Essa é uma versão dos Guias de Estudo desenvolvida para auxiliar o processo de impressão e para facilitar a leitura dos guias por programas que fornecem a leitura automatizada para suporte a todos os alunos que necessitem. Dessa forma, não apresentaremos ilustrações nesse arquivo.* *#ParaTodosLerem***.

Módulo #10

Guias de estudo

Módulo #10

Até o momento, tudo que você aprendeu sobre programação e todos os programas que você desenvolveu são executados de ***maneira sequencial****, ou seja, os c****ódigos são executados pelo interpretador do Python linha a linha, em sequência****. Observe que quanto mais linhas temos ou quanto mais as estruturas de repetições executam linhas dos nossos códigos, maior é o tempo de execução dos nossos programas. Acontece que em alguns casos os programas ficam lentos e demoram tempo demais para finalizar, por isso surgiu a* ***programação paralela****.*

Em outras palavras, a ideia de paralelizar um código acontece quando há uma tarefa muito complexa e o ideal se torna dividi-la em tarefas menores a fim de obter uma solução mais rápida. Isso se torna possível graças ao processadores atuais que possuem vários núcleos que trabalham em conjunto processando operações simultaneamente.

Vamos imaginar o programa de um carro autônomo. Ele precisa retornar de maneira rápida a existência de um obstáculo na pista para evitar acidentes, ele precisa virar a esquerda ou direita rapidamente para não errar o caminho e etc... Imagine se este sistema tiver uma resposta demorada, isso trará muitos transtornos, acidentes e prejuízos.

Um outro cenário seria um site de vendas, imagine se a cada clique que você desse demorasse 10 minutos para obter resposta, você continuaria navegando por este site? Certamente não.

É bem provável que neste momento você esteja com a seguinte dúvida: "Então é necessário desenvolver todos os programas utilizando os conceitos de programação paralela?". A resposta é não! Isso só se torna necessário quando a demanda de tempo de execução de um programa for mais demorada que o necessário.Vamos ver um exemplo de como calcular tempo de execução de um programa que gera 1000 números aleatórios:

```
import random
import time

inicio = time.time()
for x in range(1000):
 numeros = random.randint(1, 1000)
fim = time.time()

print(f"Tempo de execução: {fim-inicio}")
```

Utilizamos a função time do módulo time que foi importado. Esta função foi chamada antes e depois da repetição, depois subtraímos o time do fim pelo time do início, assim identificamos quanto tempo a repetição demorou para ser executada. A saída do programa foi:

```
Tempo de execução: 0.0018754005432128906
```

Vamos agora somar todos os números aleatórios e mostrar na telas os números e as somas:

```
import random
import time
soma=0
inicio = time.time()
for x in range(900):
 numeros = random.randint(1, 1000)
 soma+=numeros
fim = time.time()
print(f"Tempo de execução: {fim-inicio}")
```

Criamos a variável soma para controlar a soma dos números gerados. O tempo de execução foi de: `0.002166271209716797`

Agora vamos ver o tempo de execução depois de adicionar uma linha que printa os números gerados dentro da repetição:

```
import random
import time
inicio = time.time()
for x in range(1000):
 numeros = random.randint(1, 1000)
 print(numeros)
fim = time.time()
print(f"Tempo de execução: {fim-inicio}")
Tempo de execução: 0.25201845169067383
```

E se aumentarmos de 1000 números aleatórios para 1000000? Como ficará o tempo de execução?

```
import random
import time

inicio = time.time()
for x in range(1000000):
 numeros = random.randint(1, 1000000)
 print(numeros)
fim = time.time()

print(f"Tempo de execução: {fim-inicio}")
```

Tempo de execução: 389.9945969581604

Observe que a cada linha que adicionamos o tempo de execução aumenta e que algumas funções levam mais tempo para serem executadas do que outras.

Bom, vamos executar os 2 blocos de código em um único programa, o bloco de códigos que gera números aleatórios e os mostra na tela e o outro que gera números aleatórios e soma-os.

Vamos então utilizar o conceito de threads neste último programa. Será que o desempenho melhora? Em programação paralela o que nos permite executar mais de uma linha de código de uma vez são as Threads pois elas compartilham a área de memória com outras linhas.

```
from ast import arg
import random
import time
from threading import Thread

#primeiro vamos criar uma função que implementa o código anterior

def geraAleatorio(qtd):
 for x in range(qtd):
   numeros = random.randint(1, 1000000)
   print(numeros)

#cria o objeto thread t1 que executará a função acima
t1 = Thread(target=geraAleatorio,args=[1000000])
#inicia a thread
t1.start()
```

Primeiro, o bloco de códigos que gera números aleatórios e printa-os na tela foi transformado em um método chamado `geraAleatorio` *para que possamos informar a thread o que queremos executar. O parâmetro qtd informa quantos números queremos gerar.*

Execute o código acima e veja o quão rápido ele é finalizado e compare com o programa que não foi implementado usando o conceito de threads.

Lembre-se

Sempre que você iniciar o desenvolvimento de alguma parte do seu programa pense qual proporção aquele conjunto de códigos pode tomar. Se a tendência é aumentar muito o tempo de execução, implemente programação paralela naquele ponto.