
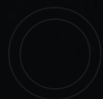




SHIFT

 FIAP



000000



FRONT-END SPECIALIST

DESENVOLVIMENTO WEB COM ANGULAR & REACT

LÓGICA DE PROGRAMAÇÃO

JAVASCRIPT



ALEX SANDER RESENDE DE DEUS

Professor

- Há 25 anos ensinando programação, é um apaixonado por tecnologia. Atualmente é coordenador de cursos na ETEC Albert Einstein do Centro Paula Souza.
- Na FIAP é professor da FIAP School e FIAP Corporate, lecionando C#, Banco de Dados e Desenvolvimento Mobile.

✉ profalex.deus@fiap.com.br

AGENDA

1

AULA 1

Definição e histórico

Fundamentos Básicos da Programação – Tipos de dados, variáveis e estruturas básicas; Usando o GIT

2

AULA 2

Estruturas condicionais, operadores lógicos e switch case

3

AULA 3

Laços de Repetição: For, while,

4

AULA 4

Objetos nativos JavaScript – Date, String, Math

AGENDA

5

AULA 5

Arrays, Filter, Map

Reduce, Split, forEach, for in, for of

6

AULA 6

Funções: tipos, declarações, escopo, retorno, closures, call-back, IIFE, factory

7

AULA 7

DOM – Document Object Model

8

AULA 8

DOM – Document Object Model

AGENDA

9

AULA 9

Eventos

Objetos: Criação, prototype, getters, setters



10

AULA 10

Classes JavaScript Assíncrono



AULA 10

JAVASCRIPT ASSÍNCRONO





Exemplo 1) Eu vou a uma pizzeria, peço uma pizza para viagem no balcão e fico plantado lá, só esperando me entregarem o pedido para que eu possa ir embora.

Exemplo 2) Eu vou a uma pizzeria, peço uma pizza para viagem no balcão e, enquanto ela não fica pronta, dou uma passada na livraria ao lado para folhear algum livro ou revista.





Se você conhece os conceitos básicos de assincronismo, é fácil compreender que o exemplo 1 é um caso típico de operação síncrona, pois minha vida parou completamente até que o pizzaiolo completasse seu trabalho. No exemplo 2, por outro lado, resolvi aproveitar meu precioso tempo ocioso aproveitando um bom livro.



Em um código **síncrono**, todas as funções e requisições trabalham em sincronia, em um contato direto, do início ao fim da comunicação. Dessa maneira, esse código só permite uma requisição por vez.


Por exemplo: se fizermos uma requisição para uma API, precisamos esperar a sua resposta. Até aí tudo bem. Mas, imagine que tenhamos mais de uma requisição para fazer. Com um código síncrono quaisquer outras requisições além da principal são **bloqueadas** até que a primeira **termine**. Em outras palavras, precisamos esperar a resposta da primeira requisição para só então ir para a próxima. Isso poderia afetar a experiência do usuário, deixando nosso site um tanto monótono.



Uma requisição **assíncrona** é diferente. Ela não trabalha em sincronia. Dessa forma, nós podemos realizar várias requisições ao mesmo tempo, e uma requisição não irá afetar a outra. Isso é impressionante, e ajuda muito na performance e na experiência do usuário também. Em poucas palavras, com um código assíncrono, nosso site se **torna mais dinâmico**.

Em JavaScript, as funções são executadas na sequência em que são chamadas:

```
<> index.html > html > body
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4
5  <head>
6    <title>JavaScript DOM</title>
7  </head>
8
9  <body>
10   <p id="demo"></p>
11   <script type="text/JavaScript" src="js/script.js"></script>
12 </body>
13
14 </html>
```

```
> JS script.js >  segundaFuncao
1  function exibir(valor) {
2      document.getElementById("demo").innerHTML = valor;
3  }
4
5  function primeiraFuncao() {
6      exibir("Olá");
7  }
8
9  function segundaFuncao() {
10     exibir("Adeus");
11 }
12
13 primeiraFuncao();
14 segundaFuncao();
```



