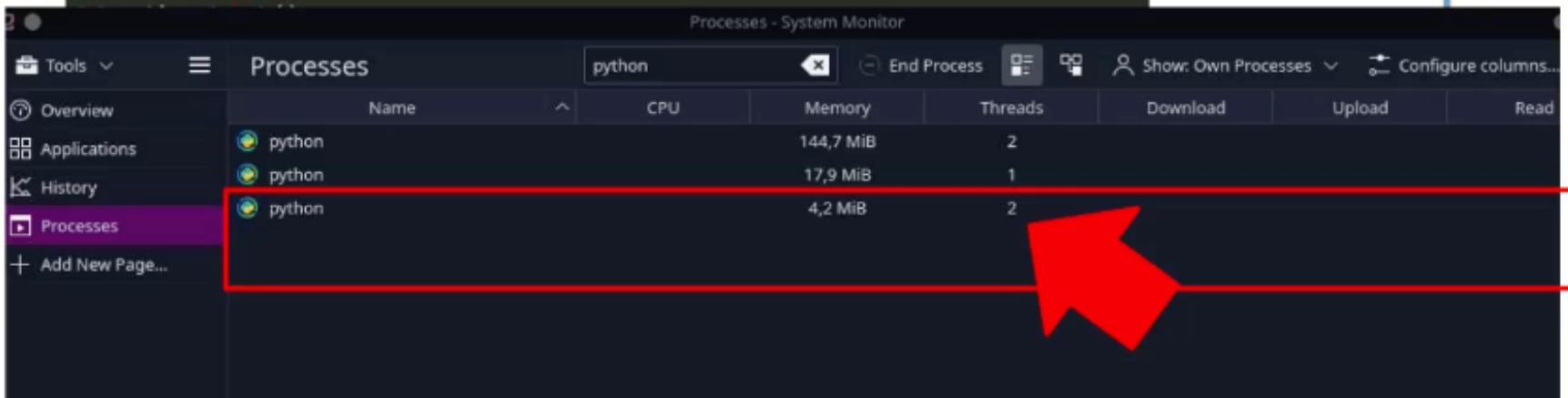
```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

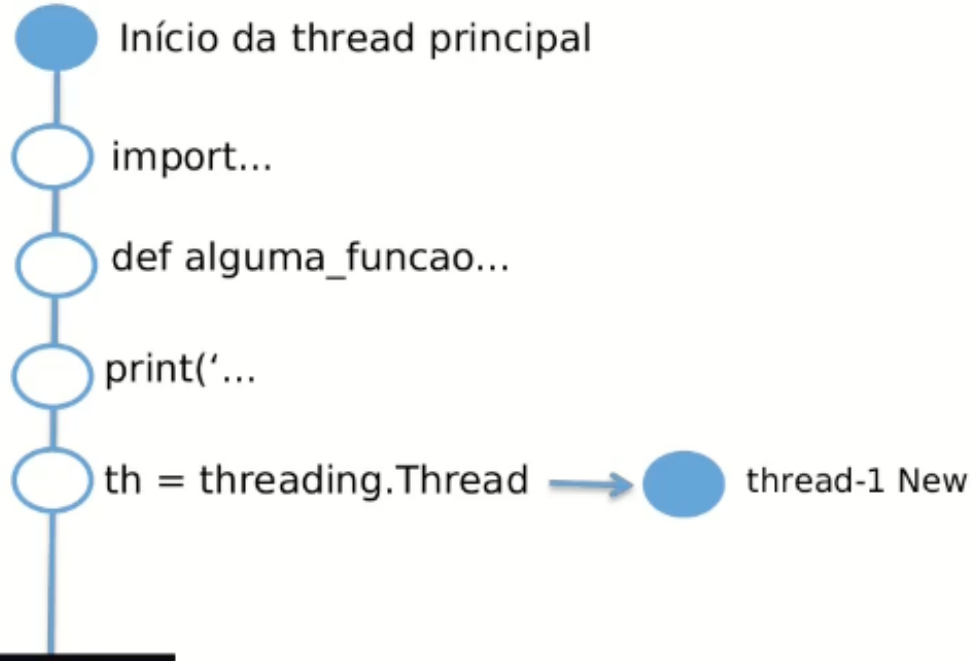


Note que neste momento ainda não executamos os métodos start() e join(). Desta forma o status da nova thread é New.

