

# **Análise e Desenvolvimento de Sistemas**

## **Frameworks Web I**

### Aula 01 - Revisão de JavaScript

---

---



# Tópicos

- **Revisão de JS**

- JavaScript e ECMAScript
- Transpilers
- Declaração de variáveis
- Operador ternário
- Arrow functions
- Operador spread
- Métodos map(), filter() e find()
- Template Literals
- Atribuição via desestruturação
- Import e Export
- Sintaxe JSX
- Estruturas de seleção
- Estruturas de repetição
- Classes



Adaptado de:

SILVA, Maurício Samy. **React - Aprenda praticando.** Novatec. 2021.

PINHO, Diego. **Let, const e var não é tudo a mesma coisa?** 2018. Disponível em: <https://tinyurl.com/2c66d7kh>. Acesso em: 7 ago. 2024.

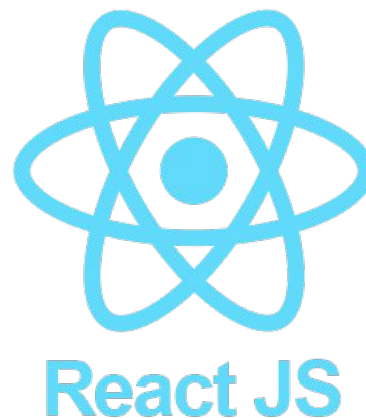


PINHO, Diego. **Let, const e var não é tudo a mesma coisa?** 2018. Disponível em: <https://tinyurl.com/2c66d7kh>. Acesso em: 7 ago. 2024.

# Revisão JavaScript

# Revisão

Algumas funcionalidades da linguagem JavaScript são usadas com frequência no desenvolvimento com React. Por isso é importante revisarmos alguns tópicos :)



# JavaScript e ECMAScript

# Revisão



JavaScript foi criada em 1995 por Brendan Eich, que trabalhava para a Netscape.

- Em 1996, a Netscape submeteu a linguagem à ECMA (Associação dedicada à padronização de sistemas de informação), para ser padronizada para uso por fabricantes de navegadores.
- Em 1997 foi publicada a ECMA262 e, com base nela, a ECMAScript1, primeira versão ECMA da JavaScript.



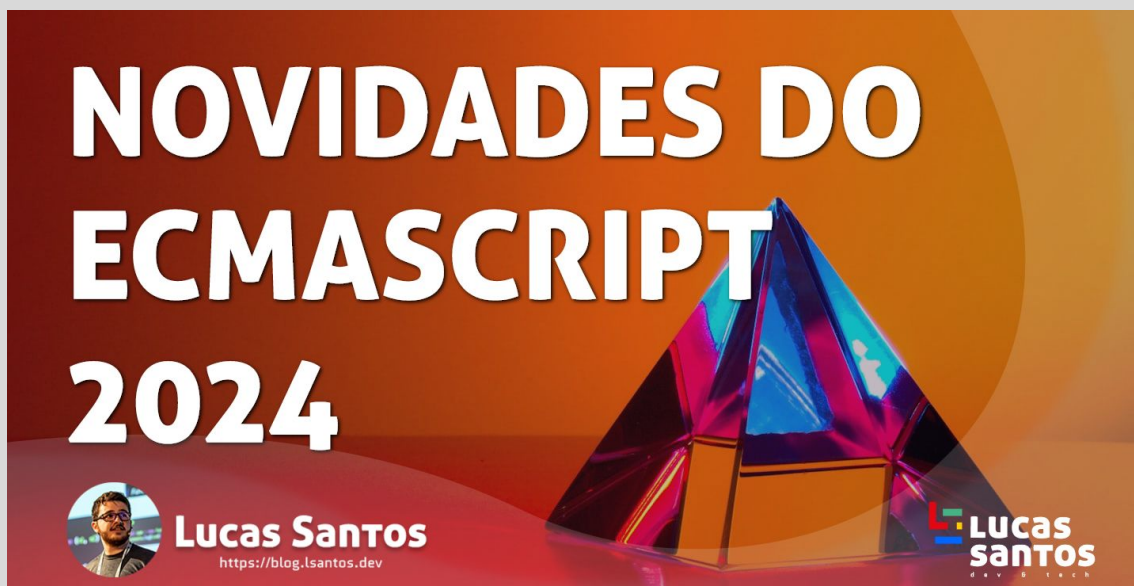
# Revisão

À época em que foi criada, o nome JavaScript era patente da Sun Microsystems (hoje Oracle).

- Quando da normatização da linguagem, ela foi renomeada para ECMAScript. Assim, os dois termos se referem exatamente à mesma linguagem.
- Em 1997 foi publicada a ECMA262 e, com base nela, a ECMAScript1, primeira versão ECMA da JavaScript.
- O termo que nos acostumamos a usar é JavaScript, mas o oficial é ECMAScript. Nos dias atuais já se usa com muito mais frequência a forma abreviada ESversão.



# Revisão



[Leia mais aqui :\)](https://blog.lsanatos.dev)

# Transpilers

# Revisão

Transpilers são compiladores do tipo código-para-código capazes de entender o código escrito em uma linguagem e produzir o código equivalente em outra.

