JavaScript

Callbacks e paradigma assíncrono

JS

JavaScript

parte 10

Alan Alencar, 2023

aprendendo Javascript

agenda

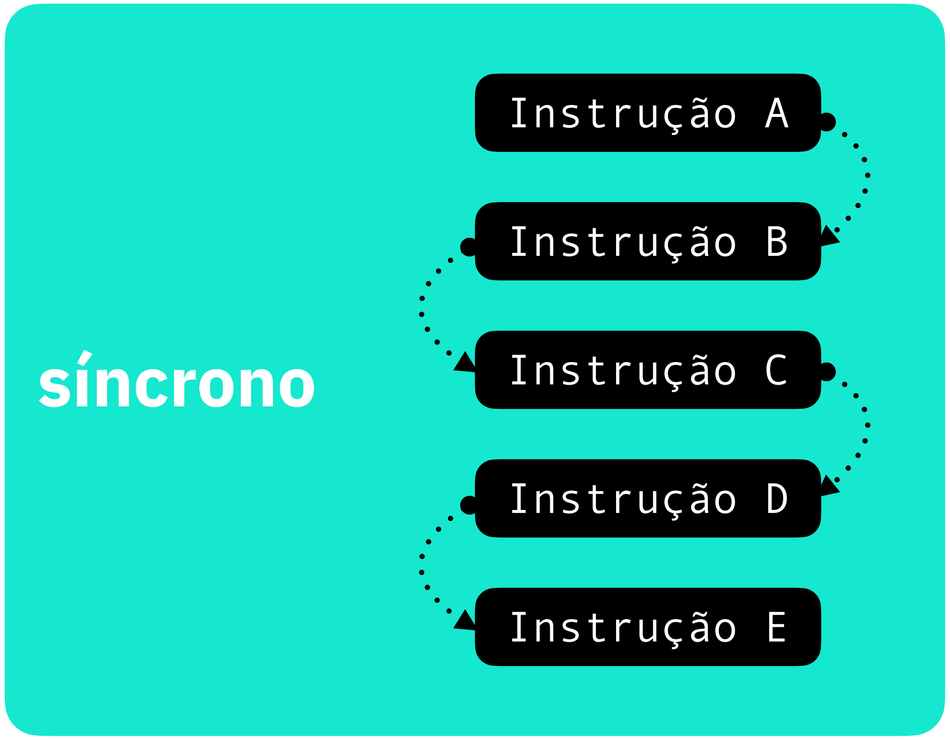
as Callbacks functions são bastante frequentes em JavaScript tornando-a uma poderosa linguagem para criar algoritmos rápidos e muito consistentes. Vamos de agora para frente explorar o potencial das callbacks functions em nosso aprendizado.

como são executados os códigos

paradigma **assíncrono**

um código escrito em Javascript é *normalmente* executado de cima para baixo, na ordem em que foram escritas as instruções, considerando por óbvio seu escopo de código, laços e decisões.

sendo assim, de forma lúdica, cada instrução vai ser executada uma após a outra, sendo que a instrução seguinte, vai aguardar sua antecessora finalizar seu trabalho, ainda que isto dure segundos, minutos ou horas.



