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 #9

Guias de estudo

Desenvolvimento Assíncrono

No curso de Python iniciante você aprendeu sobre threads, as quais possibilitam que um programa realize diversas tarefas simultaneamente, sem precisar criar vários processos separados.

Um thread é executado de forma independente das outras threads em execução e compartilham o mesmo espaço de memória, o que permite que elas acessem os mesmos dados e recursos, como variáveis, arquivos abertos, entre outros. Com isso, as threads podem se comunicar entre si facilmente.

Nesse tipo de programação, as tarefas são divididas em sub-tarefas menores e, em seguida, cada uma dessas sub-tarefas é executada em um processador ou núcleo diferente.

Você viu também que uma grande vantagem de usar threads é que isso pode aumentar a eficiência do sistema, pois diminui o tempo de execução comparando-se com um programa sequencial.

O ***desenvolvimento assíncrono****, assim como programação paralela, lida com tarefas computacionais que precisam ser executadas simultaneamente, porém são duas abordagens distintas. A programação assíncrona é um estilo de programação que permite que várias tarefas sejam executadas ao mesmo tempo, mas não necessariamente em paralelo. Em vez disso, as tarefas são executadas de forma concorrente.*

Dessa forma, você pode executar duas, três, ..., dez implementações e, em seguida, todas elas serão executadas de maneira independentemente ou ao mesmo tempo. A ***programação assíncrona*** *evita delays ou tempos de espera de execução dos programas, pois já que estamos executando algo sincronicamente, podemos ter bloqueios no processo pela necessidade de esperar que alguma execução seja finalizada.*

Que tal exemplificar tudo isso?

- *A programação assíncrona frequentemente é usada em aplicativos da Web, em que várias solicitações de diferentes usuários podem ser processadas simultaneamente.*

Como isso é organizado?

- *O sistema operacional é o responsável por compartilhar seus recursos de CPU entre todas essas instâncias, não sendo necessário codificar esta etapa, mas para utilizar este recurso é preciso fazer algumas implementações importantes em seu código.*

Vamos ver agora um exemplo de um programa criado usando a biblioteca asyncio para desenvolvimento assíncrono e outro criado usando desenvolvimento sequencial. Ambos os programas simplesmente implementam as funções soma e multiplicação que somam e multiplicam as variáveis a e b, respectivamente.

Assíncrono

```
import asyncio
import time

async def soma(a,b):
  print("Iniciando soma")
  print(f"{a} + {b} = {a+b}")
  print("Finalizando soma")

async def multiplica(a,b):
  print("Iniciando multiplicacao")
  print(f"{a} * {b} = {a*b}")
  print("Finalizando multiplicacao")

inicio=time.time()
print("Iniciando programa")

await soma(3, 4)
await multiplica(5, 5)

print("Finalizando programa")
fim=time.time()
```

*O operador `await` é utilizado para aguardar as tarefas task 1 e task 2 finalizarem para que as últimas linhas do código sejam executadas. A função `tim.time()` foi utilizada para calcularmos o tempo de execução das duas tarefas, que resultou em **0.0031323432922363 28**.*

Sequencial:

```
def soma(a,b):
    print("Iniciando soma")
    print(f"{a} + {b} = {a+b}")
    print("Finalizando soma")


def multiplica(a,b):
    print("Iniciando multiplicacao")
    print(f"{a} * {b} = {a*b}")
    print("Finalizando multiplicacao")

inicio=time.time()
print("Iniciando programa")

task1 = soma(10,5)
task2 = multiplica(4,8)

print("Finalizando programa")
fim=time.time()

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

*A função tim.time() foi utilizada para calcularmos o tempo de execução das duas tarefas, que resultou em **0.0050432682037353516**.*

Lembre-se

Antes de começar a desenvolver algum programa observe pontos em que a tarefa pode vir a ter um tempo de execução alto e utilize algum recurso assíncrono para executá-la.