- Transpilers JavaScript traduzem código não entendido, ou seja, não normalizado pela ECMA e, portanto, não suportado por dispositivos que entendem JavaScript.
- O mais conhecido e usado transpiler JavaScript é o Babel:  
<https://babeljs.io/>



# Declaração de variáveis

# Revisão

Antes da ES6, a palavra-chave para se declarar uma variável era `var`. A ES6 criou mais duas palavras-chaves para se declararem variáveis: `const` e `let`.

Recomenda-se usar `const` e `let` e evitar o uso de `var` para declarar variáveis

- A palavra-chave `const` (abreviatura de constante) destina-se a declarar variáveis que devem permanecer fixas no script. Uma vez declaradas, qualquer tentativa posterior de alterá-la ou reatribuí-la resulta em erro.
- A palavra-chave `let` destina-se a declarar variáveis que não precisam permanecer fixas, mas cujo escopo limita-se ao local onde foram declaradas

# Revisão

Como era utilizando var

```
1  var numero = 1; // ok
2  var string = "1"; // ok
3  var olaMundo = function() {
4    |  | console.log("Olá mundo!");
5    } // também ok!
```

# Revisão

## Utilizando let

O `let` é muito semelhante ao `var`, no sentido de que podemos instanciar e armazenar qualquer tipo de objeto nelas.

```
1  let numero = 1; // ok
2  let string = "1"; // ok
3  let olaMundo = function() {
4    | console.log("Olá mundo!");
5  } // também ok!
```

# Revisão

## Utilizando let

Uma vez criadas as variáveis, quando utilizamos o `let` os seus valores podem ser sobrescritos, até mesmo por outros tipos de dados como, por exemplo:

```
1  let numero = 1;  
2  numero = "1";  
3  
4  console.log(numero); // 1
```



# Revisão

## Utilizando const

O `const` também funciona de forma análoga aos demais, entretanto, uma vez que o valor foi atribuído à variável, ele não pode ser modificado.

```
1  const numero = 1;
2  numero = "1";
3
4  console.log(numero); // 1
```

Ao tentar executar este código, temos um resultado semelhante a este:

Console × ...

```
Cannot assign to "numero" because it script.js? 2:0
is a constant
```

# Revisão

## Utilizando const

Porém, podemos fazer isso mostrado abaixo, pois a referência continua a mesma. Por isso conseguimos inserir mais propriedades no objeto.

```
1  const pessoa = {  
2    nome: 'Diego',  
3    idade: 26  
4  }  
5  
6  pessoa.sobrenome = 'Pinho';  
7  
8  console.log(pessoa);
```

### Console ×

```
▼ (3) {nome: "Diego", idade: 26, sobrenome...}  
  nome: "Diego"  
  idade: 26  
  sobrenome: "Pinho"
```

# Revisão

## Const e let x var

Talvez, até agora você não tenha percebido muitas diferenças entre o `let` e o `var`. Mas, vamos analisar o código abaixo:

```
1  var mensagem = "olá";  
2  ▼ {  
3      var mensagem = "adeus"  
4      console.log(mensagem);  
5  }  
6  console.log(mensagem);
```

# Revisão

Qual você acha que será o resultado da execução deste código?

- a) olá olá
- b) olá adeus
- c) adeus olá
- d) adeus adeus

Se você respondeu qualquer alternativa que não seja a “d”, você errou

# Revisão

## Por que?

Porque o var possui o que chamamos de `escopo de função`.

- Em termos práticos, significa que `dentro de uma mesma função, a referência da variável é a mesma`.
- Como não temos nenhuma distinção de função neste código, assim que atribuímos o valor “adeus” na variável mensagem, ela é impressa com este valor no `console.log()`

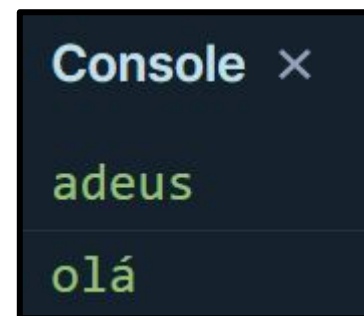
# Revisão

## Por que?

A grande diferença é que tanto o `let` quanto o `const` possuem o que chamamos de escopo de bloco.

- Uma mesma referência só existe enquanto estiver dentro de um bloco

```
1  let mensagem = "olá";  
2  {  
3      let mensagem = "adeus"  
4      console.log(mensagem);  
5  }  
6  console.log(mensagem);
```



Console ×

adeus

olá

# Operador ternário

# Revisão

O operador ternário fornece uma sintaxe abreviada para a condicional if-else da linguagem JavaScript. A sintaxe é mostrada a seguir:

```
let resultado = (condicao) ? (retorno se verdadeira) : (retorno se falsa);
```

- O valor de retorno é armazenado na variável `resultado`.
- A `condicao` é a cláusula if (falsa ou verdadeira) do if-else.
- A parte da sintaxe que declara: `? (retorno se verdadeira)` é o valor da variável `resultado` se a condição for verdadeira.
- A parte que declara: `: (retorno se falsa)` é o valor da variável `resultado` se a condição for falsa.



# Revisão

## Exemplo

```
1 ▾ if (preco < 40) {  
2     return "Livro barato!";  
3 ▾ } else {  
4     return "Livro não é barato!";  
5 }
```



```
1 let resultado = preco < 40 ? "Livro barato" : "Livro não é barato!";
```

# Revisão



[Acesse o código](#)

A cláusula `elseif` também pode ser escrita com uso do operador ternário. Nesse caso, ela vai causar um aninhamento na sintaxe. Observe a seguir um exemplo de operador ternário aninhado escrito em multilinhas.

```
1  let preco = 50;
2  preco < 40
3    ? console.log("Livro barato")
4    : preco < 70
5    ? console.log("Livro não é barato e nem caro!")
6    : console.log("Livro é caro");
```

# Arrow functions

# Revisão

A ES6 criou uma nova maneira com sintaxe muito mais amigável e intuitiva de escrever e utilizar funções JavaScript.

