

```
import time

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

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

## PROGRAMAÇÃO SÍNCRONA

Modelo de execução em que as tarefas são realizadas uma após a outra, de forma **sequencial**. Cada operação precisa ser concluída antes que a próxima inicie.

SAÍDA

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

# PROGRAMAÇÃO ASSÍNCRONA

Permite que um programa execute **várias tarefas ao mesmo tempo**, sem que uma precise esperar a outra terminar.

## CONCORRÊNCIA

≠

## PARALELISMO

**Alternância rápida entre tarefas**, dando a impressão de que rodam ao mesmo tempo.



Tarefas de I/O (API, banco de dados).

Múltiplas tarefas **são executadas ao mesmo tempo**.



Cálculos pesados e processamento de dados.



## SÍNCRONO

O garçom atende **um cliente por vez**, só pegando o próximo pedido quando terminar o primeiro.

## PARALELISMO

**Vários garçons atendem clientes simultaneamente** em mesas diferentes.

## ASSÍNCRONO

O garçom anota o pedido, passa para a cozinha e **atende outro cliente enquanto espera** a comida ficar pronta.

## CONCORRÊNCIA

O garçom pega pedidos de **vários clientes e alterna entre eles**, mas continua sozinho no trabalho.



# QUANDO USAR?

## SÍNCRONA

≠

## ASSÍNCRONA

- ✓ Fluxo do programa é simples e sequencial.
- ✓ A lógica do código depende da execução ordenada das operações.
- ✓ Há pouca ou nenhuma espera por I/O.
- ✓ O código é puramente computacional e pesado.
- ✓ A simplicidade do código é mais importante que a velocidade.

- ✓ Há múltiplas tarefas que dependem de I/O.
- ✓ Lidar com várias conexões simultâneas.
- ✓ O sistema precisa melhorar o tempo de resposta.
- ✓ Tarefas podem ser executadas paralelamente sem dependerem uma das outras.



# EXEMPLO DE USO: ENVIO DE E-MAIL DE CONFIRMAÇÃO

## SÍNCRONA


≠

## ASSÍNCRONA

- ✓ Se o e-mail for enviado e o usuário precisa esperar a resposta de confirmação antes de continuar acessando o sistema.

- ✓ Se o e-mail for enviado em segundo plano enquanto o usuário segue usando o sistema.

# A BIBLIOTECA **ASYNCIO**

 Biblioteca **padrão** do Python para escrita de código assíncrono, usada para operações como:

- ✓ 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.



```
import asyncio

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

asyncio.run(corrotina())
```

## AWAITABLES (AGUARDÁVEIS)

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

- ✓ Corrotinas.
- ✓ Tarefas.
- ✓ Futuros.

SAÍDA

Início.  
Fim.



```
import asyncio

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

asyncio.run(corrotina())
```

## COROUTINES (CORROTINAS)

É uma função especial que pode ser pausada e retomada durante sua execução.

- ✓ `async def` define a função como corrotina.
- ✓ `await` pausa a execução para aguardar outra corrotina.

SAÍDA

Início.  
Fim.





```
import asyncio

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

async def main():
    await corrotina(1)
    await corrotina(2)

asyncio.run(main())
```

SAÍDA

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



```
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



```
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.



```
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



```
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())
```

## EXECUTANDO MÚLTIPLAS TASKS

Você pode executar várias tarefas assíncronas utilizando as funções:

- ✓ `asyncio.create_task()`
- ✓ `asyncio.gather()`

SAÍDA

Tarefa 1 iniciada.  
Tarefa 2 iniciada.  
Tarefa 3 iniciada.  
Tarefa 3  
concluída.  
Tarefa 1 concluída.



## PROJETO SÍNCRONO

```
import time

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

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

SAÍDA

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

## PROJETO ASSÍNCRONO

```
import asyncio

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

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

asyncio.run(main())
```

SAÍDA

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





Para conhecer mais sobre a biblioteca asyncio, consulte a [documentação oficial](#) do Python.



Compartilhe um resumo de seus novos  
conhecimentos em suas redes sociais.  
**#aprendizadoalura**