

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 assíncrono 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.

```
import asyncio

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

```
import asyncio

async def corrotina(numero):
    print(f"Iniciando tarefa {numero}.")
    await asyncio.sleep(2)
    print(f"Tarefa {numero} concluída!")

async def main():
    tarefa1 =
    asyncio.create_task(corrotina(1))
    tarefa2 =
    asyncio.create_task(corrotina(2))
    await tarefa1
    await tarefa2

asyncio.run(main())
```

TASKS (TAREFAS)

É um objeto que executa uma corrotina de forma concorrente, permitindo que múltiplas corrotinas rodem juntas.

- ✓ `asyncio.Task` é usado para criar uma tarefa.

SAÍDA

Iniciando tarefa 1.
Iniciando tarefa 2.
Tarefa 1 concluída!
Tarefa 2
concluída!

```
import asyncio

async def corrotina(futuro):
    print("Início.")
    await asyncio.sleep(2)
    futuro.set_result("Fim.")

async def main():
    futuro = asyncio.Future()
    asyncio.create_task(corrotina(futuro))
    resultado = await futuro
    print(resultado)

asyncio.run(main())
```

FUTURE (FUTUROS)

É um objeto que representa um valor que ainda não está pronto e que será definido no futuro, usado em integração com APIs de baixo nível.

- ✓ `asyncio.Future` é usado para criar um futuro.

SAÍDA

Início.
Fim.

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

```

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)

(...)

```

```

(...)

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`

```
import asyncio

async def corrotina(nome, tempo):
    print(f"Tarefa {nome} iniciada.")
    await asyncio.sleep(tempo)
    print(f"Tarefa {nome} concluída.")

async def main():
    await asyncio.gather(
        corrotina("1",2),
        corrotina("2",3),
        corrotina("3",1)
    )

asyncio.run(main())
```

SAÍDA

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

3 > 1 > 2

```
import time

def tarefa(numero):
    print(f"Iniciando tarefa {numero}.")
    time.sleep(2)
    print(f"Tarefa {numero} concluída!")

tarefa(1)
tarefa(2)
tarefa(3)
```

PROJETO SÍNCRONO

SAÍDA

Iniciando tarefa 1.
Tarefa 1
concluída!
Iniciando tarefa 2.
Tarefa 2
concluída!



```
import asyncio
```

```
async def main():
    await asyncio.gather(tarefa(1),
tarefa(2), tarefa(3))
```

SAÍDA

Iniciando tarefa 1.
Iniciando tarefa 2.
Iniciando tarefa 3.
Tarefa 1 concluída!
Tarefa 2
concluída!

para executar códigos simultaneamente é necessário criar tasks

```
tarefa1 = asyncio.create_task(funcao(1))
```

await tarefa1

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