- Seguem alguns exemplos comparando a sintaxe segundo a ES5 e anteriores e a nova sintaxe prevista na ES6 (arrow function).

```
1  function saudacao() {  
2    |  return "Olá visitante"  
3  };  
4  console.log( saudacao() );
```

ES5

ES6

```
1  let saudacao = () => "Olá visitante";  
2  console.log( saudacao() );
```

# Revisão

Foi abolida a palavra `function` e, quando o retorno da função se faz na mesma linha de código da seta `=>`, não há necessidade de uso da palavra `return` entre sinal de chaves, tal como mostrado a seguir.

```
1  let saudacao = () => (  
2    "Olá visitante"  
3  );  
4  console.log( saudacao() );
```

# Revisão

Antes da seta estão os dados de entrada (ou parâmetros) da função e depois da seta, a saída da função (ou seu valor de retorno). Os exemplos a seguir mostram e esclarecem detalhes da sintaxe.

```
1  function dobrar(x) {  
2    |  return 2 * x;  
3  }  
4  console.log( dobrar(5) );
```

ES5

ES6

```
1  let dobrar = (x) => 2 * x;  
2  console.log( dobrar(5) );
```

# Revisão

Nas funções que admitem um ou nenhum parâmetro, o sinal de parênteses envolvendo o parâmetro é opcional, e geralmente não é usado. Assim a sintaxe alternativa (sem parênteses) para a função é mostrada a seguir.

```
1  let dobrar = x => 2 * x;  
2  console.log( dobrar(5) );
```

# Revisão

## Mais exemplos

ES5

```
1  function avaliar(x, y, z) {  
2      if( x < 10) {  
3          return y + z;  
4      } else {  
5          return y * z;  
6      }  
7  };  
8  console.log(avaliar(18, 15, 5))  
9  console.log(avaliar(2, 3, 11))
```

ES6

```
1  let avaliar = (x, y, z) => x < 10 ? y + z : y * z;  
2  console.log(avaliar(18, 15, 5));  
3  console.log(avaliar(2, 3, 11));
```



# Revisão



[Acesse o código](#)

Nesse exemplo, usou-se o operador ternário em lugar da cláusula if-else, conforme estudado anteriormente.

- Alternativamente você poderá escrever o operador ternário em linhas separadas. Nesse caso, a sintaxe multilinha para o operador é conforme mostrada a seguir.

```
1  let avaliar1 = (x, y, z) => (  
2    x < 10 // condição  
3    ? y + z // linha começa com ?  
4    : y * z // linha começa com :  
5  );
```

# Operator spread

# Revisão

O operador spread da ES6 destina-se a expandir um array, um objeto ou uma string. O símbolo para esse operador é constituído por três pontos (...).

- Considere as seguintes constantes que armazenam arrays:

```
const livros1 = ["CSS3", "JavaScript", "PHP"];  
const livros2 = ["HTML5", "React"];  
const precos = [70, 30, 90, 100, 10];
```

# Revisão

## Exemplo 1

- O resultado não é um array com os cinco livros como era de se esperar, e sim uma string como mostrada a seguir.

```
let livros = livros1 + livros2;  
console.log(livros)
```

Console ✕

CSS3, JavaScript, PHPHTML5, React

# Revisão

## Exemplo 2

- Agora sim! O uso do operador spread produz o resultado que se esperava no exemplo anterior, um array com os cinco livros como mostrado a seguir

```
let livros = [...livros1, ...livros2];  
console.log(livros)
```

### Console ×

```
▼ (5) ["CSS3", "JavaScript", "PHP", "HTML5..."]  
  0: "CSS3"  
  1: "JavaScript"  
  2: "PHP"  
  3: "HTML5"  
  4: "React"
```

# Revisão

## Exemplo 3

- Aqui passamos um array de números como parâmetro da função `Math.max()`. Isso não é permitido em ES6. O resultado é uma mensagem informando que o parâmetro `precos` não é um número, NaN.

```
let precoMaximo = Math.max(precos);  
console.log(precoMaximo)
```

Console ×

nan

# Revisão



[Acesse o código](#)

## Exemplo 4

- Agora sim! O uso do operador spread produz o resultado que se esperava, o número 100, que é o valor máximo constante do array de preços.

```
let precoMaximo = Math.max(...precos);  
console.log(precoMaximo)
```

Console ×

100

# Métodos

## map(), filter() e find()



# Revisão

## Esses três métodos são muito usados em React

- Basicamente eles manipulam objetos e se destinam a percorrer os itens de um objeto (inicial) iterável e criar um objeto (novo) com itens que resultam da manipulação dos itens do objeto (inicial).
- Esses métodos não alteram o objeto (inicial), e sim criam um objeto (novo).

# Método map()

# Revisão

Esse método admite três parâmetros: o primeiro, obrigatório, é uma função callback, o segundo, opcional, o índice do item, e o terceiro, o objeto original.

