Programação Assíncrona Python

Programação Assíncrona

https://cursos.alura.com.br/course/praticando-python-programacao-assincrona

Síncrona vs Assíncrona

A principal diferença é que a Síncrona executa o código de cima para baixo, em ordem ou seja tem que executar linha a linha, já na Assíncrona pode se realizar n atividades simultaneamente.

Paralelismo vs Concorrência

Paralelismo usá vários nucleos concorrência alterna rapidamente entre atividades

A BIBLIOTECA ASYNCIO

biblioteca do python utilizada para a escrita de código assincrono com python

✓ Chamadas de API sem bloquear o programa.
✓ Leitura e escrita de arquivos ou banco de dados.
✓ Criar servidores e bots que precisam lidar com várias conexões simultaneamente.

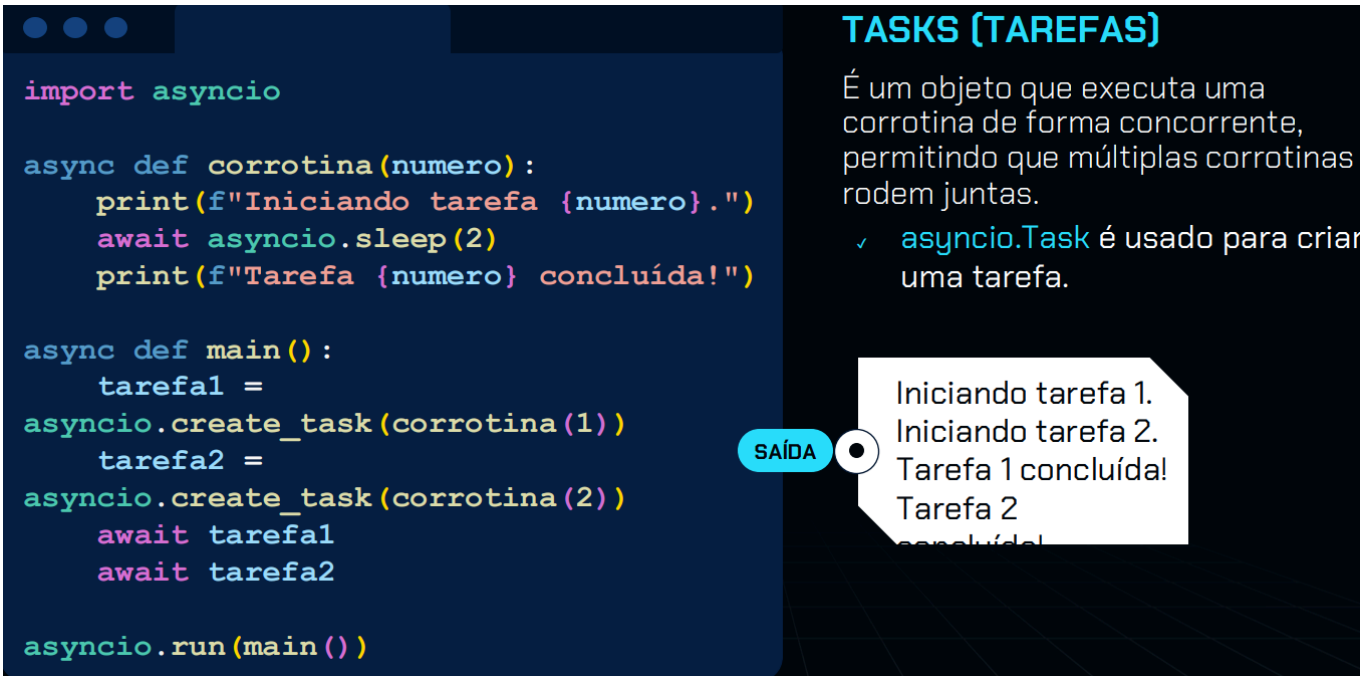
AWAITABLES (AGUARDÁVEIS)