Entendendo o funcionamento das Threads

```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param


th = threading.Thread(target=alguma_funcao, args=(42,))
```

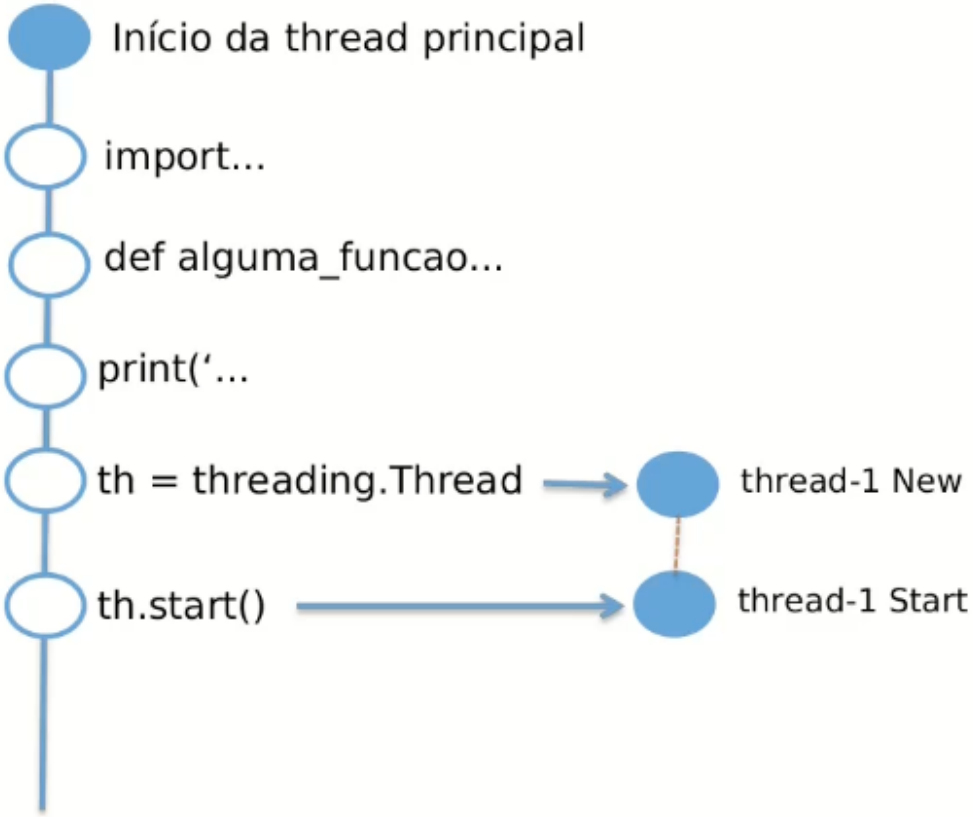




Se consultarmos o processo Python que está executando nosso programa de thread simples veremos que são criadas duas threads apesar de no programa estarmos criando diretamente apenas uma.