- Esse método percorre cada item de um objeto iterável (por exemplo: um array) e cria um novo objeto no qual cada item é o retorno da função callback aplicada sobre cada item do objeto original.

# Revisão

## Exemplo 1

- Aqui, usamos a função `map()` para criar dois novos arrays, `livros1` e `livros2`, que foram mapeados do array original `livros`.

```
1  const livros = ["CSS3", "HTML5", "JavaScript", "React", "PHP"];
2    let livros1 = livros.map( (livro) => "Livro " + livro );
3    let livros2 = livros.map((livro, index) => "Livro" + index + " " + livro);
4  console.log(livros);
5  console.log(livros1);
6  console.log(livros2);
```

### Console ×

```
▶ (5) ["CSS3", "HTML5", "JavaScript", "Rea...]  
▶ (5) ["Livro CSS3", "Livro HTML5", "Livro...]  
▶ (5) ["Livro0 CSS3", "Livro1 HTML5", "Liv...]
```

# Revisão



[Acesse o código](#)

## Exemplo 2

- Aqui, criamos dois novos arrays, livrosX e livrosY, que foram mapeados do array de objetos original livrosA.

```
1  const livrosA =  
2  [  
3    {titulo: "Construindo Sites com HTML", autor: "Maurício Samy Silva"},  
4    {titulo: "Web Scraping com Python", autor: "Ryan Mitchell"},  
5    {titulo: "CSS3", autor: "Maurício Samy Silva"}  
6  ];  
7  let livrosX = livrosA.map((livro) => "Livro: " + livro.titulo);  
8  let livrosY = livrosA.map((livro) => "Autor: " + livro.autor);  
9  console.log(livrosA);  
10 console.log(livrosX);  
11 console.log(livrosY);
```

### Console ×

```
▶ (3) [{...}, {...}, {...}]  
▶ (3) ["Livro: Construindo Sites com HTML"...]  
▶ (3) ["Autor: Maurício Samy Silva", "Auto..."]
```

# Método filter()

# Revisão

Esse método admite três parâmetros. O primeiro, obrigatório, é uma função callback, o segundo, opcional, o índice do item, e o terceiro, o objeto original.

- Esse método percorre cada item de um objeto iterável (exemplo: um array) e cria um novo objeto no qual cada item satisfaz uma condição de filtragem expressa na função callback aplicada sobre cada item do objeto original.

# Revisão



[Acesse o código](#)

## Exemplo 1

- Aqui, usamos a função `map()` para criar dois novos arrays, `livros1` e `livros2`, que foram mapeados do array original `livros`.

```
1  const livros =
2  [
3    {titulo: "Construindo Sites com HTML", autor: "Maurício Samy Silva"},
4    {titulo: "Web Scraping com Python", autor: "Ryan Mitchell"},
5    {titulo: "CSS3", autor: "Maurício Samy Silva"}
6  ];
7  let livros1 = livros.filter((livro) => livro.titulo === "CSS3");
8  let livros2 = livros.filter((livro) => livro.autor === "Maurício Samy Silva");
9  let livros3 = livros.filter((livro) => livro.titulo.includes("com"));
10 console.log(livros1);
11 console.log(livros2);
12 console.log(livros3);
```

Console ×

▶ (1) [{...}]

▼ (2) [{...}, {...}]

▶ 0: (2) {titulo: "Construindo Sites com HTML..."}

▶ 1: (2) {titulo: "CSS3", autor: "Maurício Sa..."}

▶ [[Prototype]]: []

▶ (2) [{...}, {...}]



# Método find()

# Revisão



[Acesse o código](#)

Esse método percorre cada item de um objeto iterável (por exemplo: um array) e cria um novo objeto no qual cada item satisfaz a condição expressa no retorno da função callback aplicada sobre cada item do objeto original.

```
1  const livros =  
2    [  
3      {id: 1, titulo: "Construindo Sites com HTML"},  
4      {id: 2, titulo: "Web Scraping com Python"},  
5      {id: 3, titulo: "CSS3"}  
6    ];  
7  let livros1 = livros.find((livro) => livro.id === 3);  
8  console.log(livros1);  
9  console.log(livros1.titulo);
```

Console ×

```
▼ (2) {id: 3, titulo: "CSS3"}  
  id: 3  
  titulo: "CSS3"  
  ► [[Prototype]]: {}
```

CSS3

# Template Literals

# Revisão

Trata-se de uma sintaxe para se declararem strings, a qual forneceu à linguagem um construtor semelhante aos existentes em outras linguagens.

- Antes da ES6, a sintaxe para se declarar uma string impunha o uso de aspas simples ou duplas envolvendo a string.
- Concatenar string impunha o uso de sinal de adição ou o método `concat()`.
- Pular linhas ou tabular strings impunha o uso de caracteres especiais escapados e assim por diante.

A nova sintaxe prevê o uso do sinal de crase ( ``` ) para envolver a string

# Revisão



[Acesse o código](#)

## Exemplo

```
1  const titulo = `Livro React do "Maujor"`;  
2  const preco = 80;  
3  const mensagem = `O preço normal do ${titulo} é de R${preco},00  
4  Na promoção o preço cai para R${preco * 0.8},00`;  
5  
6  console.log(mensagem);
```

Console ×

```
O preço normal do Livro React do "Maujor" é de R$80,00  
Na promoção o preço cai para R$64,00
```

Observe que a sintaxe para interpolar em uma string é `${ }`. Dentro do sinal de chaves pode-se inserir o nome de uma variável ou qualquer expressão válida na linguagem JavaScript.