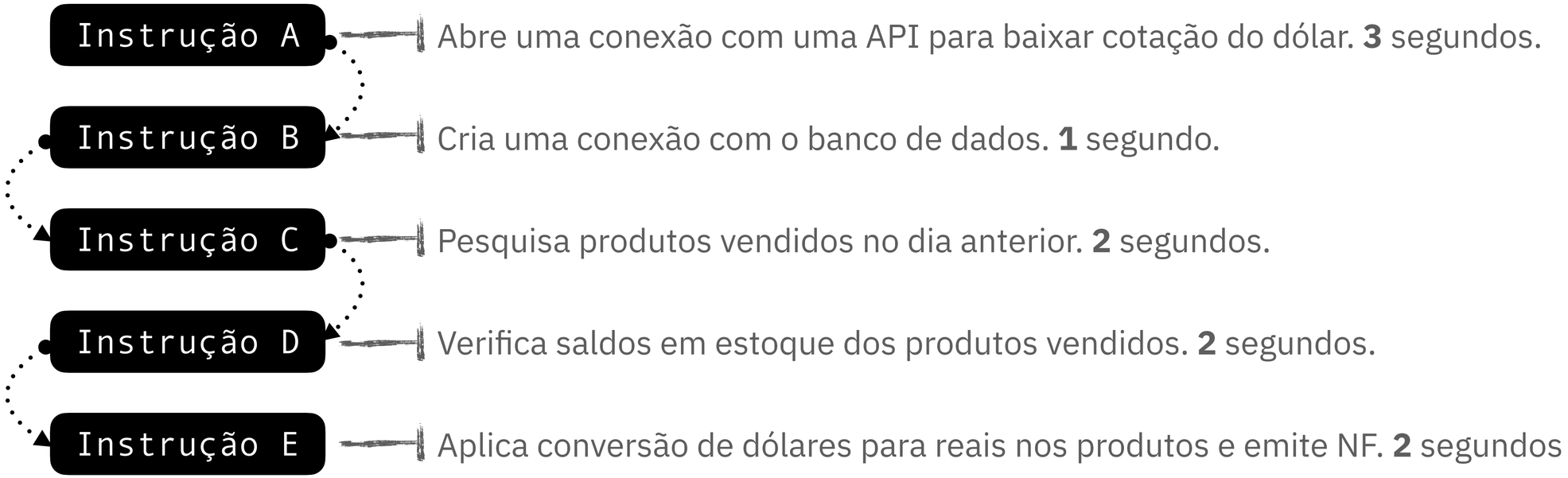
uma única thread é responsável por executar o código do JavaScript, instrução por instrução. Para que a instrução **B** seja executada, a instrução **A** tem que finalizar seu processamento e assim até o fim das instruções.

esta abordagem comum em linguagens de programação é funcional, porém se uma instrução é demasiado demorada, todo o restante do processamento que nem necessita dela fica comprometido, denegrindo a eficiência no algoritmo.

como são executados os códigos

paradigma **assíncrono**

síncrono



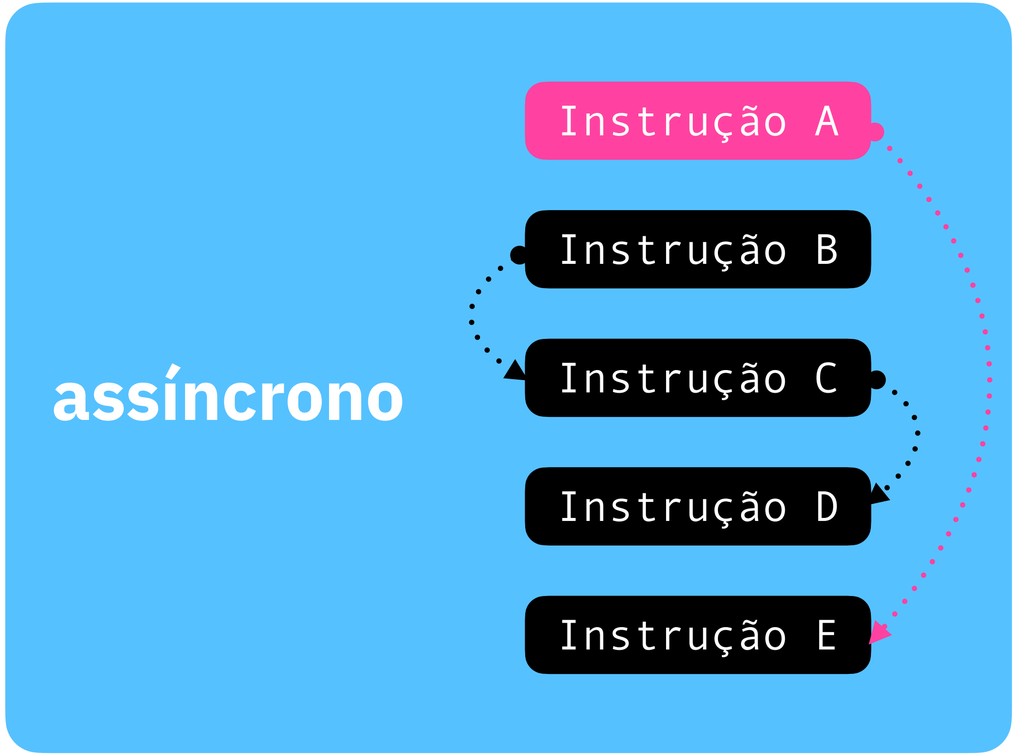
10
segundos

é o tempo final de execução desta rotina em 99% das implementações, que consideram sempre execuções síncronas. 1.000 requisições dia, estamos falando de 10.000 segundos ou 2hr42min aproximadamente de processamento.

como são executados os códigos

paradigma **assíncrono**

o mesmo algoritmo usando o paradigma da programação assíncrona que o Javascript tem nativamente, vai permitir olhar para este ciclo de execução com outros olhos e fazê-lo mais eficiente, ainda mais considerando o volume de execuções diárias.



