

# Boas-vindas!

Esteja confortável, pegue uma água e se acomode em um local tranquilo que já começamos.

Como você **chega?**

1



2



3



Esta aula será

- gravada

# Resumo

## da última aula

- ✓ Relação e diferença entre FrontEnd e BackEnd
- ✓ Fazendo as pazes: Stack MERN
- ✓ Como testar Javascript
- ✓ Tipo de dados de Javascript
- ✓ Variáveis de Javascript

Perguntas?

Aula 02. BACKEND

# Princípios básicos de Javascript e novos recursos ECMAScript

# Objetivos da aula



Familiarizar-se com estruturas e conceitos fundamentais ao programar usando JavaScript



Conhecer os novos elementos de linguagem fornecidos pelo ES6



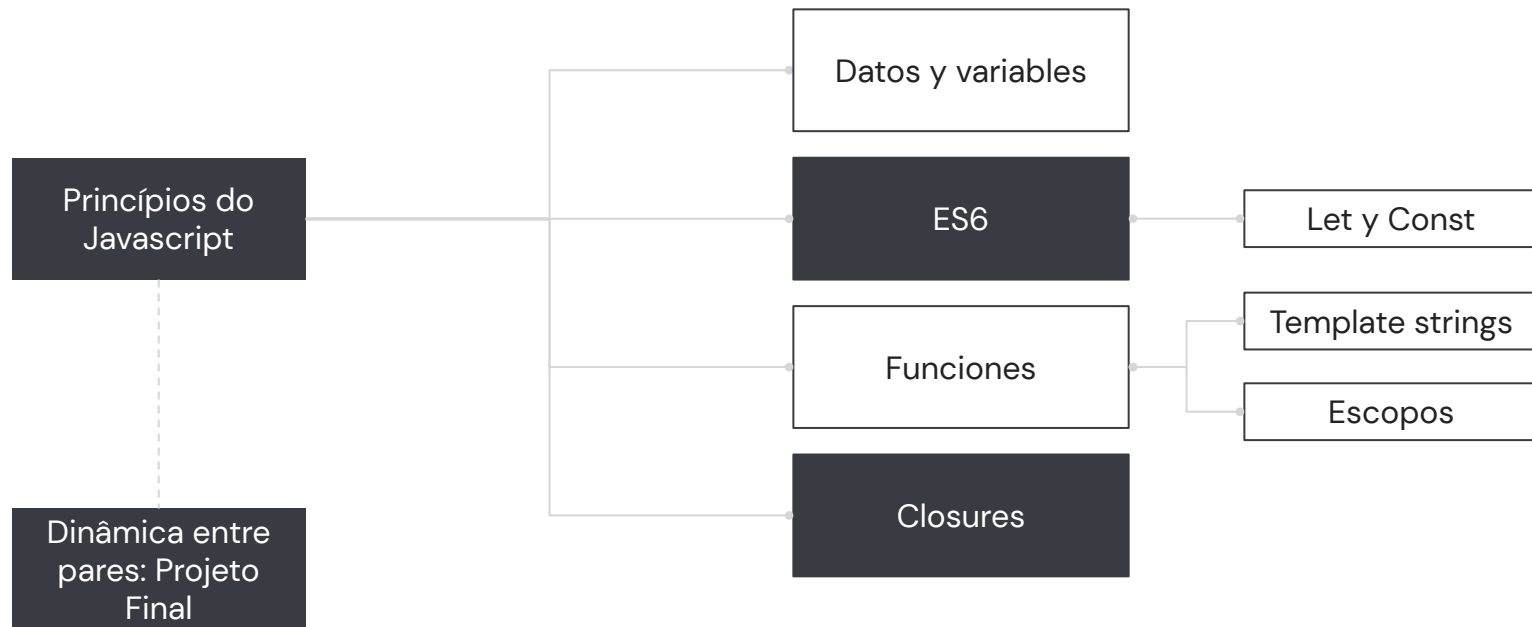
Conhecer as características do ECMAScript



Aplice os conceitos incorporados no desenvolvimento do backend

# Princípios básicos de JavaScript

## MAPA DE CONCEITOS – Javascript







## MAPA DE CONCEITOS - ECMAScript





# Revendo conceitos da última aula



ATIVIDADE EM SALA

# Revisando conceitos da última aula

Na aula 1, falamos dos tipos de dados com os quais podemos trabalhar no JavaScript.