# Atribuição via desestruturação

# Revisão

Atribuição via desestruturação é a sintaxe para uma expressão JavaScript que possibilita a extração de dados de arrays ou propriedades de objetos com uso de variáveis distintas.

## Exemplo 1

```
1  const livros = ["CSS3", "HTML5", "JavaScript", "React"];
2  let [css, html5, js, react] = livros; // aqui a desestruturação
3  console.log(css);
4  console.log(react);
```

Console ×

CSS3

React

# Revisão



[Acesse o código](#)

## Exemplo 2

```
1  const livros = [  
2    { titulo: "React", autor: "Maurício Samy Silva" },  
3    { titulo: "Node Essencial", autor: "Ricardo R. Lecheta" },  
4    { titulo: "UX Desing", autor: "Will Grant" }  
5  ];  
6  let [lUm, lDois, lTres] = livros;  
7  console.log(lUm.titulo);  
8  console.log(lTres.autor);
```

Console ×

React

Will Grant



# Import e Export

# Revisão

A diretiva `import` possibilita que você use um script dentro de outro script, ou seja, importe um script para dentro de outro script. E use esse outro script em uma página HTML. Esse mecanismo é largamente usado em React.

- No exemplo a seguir criaremos uma página HTML que usa o script `main.js`, que por sua vez importa o script `utils.js`.
- O script `utils.js` define duas funções (`estudar()` e `elogiar()`) que são declaradas exportáveis.

# Revisão

utils.js ×

```
1  const estudar = (texto) => { console.log(texto); };
2  const elogiar = (elogio) => { console.log(elogio); };
3
4  export default estudar;
5  export { elogiar };
```

- **As duas primeiras linhas** contêm funções que serão exportadas. Normalmente no final do arquivo (opcionalmente no início) declara-se a diretiva export.
- **Na linha 4**, temos a exportação por default. A exportação com uso de default, em geral, é para a função principal, ou única, existente no arquivo (módulo).
- **Na linha 5** temos a exportação com chaves. Havendo mais funções no módulo, usa-se a exportação com uso de chaves { }. Nesse caso, declara-se o nome das funções separadas por vírgula; por exemplo: {elogiar, criticar, reclamar}.

# Revisão

Em outro arquivo js, importamos as funções exportadas no arquivo anterior :)

main.js ×

```
1  import estudar from './utils.js';
2  import { elogiar } from './utils.js';
3
4  const btn1 = document.querySelector('#btn1');
5  btn1.addEventListener('click', () => {
6    |   estudar('Vamos estudar React. ');
7  });
8
9  const btn2 = document.querySelector('#btn2');
10 btn2.addEventListener('click', () => {
11   |   elogiar('O livro React do Maujor é muito bom!');
12 });
```

# Revisão

No arquivo HTML, podemos testar o comportamento esperado

index.html ×

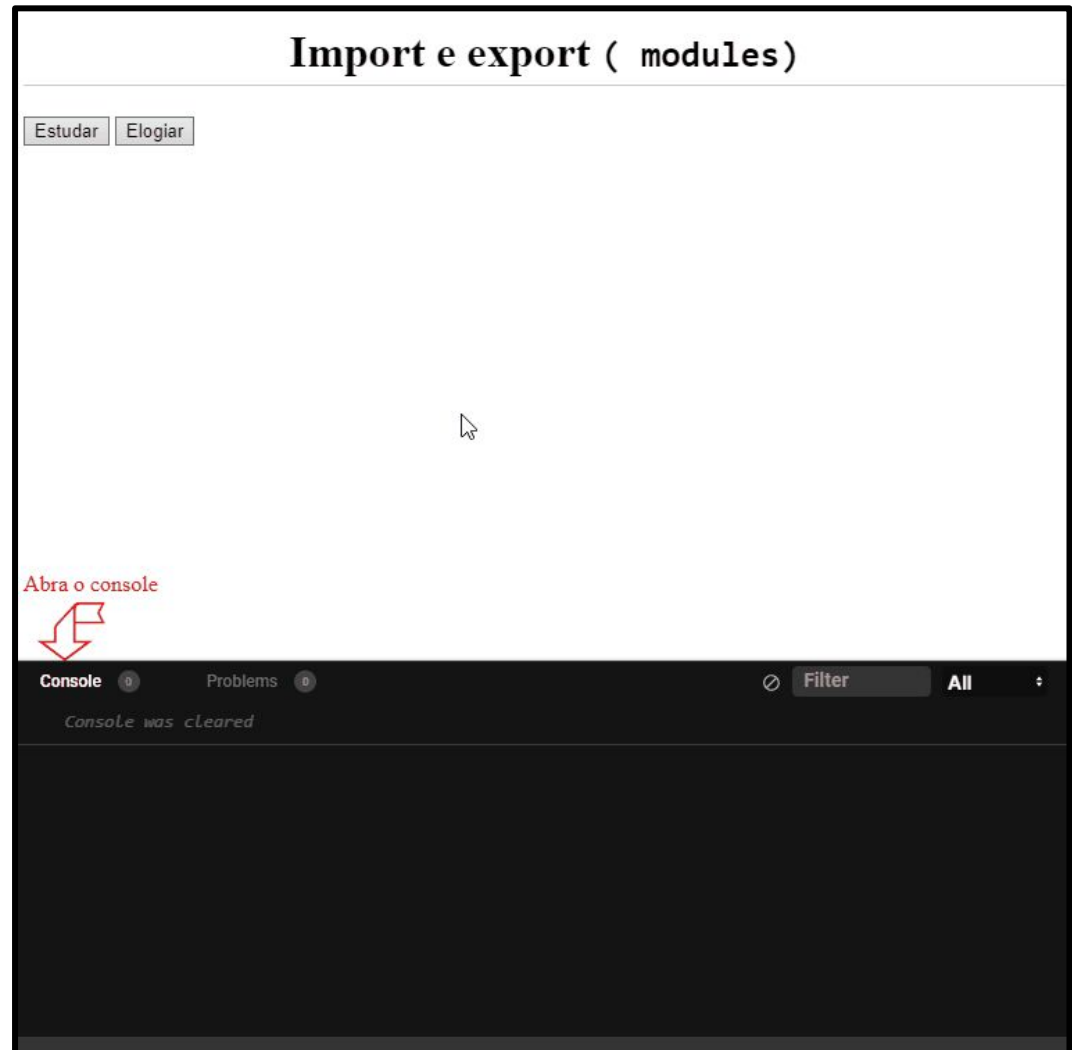
```
1  <body>
2    <h1>Exemplos</h1>
3    <p><code>( import / export )</code></p>
4    <button id="btn1">Estudar</button>
5    <button id="btn2">Elogiar</button>
6    <script type="module" src="./js/main.js"></script>
7  </body>
```