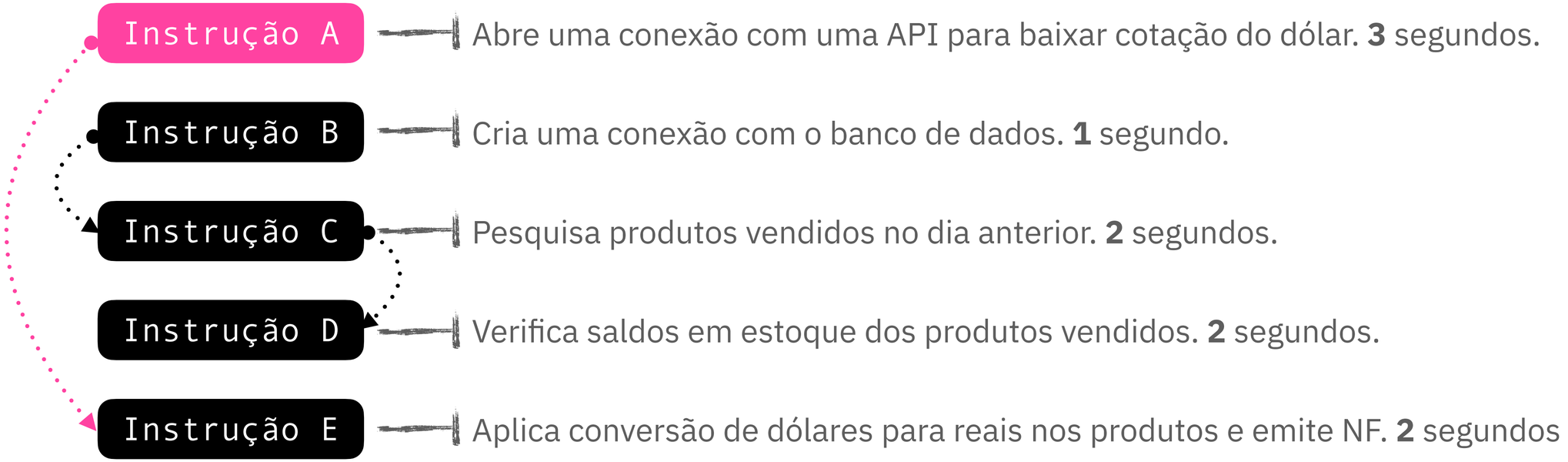
a thread do Javascript continua sendo única, porém a instrução **A**, por ter uma característica específica será entregue para outra thread e não será impeditiva para a instrução **B** executar como no modelo síncrono. Ao final a instrução **A** vai chamar a instrução **E** como callback e vai finalizar o algoritmo, gerando mais eficiência.

nesta nova abordagem, tudo fica mais performático já que eu não preciso aguardar o término da instrução A que vai demorar para iniciar a instrução B, depois a C e D. Já a instrução E será chamado pela instrução A assim que ela for finalizada.

como são executados os códigos

paradigma **assíncrono**

assíncrono



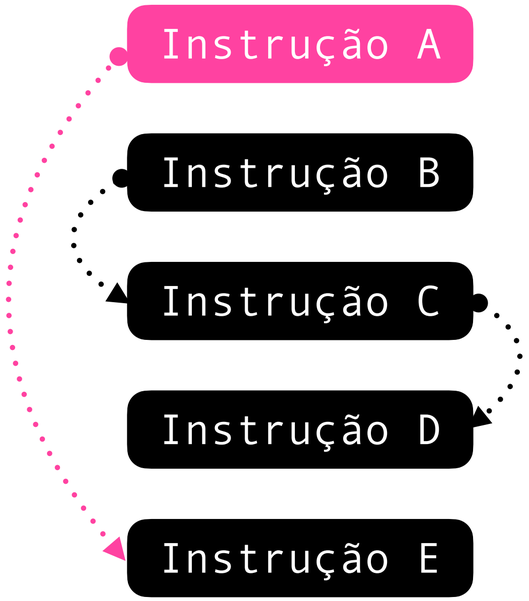
7
segundos

*nesta abordagem ganhamos 30% de eficiência em relação do modelo síncrono . 1.000 requisições dia, estamos falando de 7.000 segundos ou 1hr56min aproximadamente de processamento, ou **46** minutos a menos.*