Vamos colocar esse conhecimento em jogo!

Vejamos um bloco de código e tente identificar os tipos de dados presentes nele.  
**Escreva no chat o tipo de dados que vocês identificam.**

## RELEMBRANDO 🤖

		Nome	Descrição
Tipos de dados em Javascript	Tipos primitivos que características compartilham?	String	Cadeias de texto
		Number	Valores numéricos
		Booleans	True/false
		Null	Tipo especial, contém null
		Undefined	Tipo especial, contém undefined.

## RELEMBRANDO 🤖

		Nome	Descrição
Tipos de dados em Javascript	Tipos objeto que características compartilham?	Tipos predefinidos de Javascript	Date (data) RegExp (expressões regulares) Error (dados de erros)
		Tipos definidos pelo programador e usuário	Funções Simples Classes
		Arrays	Série de elementos que formam uma matriz ou vetor. Vamos considerá-lo um objeto especial que necessita de métodos.
		Objetos especiais	Objeto global
			Objeto protótipo
			Outros

JS ExercicoTiposDeDados.js > ...

```
1  let people = [  
2    {id:1, first_name: 'Luiz', last_name: 'Pereira', age:25, gender:'M'},  
3    {id:2, first_name: 'Maria', last_name: 'Silva', age:30, gender:'F'},  
4    {id:3, first_name: 'José', last_name: 'Silveira', age:42, gender:'M'},  
5    {id:4, first_name: 'Luiza', last_name: 'Camões', age:39, gender:'F'}  
6  ];  
7  
8  let person = people.find(p=>{  
9    let test;  
10   test = p.id === 3;  
11   return test  
12 });  
13  
14 console.log(person);
```



MATERIAL EXTRA

# Documentação de Javascript

- ✓ [w3schools](https://www.w3schools.com/js/)
- ✓ [javascript.info](https://javascript.info/)



Para pensar

Uma variável é um espaço de memória alocado pelo computador, para poder salvar dados.

É verdadeiro ou falso?

**Compartilhe no chat suas opiniões**





Para pensar

Uma variável não pode alterar seu valor, embora o programa precise. Portanto, não pode ser reutilizada.

É verdadeiro ou falso?

**Compartilhe no chat suas opiniões**

# O que é ECMAScript?

É um padrão responsável por **organizar como a linguagem JavaScript deve ser interpretada**. Com o avanço da linguagem e o crescimento dos navegadores, era necessário estabelecer o controle.

O **ECMAScript** levanta compatibilidades, novas funções, funções de desuso e outras em cada um de seus novos lançamentos.

Cada novo release (versão) do ECMAScript que é lançado é indicado pelo ES+Número da versão:

**ES + Número da versão**

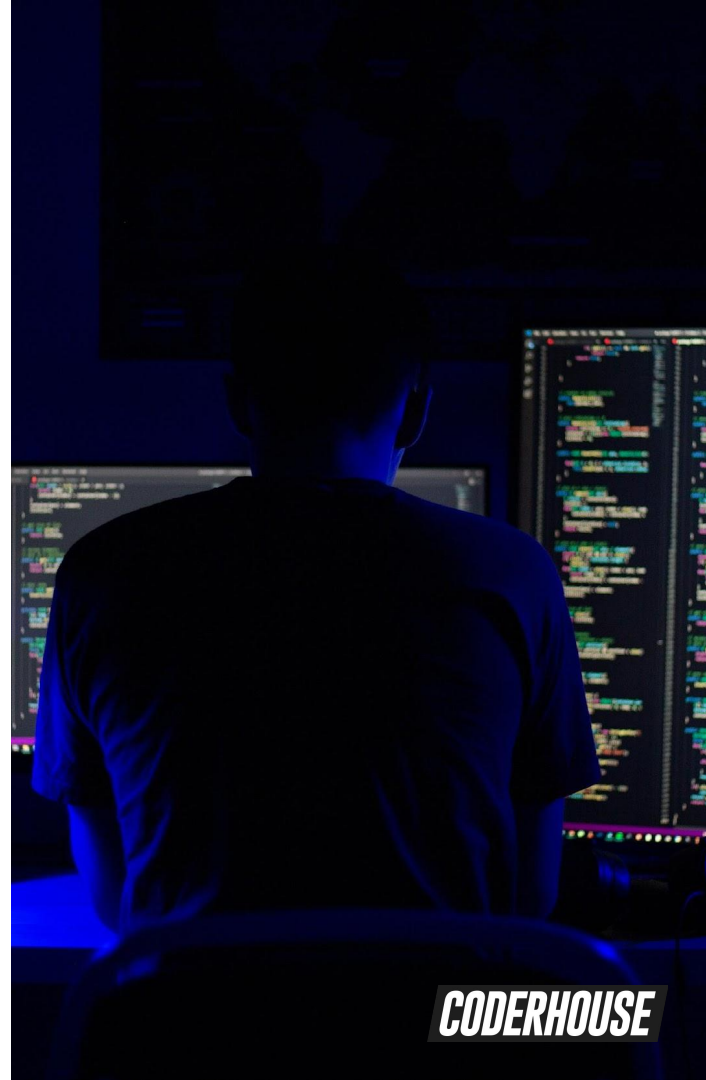
*Ex.: ES6, ES7, ..., ES11*

# Mudanças importantes do ES6

# let e const

**let e const** são duas maneiras de declarar variáveis no JavaScript introduzidas no ES6 que limitam o escopo da variável ao bloco em que foi declarado (*antes disso, não era assim*).

Você pode encontrar exemplos e codificar na Internet usando a palavra "var" para criar variáveis. Assim que era feito antes do ES6 e **seu uso não é recomendado!**



# let

Um bloco no JavaScript pode ser entendido como "o que existe entre duas chaves", sejam eles definições de funções ou blocos if, while, for e para loops semelhantes.

## Documentação let

Se uma variável for declarada com **let no escopo global ou no escopo de uma função**, a **variável pertence ao escopo global ou ao escopo da função, respectivamente**.

# Exemplo let

Aqui a variável `i` é **global** e a variável `j` é **local**.

Mas se declararmos uma variável com `let` dentro do bloco, que por sua vez está dentro de uma função ou `if`, a variável pertence apenas a esse bloco.

```
let i = 0;
function foo() {
  i = 1;
  let j = 2;
  if (true) {
    console.log(i); // 1
    console.log(j); // 2
  }
}
foo();
```

```
function foo() {
  let i = 0;
  if (true) {
    // Será outra variável i
    // apenas no bloco if
    let i = 1;
    console.log(i); // 1
  }
  console.log(i); // 0
}
foo();
```

# Exemplo let

```
function foo() {  
  if(true) {  
    let i = 1;  
  }  
  // ReferenceError:  
  // i is not defined  
  console.log(i);  
}  
foo();
```

Fora do bloco onde se declara let, a **variável não está definida**.



# const

Semelhante a declaração let, o escopo (scope) para uma variável declarada com const é o bloco.

## documentação de const

No entanto, const também **proíbe a reatribuição de valores** (const vem de constant).

# Exemplo const

```
const i = 0;
```

```
// TypeError:
```

```
// Assignment to constant variable
```

```
i = 1;
```

Se tentar reatribuir o valor de uma constante, **será retornado um erro.**

# Mutabilidade e const

Embora com `let` uma variável possa reatribuir um valor e com `const` não seja possível, pois se você tentar reatribuir um valor a uma constante, será obtido um erro `TypeError`.

Porém, não poder reatribuir valores não significa que sejam imutáveis.