# Revisão



[Acesse o código](#)

## Exemplo completo



# Sintaxe JSX

# Revisão

JSX é a abreviatura para JavaScript XML, uma tecnologia criada com a finalidade de simplificar e facilitar a escrita de códigos JavaScript e amplamente usada em React.

- Trata-se de uma sintaxe declarativa que descreve, com uso de JavaScript, HTML e CSS, a estrutura, o comportamento e a apresentação de um componente.



# Revisão

## Exemplo de criação de marcação HTML com uso de sintaxe JavaScript puro

- Trata-se de uma sintaxe declarativa que descreve, com uso de JavaScript, HTML e CSS, a estrutura, o comportamento e a apresentação de um componente.

```
1  let elemento = document.createElement("h1");  
2  let titulo = document.createTextNode("Livro React do Maujor");  
3  const resultado = elemento.appendChild(titulo);
```

- O mesmo exemplo anterior escrito com sintaxe JSX. Nesse exemplo, uma linha de sintaxe JSX produz o mesmo efeito que três linhas de JavaScript puro.

```
1  let resultado = <h1>Livro React do Maujor</h1>;
```

# Revisão

## Outro exemplo

- Observe a sintaxe JSX para inserção de atributos.

```
1  let minhaId = "topo";  
2  const elemento = <div id = { minhaId }>TOPO</div>;
```

- JSX é uma sintaxe XML, portanto todas as tags devem ser fechadas seja com a tag de fechamento, seja com uso de barra como em <br />, <img />, <div />, <p /> ou <p></p>
- Nomes de atributos devem seguir a sintaxe camelCase como em onClick, onSubmit e colSpan

# Estruturas de seleção

if, else e else if

# Revisão

Neste exemplo mostramos como é possível atribuir um comportamento de tomada de decisão utilizando as estruturas condicionais if, else if e else no JavaScript.

```
1  let idade = 25;
2
3  if ((idade > 15 && idade < 18) || idade > 70) {
4    console.log(`O voto é opcional`);
5  } else if (idade < 16) {
6    console.log(`Você não pode votar`);
7  } else {
8    console.log(`Você é obrigado a votar`);
9  }
```

Console ×

Você é obrigado a votar

# Switch

# Revisão

O Switch Case é uma instrução que se comporta de forma semelhante ao if / else, porém possui uma estrutura mais organizada e de fácil compreensão.

- Só são aceitos valores pré-definidos e não expressões condicionais

```
1  let signo = `Leão`;
2
3  switch (signo) {
4    case `Áries`:
5      console.log(`De 21 março a 20 abril`);
6      break;
7    case `Touro`:
8      console.log(`de 21 abril a 20 maio`);
9      break;
10   case `Gêmeos`:
11     console.log(`de 21 maio a 20 junho`);
12     break;
13   case `Câncer`:
14     console.log(`de 21 junho a 22 julho`);
15     break;
16   case `Leão`:
17     console.log(`de 23 julho a 22 agosto`);
18     break;
19   case `Virgem`:
20     console.log(`de 23 agosto a 22 setembro`);
21     break;
22   default:
23     console.log(`Signo não registrado`);
24     break;
25 }
```

# Estruturas de repetição



# While

# Revisão

Podemos utilizar a estrutura de repetição `while` caso seja necessário repetir um bloco de comandos por N vezes.

```
1  let i = 0;
2  while (i < 11) {
3      console.log(`5 x ${i} = ${5 * i}`);
4      i++;
5  }
```

Console ×

5 x 0 = 0

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

# Do While

# Revisão

O do-while tem o mesmo conceito que o while com uma única diferença, a condição é verificada após os comandos do bloco serem executados,

- Ou seja, mesmo que a condição seja falsa, é garantida que o bloco será executado ao menos uma vez.

```
1  let contador = 0;
2
3  do{
4      console.log(`0 contador vale: ${contador}`);
5      contador++;
6  }while(contador == 1)
```

Console ×

0 contador vale: 0

0 contador vale: 1

For

# Revisão

A estrutura de repetição for no JavaScript segue o mesmo princípio que o while, porém este recurso é mais utilizado quando se sabe o número de iterações da repetição, como ao percorrer um vetor, por exemplo.