Armazenamento x



Lógica - Java



Arquivo | C:/User

Adeus

CALLBACK

CallBack é uma função passada como um argumento para outra função.

```
<> index.html > html > head
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4  <head>
5      <title>JavaScript DOM</title>
6  </head>
7
8  <body>
9      <p id="demo"></p>
10     <script type="text/JavaScript" src="js/script.js"></script>
11 </body>
12
13 </html>
```

> JS script.js > ...

```
1  function exibir(valor) {  
2      document.getElementById("demo").innerHTML = valor;  
3  }  
4  
5  function calcular(num1, num2, callback) {  
6      let soma = num1 + num2;  
7      callback(soma);  
8  }  
9  
10 calcular(5, 5, exibir);
```



Armazenamento



Lógica - Ja



Arquivo

C:/User


10

ESPERANDO POR UM TEMPO LIMITE

Ao usar a função JavaScript `setTimeout()`, você pode especificar uma função de callback a ser executada no tempo limite

```
<> index.html > html
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4  <head>
5  |   <title>JavaScript DOM</title>
6  </head>
7
8  <body>
9  |   <p>Esta página será alterada em 3 segundos.</p>
10 |   <h1 id="demo"></h1>
11 |   <script type="text/JavaScript" src="js/script.js"></script>
12 </body>
13
14 </html>
```

ESPERANDO POR UM TEMPO LIMITE

```
js > JS script.js >  minhaFuncao
```

```
1   setTimeout(minhaFuncao, 3000);
```

```
2
```

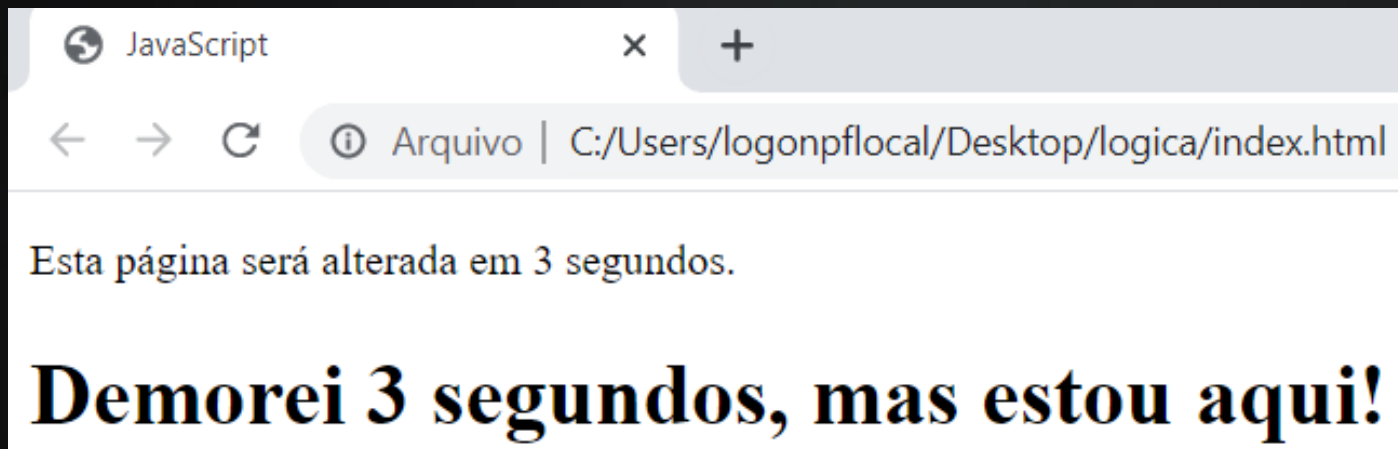
```
3   function minhaFuncao() {
```

```
4     document.getElementById("demo").innerHTML = "Demorei 3 segundos, mas estou aqui!";
```

```
5
```

```
   }
```