Se o valor de uma constante for algo "**mutável**", como uma matriz ou um objeto, **você poderá alterar os valores internos de seus elementos**.

## Não reatribuição $\neq$ Imutável




EXEMPLO AO VIVO

É possível declarar uma constante do tipo string e alterar o valor desta constante para outra string?

É possível declarar uma constante do tipo `Array<Number>` e alterar o último e primeiro elemento? Por quê?

**Compartilhe no chat suas opiniões**

# Exemplo ao vivo

 **Instruções para equipe de aula:** professor realiza o exemplo ao vivo e traz a discussão para o chat com a participação dos estudantes na resolução das perguntas direcionadoras.

# Exemplo de mutabilidade

A imutabilidade oferece controle mais rígido sobre seus dados imediatamente, tornando seu código mais seguro e previsível.

```
const user = 'Juan';  
//TypeError: Assignment to constant  
user = 'Manolo';
```

```
const user = { name: 'Juan' };  
user.name = 'Manolo';  
console.log(user.name); // Manolo
```

# Tipos de dados em JavaScript

# O que é uma função?

São **blocos de instrução** que **funcionam em um escopo interno** (conhecido como escopo local). Eles podem ser encontrados em sua sintaxe básica ou em sua notação de flecha (arrow function).

```
<body>
  <script>
    var anosEstudo = 1;

    function experiencia(anos) {
      if (anos <= 1) {
        return 'Iniciante';
      } else if (anos <= 3) {
        return 'Intermediário';
      } else if (anos <= 6) {
        return 'Avançado';
      } else {
        return 'Jedi Master';
      }
    }

    console.log(experiencia(anosEstudo));
  </script>
</body>
```





## EXEMPLO AO VIVO

Execução de uma função com estrutura básica. Os elementos e um caso de uso serão destacados.

Execução de uma a mesma função com a estrutura da flecha (arrow function). As diferenças serão destacadas.

# Exemplo de diferentes tipos de função

```
JS 1.js > ...
1 function nomeDaFuncao(parametros){
2   /*Corpo da função, todas as instruções
3    * internas necessárias para que a função
4    * execute
5    */
6   let variavelParaMinhaFuncao=2;
7   return variavelParaMinhaFuncao;
8   /* Com a palavra "return" podemos enviar o
9    * valor da variável para outro escopo
10    * que necessite deste valor
11    */
12 }
13
14 /*EXEMPLO COMPLETO*/
15 function somarDoisNumeros(numero1,numero2){
16   ---// resultado só existe dentro da função
17   ---let resultado;
18   ---resultado = numero1 + numero2;
19   ---/* Após a execução da função é necessário
20    * que retorne o valor da soma dos números
21    * para outro escopo
22    */
23   ---return resultado;
24 }
25
26 let total = somarDoisNumeros(2,3);
27 console.log(total);
```

Função "normal"

```
JS 2.js > ...
1 /* Uma função flecha(arrow function) é anônima, quer dizer que
2  * que não possui nome, porém é possível atribuí-la a uma variável
3  * para que possa identifica-la
4  */
5 const indentificadorDeFuncao = (parametros) => {
6   /*Corpo da função, todas as instruções
7    * internas necessárias para que a função
8    * execute
9    */
10   let variavelParaMinhaFuncao=2;
11   return variavelParaMinhaFuncao;
12   /* Com a palavra "return" podemos enviar o
13    * valor da variável para outro escopo
14    * que necessite deste valor
15    * A função flecha(arrow function), possui retorno implícito
16    */
17 }
18
19 /* A função flecha(arrow function), possui retorno implícito */
20 const indentificadorDeFuncaoRetornoImplicito = (parametros) => parametros;
21
22 /*EXEMPLO COMPLETO*/
23 const somarDoisNumeros = (numero1,numero2) => {
24
25   let resultado;
26   resultado = numero1 + numero2;
27   return resultado;
28 }
29
30 const somarDoisNumerosImplicito = (numero1,numero2) => numero1+numero2;
```

Arrow function

# Importante revisar

A função de seta permite um retorno implícito, o que **permite usar instruções sem chaves**. Isso só é possível se a função tiver uma instrução. Você verá muitos deles em um ambiente de trabalho.