- Ou seja, mesmo que a condição seja falsa, é garantida que o bloco será executado ao menos uma vez.

```
1  for(let i = 0; i < 11; i++){  
2  |  console.log(`5 x ${i} = ${5*i}`);
```

Console ×

5 x 0 = 0

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

For in

# Revisão

O `for...in` é utilizado para percorrer as propriedades de um objeto, por exemplo:

```
1  let carro = {  
2    marca: `ford`,  
3    modelo: `fiesta`,  
4    cor: `prata`,  
5  };  
6  
7  for (propriedade in carro) {  
8    console.log(propriedade);  
9  }
```

Console ×

marca

modelo

cor



# Revisão

Também podemos acessar os atributos de cada propriedade, para isso basta usarmos propriedade como índice do nosso objeto, desta maneira:

```
1  let carro = {  
2    |  marca: `ford`,  
3    |  modelo: `fiesta`,  
4    |  cor: `prata`,  
5  |  };  
6  
7  for (propriedade in carro) {  
8    |  |  console.log(carro[propriedade])  
9  |  }
```

Console X

ford

fiesta

prata

# Revisão

Podemos também exibir o objeto por completo somente usando o for...in:

```
1  let carro = {  
2    marca: `ford`,  
3    modelo: `fiesta`,  
4    cor: `prata`,  
5  };  
6  
7  for (propriedade in carro) {  
8    console.log(propriedade + ': ' + carro[propriedade]);  
9  }
```

Console ×

marca: ford

modelo: fiesta

cor: prata

For of

# Revisão

O `for...of` nós podemos utilizar para percorrer objetos iteráveis como Maps, Sets e Vetores de forma simples e eficaz, da seguinte forma:

```
1  let carros = ['fiesta', 'onix', 'fusca', 'saveiro'];  
2  
3  for (let carro of carros) {  
4    | console.log(carro);  
5  }
```

Console ×

fiesta

onix

fusca

saveiro

# Revisão

Também é possível exibir o índice referente a cada item usando o método `entries()` da seguinte forma:

```
1  let carros = ['fiesta', 'onix', 'fusca', 'saveiro'];
2
3  for (let [index, carro] of carros.entries()) {
4    |  console.log(index, carro);
5  }
```

Console X

0 fiesta

1 onix

2 fusca

3 saveiro

# Classes

# Revisão

O JavaScript introduziu a palavra-chave `class` no ECMAScript 2015. Ela faz com que o JavaScript aparente ser uma linguagem de POO. O exemplo a seguir é um uso geral de uma class em JavaScript:

```
1  class Animals {
2    constructor(name, especie) {
3      this.name = name;
4      this.especie = especie;
5    }
6    sing() {
7      return `${this.name} can sing`;
8    }
9    dance() {
10     return `${this.name} can dance`;
11   }
12 }
13 let bingo = new Animals(`Bingo`, `Hairy`);
14 console.log(bingo);
```

Console ×

```
▼ Animals {name: "Bingo", especie: "Hairy"}
  name: "Bingo"
  especie: "Hairy"
  ▶ constructor: f Animals()
  ▶ sing: f sing()
  ▶ dance: f dance()
  ▶ [[Prototype]]: {}
```

# Revisão

## Outro exemplo

Console ×

Clara can sing

I have indigo whiskers

```
1  class Animals {
2      constructor(name, age) {
3          this.name = name;
4          this.age = age;
5      }
6      sing() {
7          return `${this.name} can sing`;
8      }
9      dance() {
10         return `${this.name} can dance`;
11     }
12 }
13 class Cats extends Animals {
14     constructor(name, age, whiskerColor) {
15         super(name, age);
16         this.whiskerColor = whiskerColor;
17     }
18     whiskers() {
19         return `I have ${this.whiskerColor} whiskers`;
20     }
21 }
22
23 let clara = new Cats(`Clara`, 33, `indigo`);
24 console.log(clara.sing());
25 console.log(clara.whiskers());
```



# Revisão

## Conteúdo extra sobre POO em JS

- **Classes:** <https://tinyurl.com/3nb2aphx>
- **This:** <https://tinyurl.com/ynr7p5yv>



# Exercícios resolvidos

# Exercícios

## Exercício 1 - Manipulação de Lista de Tarefas

### Visão geral

Neste exercício, deve-se criar um programa simples para gerenciar uma lista de tarefas. Com este exercício vocês vão praticar a criação de variáveis, o uso de estruturas de repetição, operadores ternários e métodos `find`, `map` e `filter`.

# Exercícios

## Instruções

1. Crie uma lista vazia chamada `listaDeTarefas`.
2. Crie um loop onde o usuário pode adicionar tarefas à lista. O loop deve continuar até que o usuário decida parar.
3. Cada tarefa deve ser um objeto com as propriedades:
  - `descricao`: uma string que descreve a tarefa.
  - `concluida`: um valor booleano que indica se a tarefa foi concluída ou não (inicialmente, definido como `false`).
4. Após o loop de adição de tarefas, imprima a lista de tarefas na tela, mostrando a descrição e o status de conclusão de cada tarefa.
5. Use o método `map` para criar uma nova lista contendo apenas as descrições das tarefas.
6. Use o método `filter` para criar uma nova lista contendo apenas as tarefas concluídas.
7. Peça ao usuário para inserir o texto de uma tarefa e use o método `find` para verificar se a tarefa existe na lista. Se existir, exiba um feedback dizendo que a tarefa foi encontrada; caso contrário, diga que a tarefa não foi encontrada.