Entendendo o funcionamento das Threads

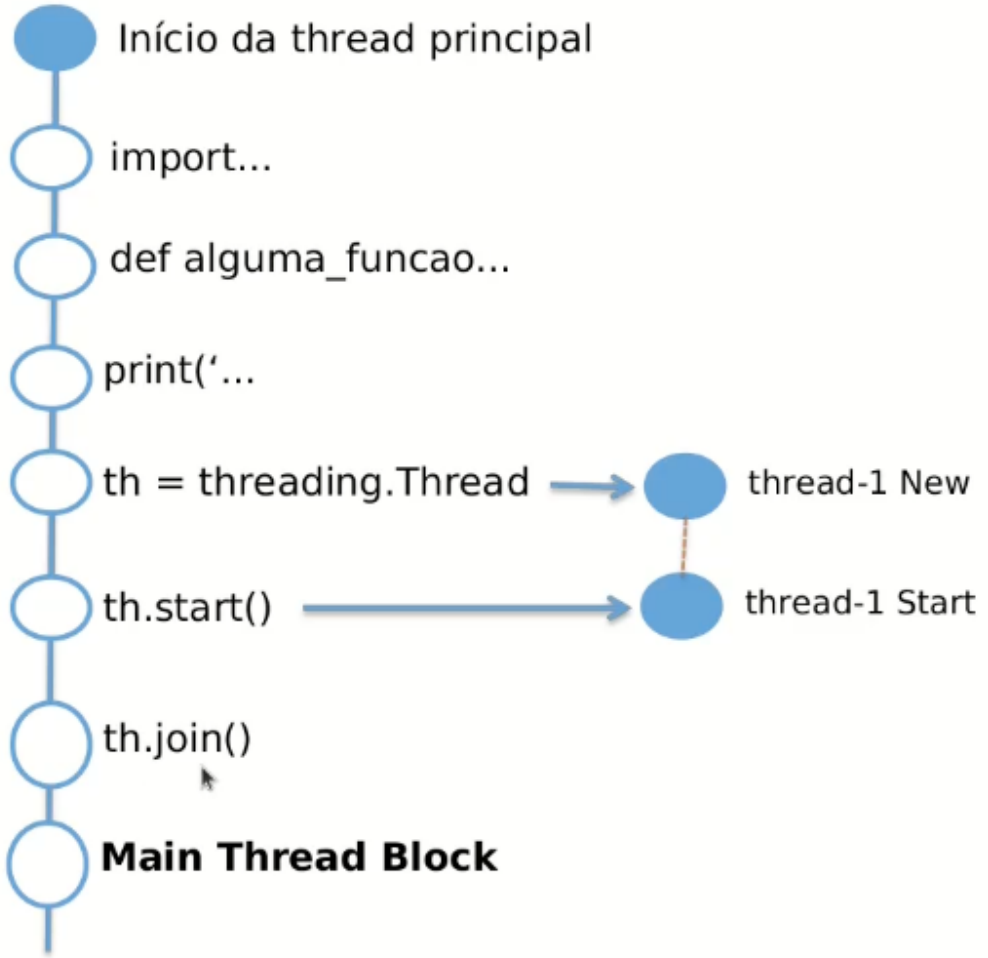
```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```



A partir do momento que executamos o método start() da thread, ela fica disponível (ready state) para o sistema operacional para que seja agendada sua execução, de acordo com a disponibilidade do processador.

Entendendo o funcionamento das Threads

```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

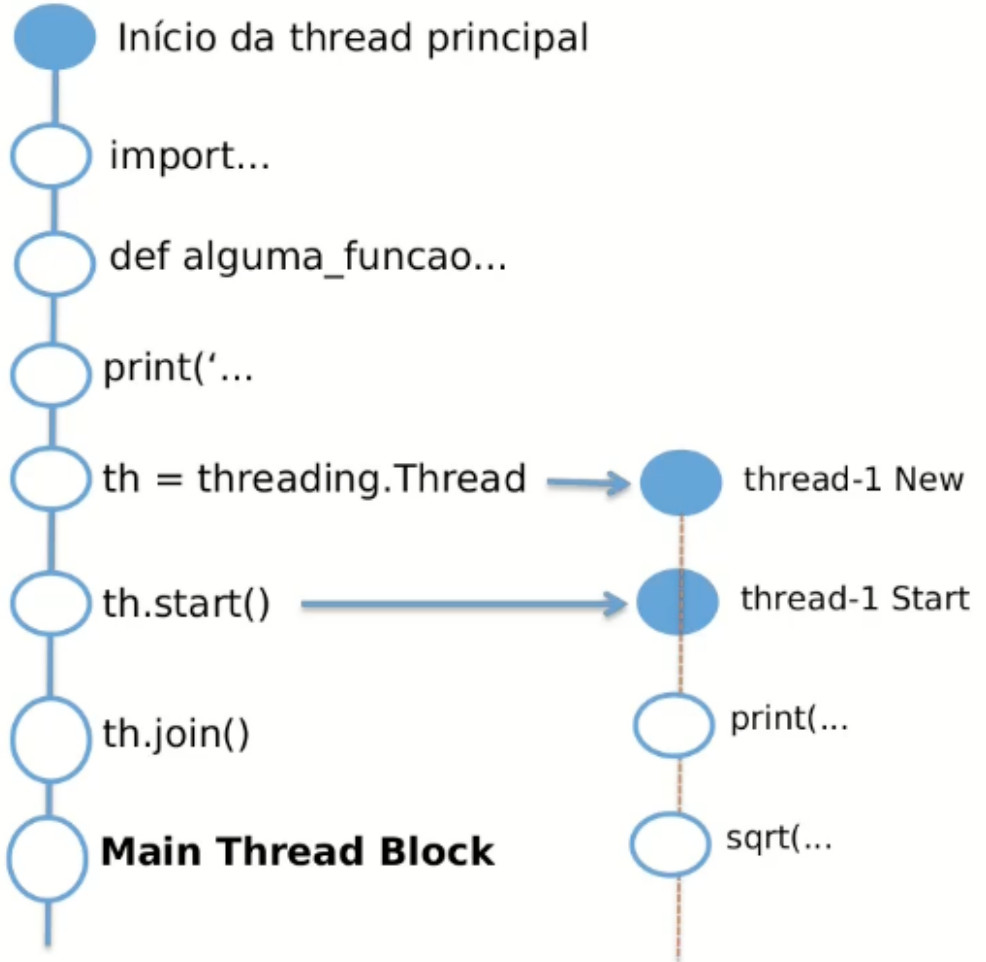


Quando executamos o método join() da thread, ela fica não somente com status "running" mas também ativa o Main Thread Block que faz com que qualquer outra coisa só seja executada após a thread finalizar seu trabalho.

Entendendo o funcionamento das Threads

```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

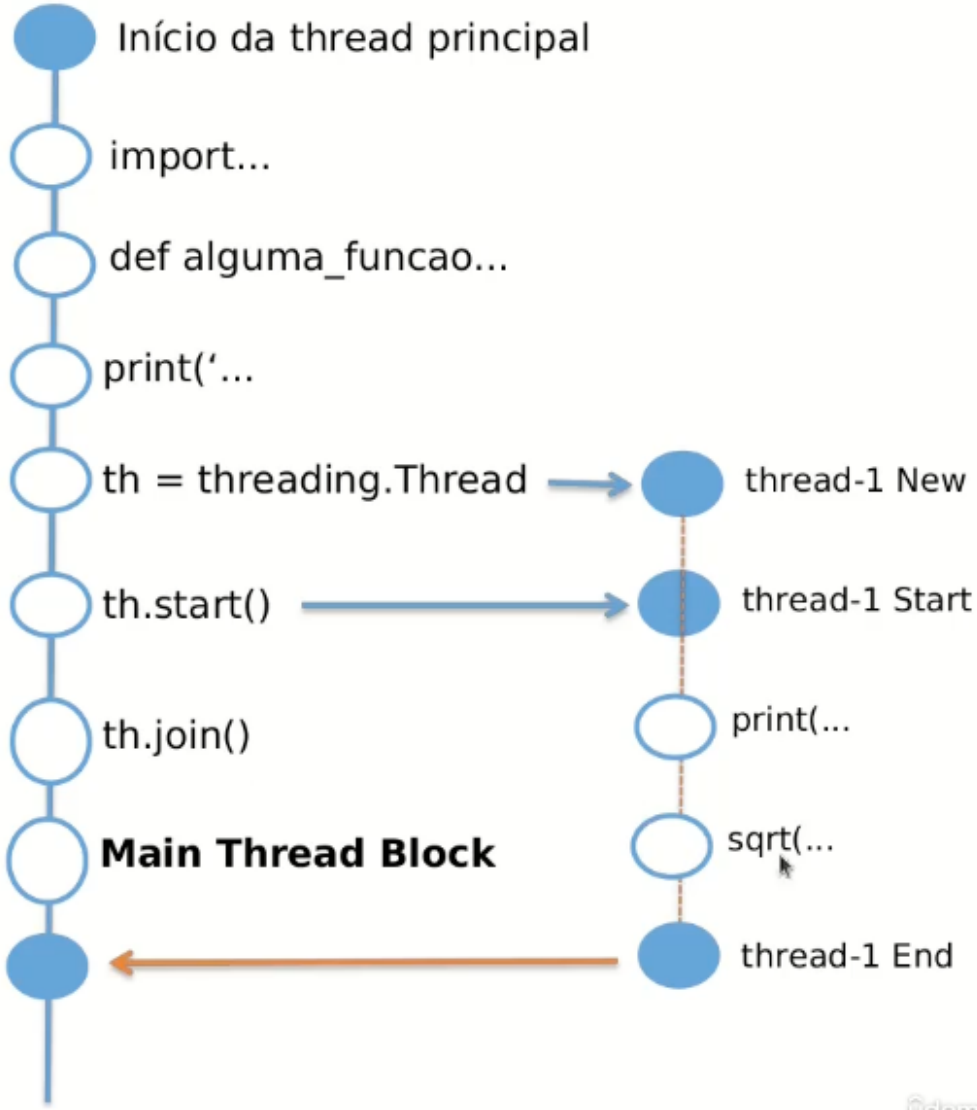