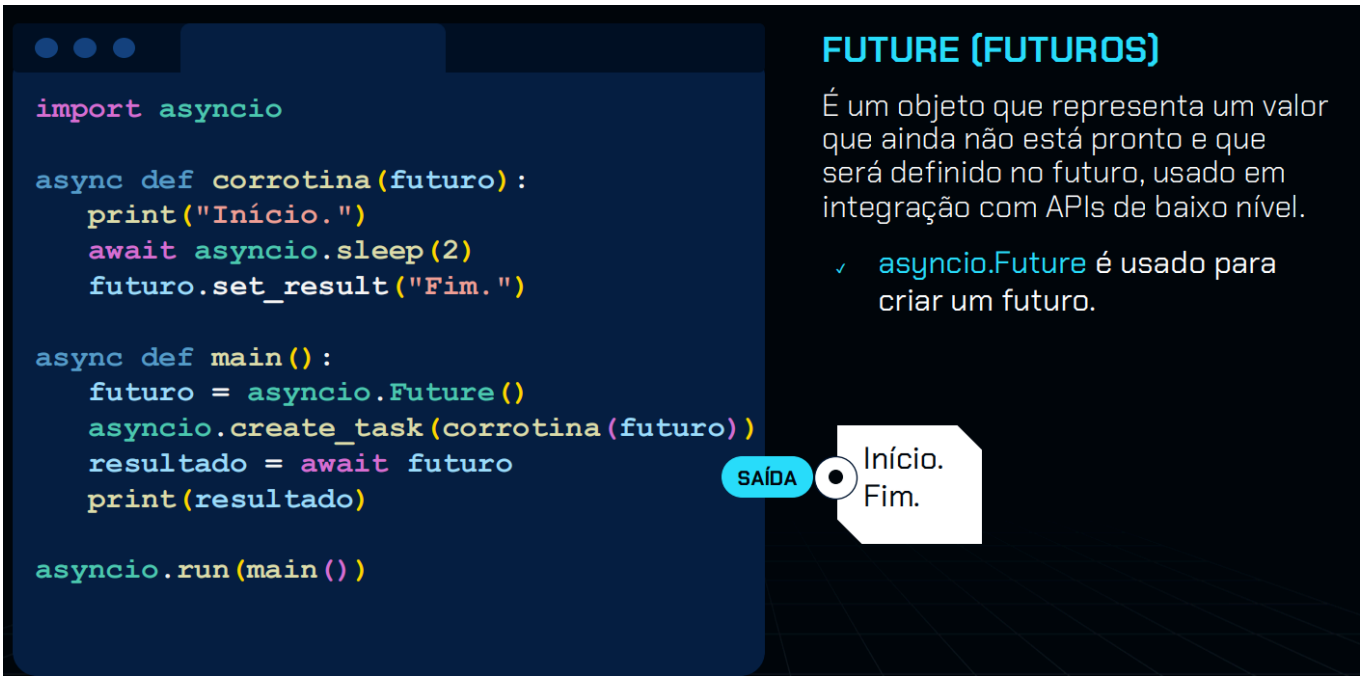
É qualquer objeto que pode ser esperado com o await. Existem 3 tipos:

✓ Corrotinas.
✓ Tarefas.
✓ Futuros.

```python
import asyncio

async def corrotina():
    print("Início.")
    await asyncio.sleep(2)
    print("Fim.")
asyncio.run(corrotina())
```





Aqui eu entendi que, ambas as tarefas são executadas, mas como a segunda espera o resultado futuro que é fornecido pela primeira, a primeira deve ser finalizada para dar procedimento a segunda

```python
import asyncio

async def corrotina1(futuro):
    print("Tarefa 1 iniciada.")
    await asyncio.sleep(2)
    futuro.set_result("Resultado da Tarefa 1")
    print("Tarefa 1 finalizada.")

async def corrotina2(futuro):
    print("Tarefa 2 iniciada, aguardando o futuro.")
    resultado = await futuro
    print("Tarefa 2 finalizada com o resultado:", resultado)

(...)
```

```python
(...)

async def main():
    futuro = asyncio.Future()
    tarefa1 = asyncio.create_task(corrotina1(futuro))
    tarefa2 = asyncio.create_task(corrotina2(futuro))

    await tarefa1
    await tarefa2

asyncio.run(main())
```

SAÍDA

```
Tarefa 1 iniciada.
Tarefa 2 iniciada, aguardando o futuro
Tarefa 1 finalizada.
Tarefa 2 finalizada com o resultado:
Resultado da Tarefa 1
```

EXECUTANDO MÚLTIPLAS TASKS

✓ asyncio.create_task()

✓ asyncio.gather()

ao invés de usar o create_task várias vezes é possível utilizar apenas o gather

nesse caso as 3 serão executadas, mas a ordem de finalização sera:

3 > 1 > 2

para executar códigos simultaneamente é necessário criar tasks

exemplo

tarefa1 = asyncio.create_task(funcao(1))
tarefa2 = asyncio.create_task(funcao(2))
await tarefa1
await tarefa2

https://cursos.alura.com.br/course/praticando-python-programacao-assincrona/task/187707