ESPERANDO POR UM TEMPO LIMITE



ESPERANDO POR INTERVALOS:

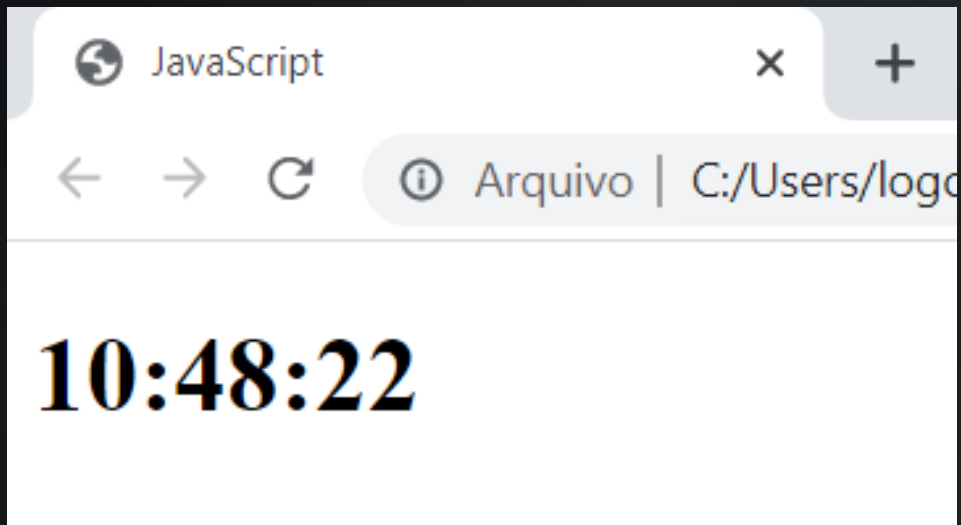
Ao usar a função JavaScript `setInterval()`, você pode especificar uma função de callback a ser executada para cada intervalo:

```
<> index.html > html
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4
5  <head>
6  |   <title>JavaScript</title>
7  </head>
8
9  <body>
10 |   <h1 id="demo"></h1>
11 |   <script type="text/JavaScript" src="js/script.js"></script>
12 </body>
13 </html>
```

ESPERANDO POR INTERVALOS:

```
> JS script.js > ...  
1   setInterval(minhaFuncao, 1000);  
2  
3   function minhaFuncao() {  
4       let d = new Date();  
5       document.getElementById("demo").innerHTML=  
6       d.getHours() + ":" +  
7       d.getMinutes() + ":" +  
8       d.getSeconds();  
9   }
```


ESPERANDO POR INTERVALOS:



PROMISES

Promise é um objeto usado para processamento assíncrono. Um Promise (de "promessa") representa um valor que pode estar disponível agora, no futuro ou nunca.

Um Promise está em um destes estados:

pending (pendente): Estado inicial, que não foi realizada nem rejeitada.

fulfilled (realizada): sucesso na operação.

rejected (rejeitado): falha na operação.

Uma promessa **pendente** pode se tornar **realizada** com um valor ou **rejeitada** por um motivo (erro). Quando um desses estados ocorre, o método **then** do Promise é chamado, e ele chama o método de tratamento associado ao estado (**rejected** ou **resolved**). Se a promessa for realizada ou rejeitada quando o método de tratamento correspondente for associado, o método será chamado. Desta forma, não há uma condição de competição entre uma operação assíncrona e seus manipuladores que estão sendo associados.

index.html > html > meta

```
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4  <head>
5    <title>JavaScript</title>
6  </head>
7  <body>
8    <h2>JavaScript Promise</h2>
9    <p id="demo"></p>
10   <script type="text/JavaScript" src="js/script.js"></script>
11 </body>
12
13 </html>
```

```
js > JS script.js > [x] myPromise > <function> > [x] x
1  function exhibir(valor) {
2      document.getElementById("demo").innerHTML = valor;
3  }
4
5  let myPromise = new Promise(function (myResolve, myReject) {
6      let x = 0; // experimente trocar o valor de x para 5
7      if (x == 0) {
8          myResolve("OK");
9      } else {
10         myReject("Error");
11     }
12 });
13
14 myPromise.then(
15     function (value) { exhibir(value); },
16     function (error) { exhibir(error); }
17 );
```