Quando executamos o método join() da thread, ela fica não somente com status “running” mas também ativa o Main Thread Block que faz com que qualquer outra coisa só seja executada após a thread finalizar seu trabalho.



Entendendo o funcionamento das Threads

```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

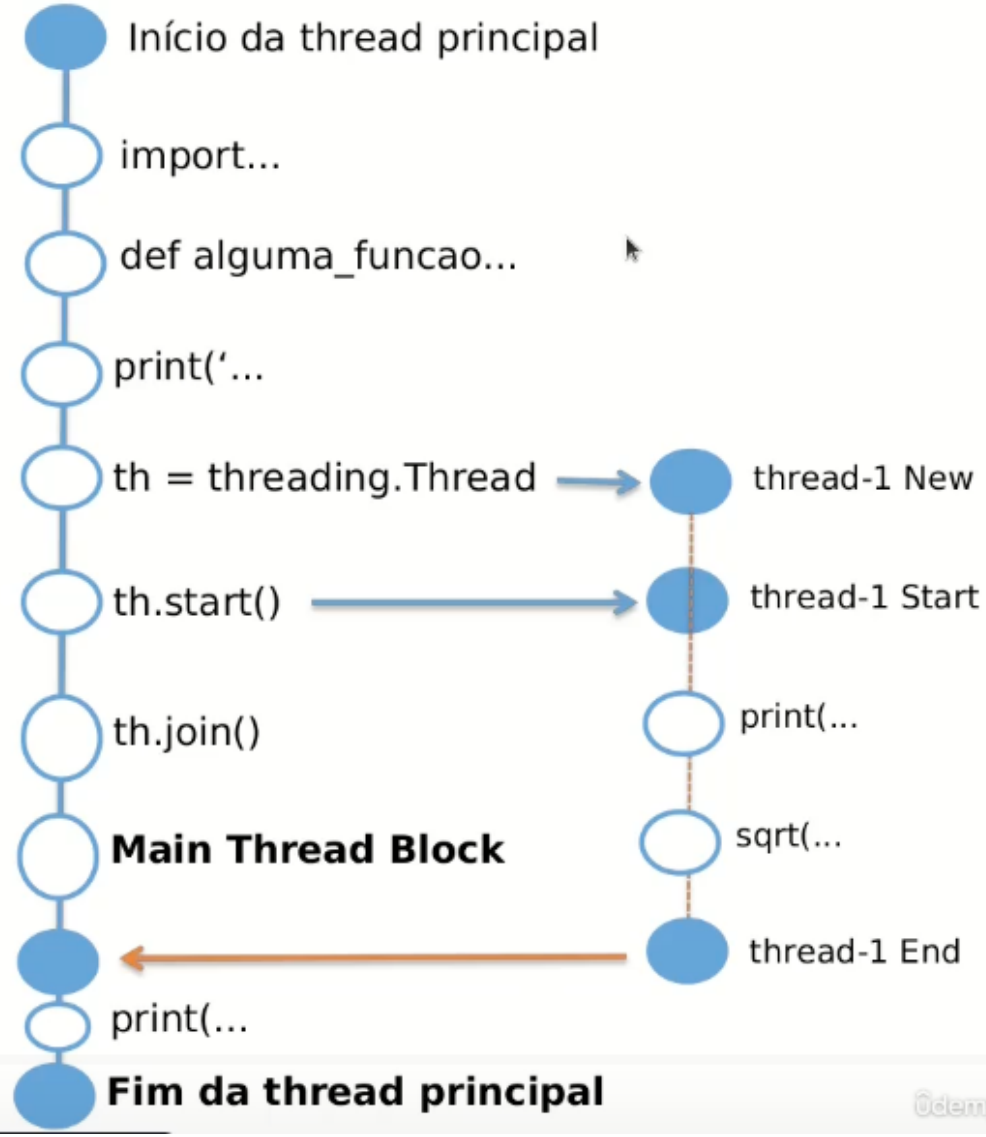
Quando executamos o método join() da thread, ela fica não somente com status “running” mas também ativa o Main Thread Block que faz com que qualquer outra coisa só seja executada após a thread finalizar seu trabalho.



Entendendo o funcionamento das Threads

```python
import threading


def alguma_funcao(param):
    print('Executa algo...')
    print(f'Usa o parâmetro recebido: {param}')

    return param * param



th = threading.Thread(target=alguma_funcao, args=(42,))

th.start()
th.join()

```