como são executados os códigos

paradigma **assíncrono**

assíncrono



7
segundos

síncrono

10
segundos

o que são callbacks functions

como fazer uso delas

de uma maneira bem simplista, funções de callbacks nada mais é que uma função sendo passado como parâmetro para outra função.

É bem comum em JavaScript métodos de objetos globais receberem como argumento uma *callback function* para que ela possa ser chamada sempre que algum comportamento acontecer. Vale relembrar que JavaScript é uma linguagem que utiliza do paradigma do processamento **assíncrono**. É, aqui tudo acontece um pouco diferente das linguagens mais tradicionais, que claro, implementam também oportunidade de processos assíncronos mas com um grau de complexidade alto, coisa que em JavaScript não é necessário desenvolver.

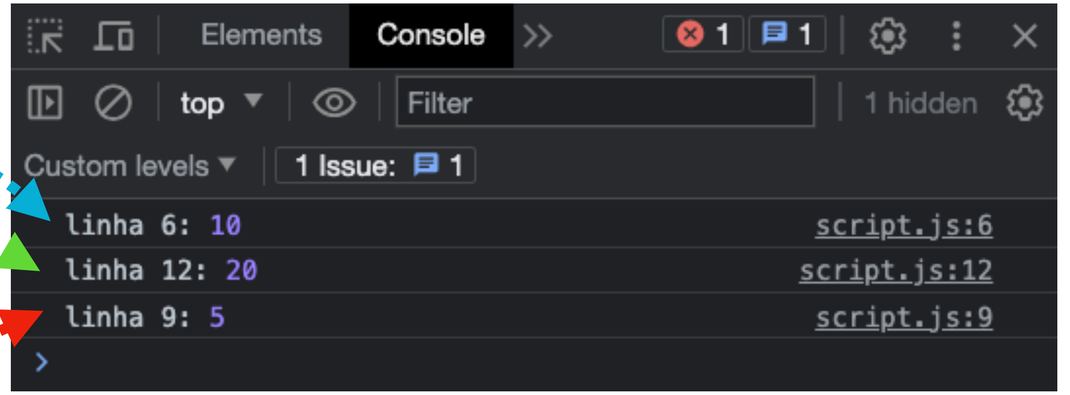
mas antes de entrarmos mais a fundo nas callbacks functions vamos ver um exemplo clássico de processamento **assíncrono** em JS.

```
1 /**
2  * script.js
3  * código assíncrono com javascript.
4  */
5 let foo = 10; // iniciando com valor 10, number.
6 console.log('linha 6:', foo);
7 setTimeout(() => {
8     foo = 5;
9     console.log('linha 9:', foo);
10 }, 3000);
11 foo *= 2;
12 console.log('linha 12:', foo);
```

funções assíncronas

podem nos confundir num primeiro instante

```
1  /**
2   * script.js
3   * código assíncrono com javascript.
4   */
5  let foo = 10; // iniciando com valor 10, number.
6  console.log('linha 6:', foo);
7  setTimeout(() => {
8      foo = 5;
9      console.log('linha 9:', foo);
10 }, 3000);
11 foo *= 2;
12 console.log('linha 12:', foo);
```