ESPERANDO POR UM TEMPO LIMITE COM PROMISE

```
<> index.html > html > body
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4  <head>
5  |   <title>JavaScript</title>
6  </head>
7  <body>
8  |   <h2>JavaScript Promise</h2>
9  |   <p>Esta página será atualizada em 3 segundos</p>
10 |   <h1 id="demo"></h1>
11 |   <script type="text/JavaScript" src="js/script.js"></script>
12 </body>
13
14 </html>
```

ESPERANDO POR UM TEMPO LIMITE COM PROMISE

```
> JS script.js > [🔗] myPromise > 📦 <function> > 📦 setTimeout() callback
1  const myPromise = new Promise(function(myResolve, myReject) {
2      setTimeout(function(){ myResolve("Após 3 segundos, apareci!"); }, 3000);
3  });
4
5  myPromise.then(function(value) {
6      document.getElementById("demo").innerHTML = value;
7  });
```


ASYNC E **AWAIT**

ASYNC

index.html > html > body

```
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4  <head>
5  |   <title>JavaScript</title>
6  </head>
7  <body>
8  |   <h2>JavaScript async / await</h2>
9  |   <p id="demo"></p>
10 |   <script type="text/JavaScript" src="js/script.js"></script>
11 </body>
12 </html>
```

ASYNC


```
> JS script.js > ...
1  function exibir(valor) {
2      document.getElementById("demo").innerHTML = valor;
3  }
4
5  async function minhaFuncao() { return "Olá!!!!!!"; }
6
7  minhaFuncao().then(
8      function (value) { exibir(value); },
9      function (error) { exibir(error); }
10 );
```

AWAIT

index.html > html > body

```
1  <!DOCTYPE html>
2  <html>
3  <meta charset="UTF-8">
4  <head>
5    <title>JavaScript</title>
6  </head>
7  <body>
8    <h2>JavaScript async / await</h2>
9    <h1 id="demo"></h1>
10   <script type="text/JavaScript" src="js/script.js"></script>
11 </body>
12 </html>
```

AWAIT

s > JS script.js >  exhibir

```
1  async function exhibir() {  
2      let myPromise = new Promise(function (resolve, reject) {  
3          resolve("Olá!!!!");  
4      });  
5      document.getElementById("demo").innerHTML = await myPromise;  
6  }  
7  exhibir();
```

MOMENTO **HANDS ON**

Elabore o JavaScript que leia 2 notas de um aluno, calcule e exiba a média. Exiba também a mensagem APROVADO quando a média for maior ou igual a 6. Caso contrário, exiba RECUPERAÇÃO.


Obs. Vamos fazer este script de 3 formas diferentes: usando **Promise**, **Async** e **Await**

LEMBRE-SE DE VERSIONAR
SEUS PROJETOS NO GIT

BIBLIOGRAFIA

- <https://developer.mozilla.org/pt-BR/docs/Aprender/JavaScript>
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/String
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Math
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array
- https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM
- <https://www.w3schools.com/>
- <http://docplayer.com.br/17393758-Javascript-eventos-e-objetos-nativos.html>
- <https://www.adalgisa-souza.appspot.com/javascript/>
- <https://blog.schoolofnet.com/tudo-o-que-voce-precisa-saber-sobre-classes-no-javascript/>

OBRIGADO

 profalex.deus@fiap.com.br

 /alexanderresende

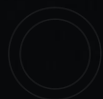
FIAP

Copyright © 2020 | Alex Sander Resende de Deus

Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento,
é expressamente proibido sem consentimento formal, por escrito, do professor/autor.

SHIFT

 FIAP



000000