Quando executamos o método join() da thread, ela fica não somente com status “running” mas também ativa o Main Thread Block que faz com que qualquer outra coisa só seja executada após a thread finalizar seu trabalho.

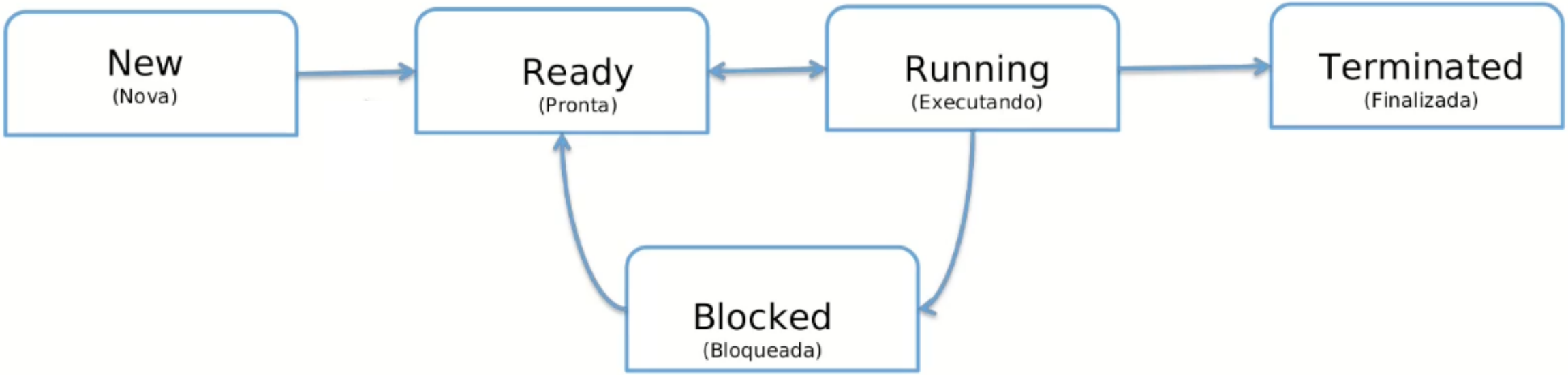


Entendendo o funcionamento das Threads

Ciclo de vida das Threads

Entendendo o funcionamento das Threads

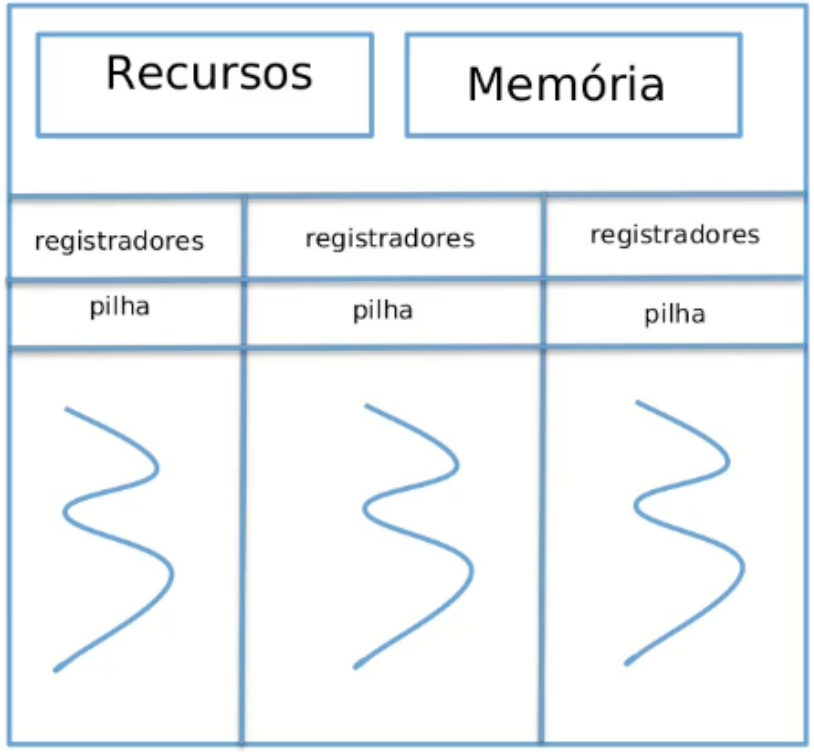
Ciclo de vida das Threads



Note que uma thread pode ficar alternando entre os estados ready/running/blocked...

Entendendo o funcionamento das Threads

Múltiplas threads



Em um programa com múltiplas threads, cada thread tem seu próprio registrador e pilha de execução. Porém, uma thread pode ler ou modificar dados de qualquer outra thread que faz parte de um mesmo processo.

Entendendo o funcionamento das Threads

Agendador (Scheduler)