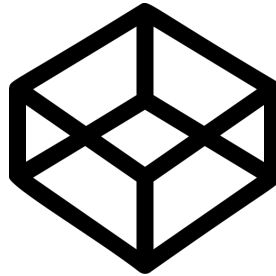
# Exercícios

## Dicas

- Use a função ``prompt`` para receber entradas do usuário.
- Utilize estruturas de repetição como ``while`` ou ``do-while`` para criar o loop de adição de tarefas.
- Ao usar o método ``map`` e ``filter``, lembre-se de que eles retornam novas listas, então você precisará armazenar esses resultados em variáveis separadas.

# Exercícios

Resposta



<https://codepen.io/joaopaulofcc/pen/JjegMoo>

# Exercícios

## Exercício 2 - Sistema de Gerenciamento de Biblioteca

### Visão geral

Deve-se criar um sistema de gerenciamento de uma biblioteca usando conceitos de orientação a objetos em JavaScript. Com este exercício vocês vão praticar a criação de classes, construtores, métodos e herança.

# Exercícios

## Instruções

1. Crie uma classe `Livro` com as propriedades:
  - `titulo`: título do livro (string).
  - `autor`: autor do livro (string).
  - `anoPublicacao`: ano de publicação do livro (number).
  - `disponivel`: indica se o livro está disponível para empréstimo (boolean, inicialmente `true`).
2. Crie uma classe `Usuario` com as propriedades:
  - `nome`: nome do usuário (string).
  - `email`: endereço de e-mail do usuário (string).
  - `livrosEmprestados`: uma lista de livros que o usuário pegou emprestado (array, inicialmente vazio).
3. Crie métodos na classe `Usuario`:
  - `emprestarLivro(livro)`: recebe um objeto `Livro` como argumento e adiciona o livro à lista `livrosEmprestados` do usuário. Se o livro estiver disponível, atualize sua propriedade `disponivel` para `false`.
  - `devolverLivro(livro)`: recebe um objeto `Livro` como argumento e remove o livro da lista `livrosEmprestados` do usuário. Atualize a propriedade `disponivel` do livro para `true`.
4. Crie uma classe `Biblioteca` que herda propriedades e métodos de `Livro`:
  - Adicione uma propriedade `catalogo` que é uma lista de objetos `Livro`.
  - Crie um método `buscarLivro(titulo)` que recebe o título de um livro e retorna o objeto `Livro` correspondente do catálogo, ou uma mensagem caso o livro não seja encontrado.



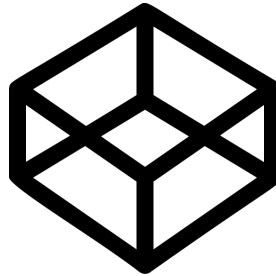
# Exercícios

## Dicas

- Use a palavra-chave `class` para definir suas classes.
- Utilize o construtor (`constructor`) para inicializar as propriedades das classes.
- Lembre-se de que o conceito de herança é implementado com a palavra-chave `extends`.

# Exercícios

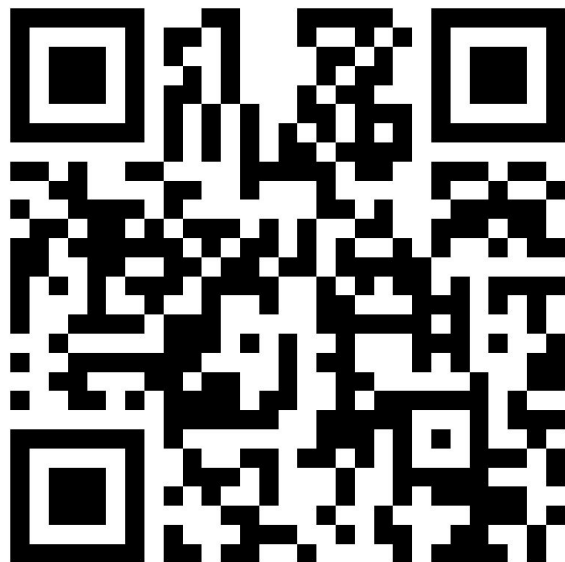
Resposta



<https://codepen.io/joaopaulofcc/pen/YzRbYqp>

Avalie a aula!

Avalie a aula de hoje :)



# Referências

# Referências



SILVA, Maurício Samy. **React - Aprenda praticando.** Novatec. 2021.

PINHO, Diego. **Let, const e var não é tudo a mesma coisa?** 2018. Disponível em: <https://tinyurl.com/2c66d7kh>. Acesso em: 7 ago. 2024.



# Referências



GADO, Wesley. Estruturas condicionais e estruturas de repetição em JavaScript. 2021. Disponível em: <https://tinyurl.com/2kc8v8e5>. Acesso em: 7 ago. 2024.

# **Análise e Desenvolvimento de Sistemas**

## **Frameworks Web I**

### **Aula 01 - Revisão de JavaScript**

---

---