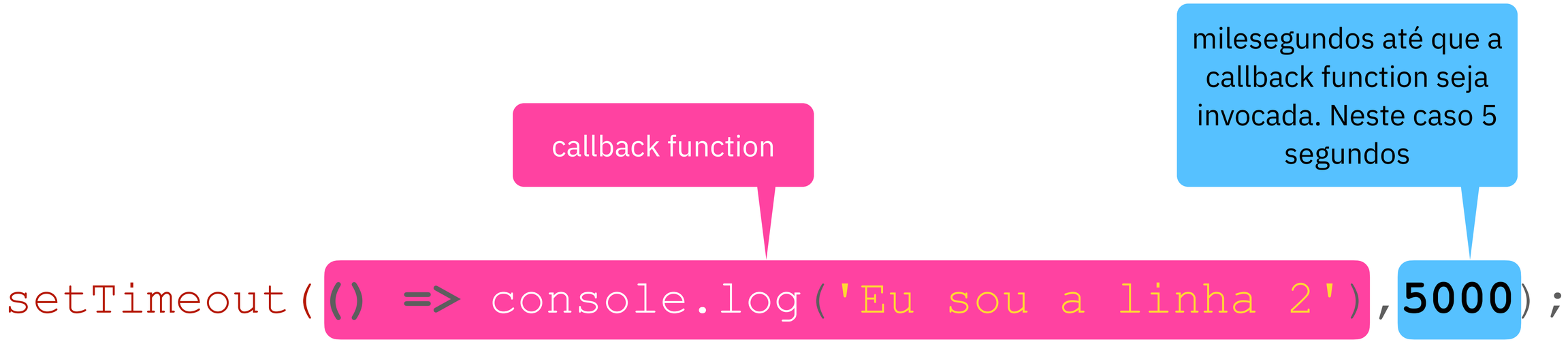
dissecando

callback function

```
setTimeout(callback function , milissegundos);
```

dissecando

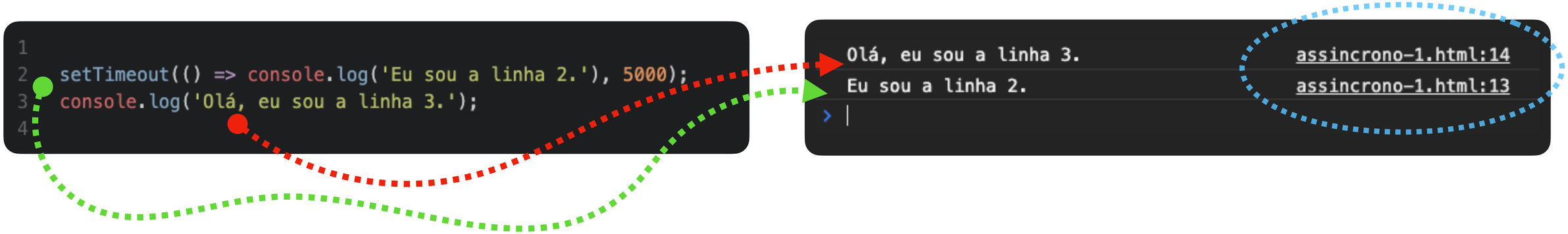
callback function



A callback function passada como parâmetro para a função `setTimeout` neste exemplo é uma **arrow function** mas poderia ser uma **function expression** ou **function declaration**.

executando as callbacks

como fazer uso delas



apesar da instrução que deveria exibir “**Eu sou a linha 2**” ter sido processada antes, ela foi colocada na pilha de execuções do JavaScript que entendeu através da instrução `setTimeout()` que ela deveria ser executada somente após passar 5 segundos. Para não ficar travado aguardando este tempo desnecessariamente, o JavaScript seguiu executando o código e portanto ecoou a instrução da linha três primeiro.

mais um exemplo

usando callbacks em chamadas síncronas

não é mandatório que se tenha operações assíncronas para usar callbacks. Ainda que ela seja importantíssima por lá, também pode ser usado em situações interessantes onde a chamada é síncrona.

a função *qualSeuNome* recebe como parâmetro uma outra função, neste caso *digaOla* que será invocada dentro do escopo dela.

este recurso pode ser muito interessante quando você precisa isolar um determinado comportamento e durante a execução de um longo trecho de código, necessite síncrono ou assincronamente chamá-lo.

```
/**
 * função digaOla que será usada como callback
 */
const digaOla = (nome) => {
    console.log(`Olá ${nome}, seja bem-vindo(a)!!!`);
}

/**
 * Recebe a função DigaOla como parâmetro de callback
 * @param {*} callback
 */
const qualSeuNome = (callback) => {
    const nome = prompt('Digite seu nome');
    callback(nome);
}

qualSeuNome(digaOla);
```


VAMOS PRATICAR
programming

revendo a prática

princípio básico

1. aprendemos o que são **callback** functions em JavaScript.
2. apesar de usadas em comportamento **assíncronos**, também é possível usar callbacks em situações **síncronas**.
3. é uma forma muito poderosa de escrever **códigos e algoritmos** complexos em JavaScript.