Sistemas operacionais possuem um módulo que seleciona as próximas tarefas a serem admitidas no sistema e desta forma o próximo processo a ser executado.

Ou seja, quando um processo é executado, o sistema operacional utiliza um algoritmo* de escalonamento de processos que organiza e gerencia os novos processos. Este algoritmo possui um agendador, chamado de 'scheduler' que controla quando um processo ou thread irá executar e por quanto tempo.

Conforme vimos no ciclo de vida das threads, uma thread pode "navegar" pelos estados de ready, running blocked até ser terminada.

* O algoritmo de escalonamento difere de acordo com o sistema operacional utilizado. Desta forma a performance de execução de processos e threads é diferente de acordo com o sistema operacional.

Entendendo o funcionamento das Threads

Agendador (Scheduler)

Quando o agendador alterna entre uma thread e outra para execução ele faz isso usando um recurso chamado de “context switch” que salva e restaura o estado da thread ou processo.

OBS: Quando o agendador faz uso do “context switch” e o que está sendo alternado é uma thread pertencente a um mesmo processo é gasto menos recursos do processador. Quando o que é alternado é uma thread pertencente a outro processo ou mesmo um outro processo o processador gasta muito mais recursos.

Isso significa que, teoricamente, criar programas multi-threads deveriam performar melhor se comparado com multi-processos, porém ao fazer uso de multi-threads podemos nos deparar facilmente com um problema de interferência de threads, conhecida como “race conditions”, que é o processo de uma thread interferir nos resultados de outra.

Iremos ver isso na prática ainda nesta seção.

Geek University

Evolua seu lado geek!

www.geekuniversity.com.br