Se a função de seta tiver apenas um argumento, não será necessário envolver o parâmetro em um parêntese. Isso **só é necessário ao usar dois argumentos ou mais**.

Documentação auxiliar(function)

Arrow Function

Escopos

O **escopo** define o **alcance de uma variável ou constante para um determinado contexto**. Isso permite que o mesmo nome seja usado para diferentes variáveis, sem confundir a compilação.

O **escopo global** **afetará todo o nível do arquivo no qual trabalhamos**, enquanto o **escopo local** **afetará a função ou bloco em que é declarado**.

# Exemplo de escopo inválido

Se a variável for **definida exclusivamente dentro da função**, ela **não será acessível** de fora da função ou de outras funções.

```
function exemploFuncao() {  
  // x só pode ser acessada dentro da função exemploFuncao  
  const x = 'declarada em escopo local';  
  console.log(x);  
}  
  
console.log(x); // ReferenceError: x is not defined
```

# Exemplo de escopo válido

O código a seguir é **válido** porque a variável é declarada fora da função, tornando-a global

[Documentação complementar de escopos](#)

[Documentação ES6 escopos](#)

```
const x = 'declarada no escopo global'

function example() {
  console.log(x) // x existe na função
}

example() // aqui não ocorrerá erro

console.log(x) // x existe aqui fora também
```



# Break

5 minutos e voltamos!







# Break

10 minutos e voltamos!



# Novos recursos ECMAScript

# Processo de alteração por ECMAScript

1

São definidas as novas funções que estarão na linguagem e eliminam as obsoletas

2

Estas alterações vão para uma fase de testes para corroborar as funcionalidades

3

A nova versão é lançada, os navegadores devem se ajustar a esta nova versão.

# Ponto de partida ES6

# ECMAScript



Desde o ES5 em 2009, houve um longo período de tempo em que não houve grandes atualizações nos padrões de linguagem. Até o ECMAScript 6 em 2015, que foi uma revolução na linguagem devido às suas grandes mudanças.

O lançamento do ES6 marcou um antes e um depois na história da linguagem, pois a partir de então passou a ser considerada uma implementação "moderna".

# JavaScript e ECMAScript 7

# O que há de novo?

As principais características desta versão são:

- ✓ O operador exponencial `**` é introduzido, gradualmente tornando-se independente da biblioteca Math.
- ✓ Manipulação de array com includes. Que nos permite saber se existe algum elemento dentro do array.



## EXEMPLO AO VIVO

Uso do operador exponencial e manipulação de array com includes.

```
JS exponencial_includes.js > ...
1 // Exponencial ** permite fazer o equivalente da operação Math.pow(base,exp), para elevar um valor base a um expoente dado.
2 let valoresBase = [1,2,3,4,5,6];
3 let novosValores = valoresBase.map((numero, indice) => numero**indice);
4 console.log(novosValores); // resultado: [ 1, 2, 9, 64, 625, 7776 ]
5 /**
6  * O Código mostrado acima pega um array com valores base e, com utilização do método map, utiliza o operador exponencial para elevar
7  * o valor base, pelo seu índice: (1**0, 2**1, 3**2, 4**3, 5**4, 6**5)
8  */
9
10 //Includes: Verifica se existe algum elemento dentro do array
11 let nomes = ['Maria', 'João', 'José', 'Ancelmo'];
12 /** includes retornará apenas true ou false, retornará true quando existir algum elemento do parâmetro passado no includes
13  * e retornará false quando não existir
14  */
15 if (nomes.includes('Ancelmo')) {
16   console.log('Ancelmo existe dentro do array');
17 } else {
18   console.log('Nome não encontrado na base de dados');
19 }
```



# JavaScript e ES8

# O que há de novo?

As principais características desta versão são:

- ✓ Async e await por um melhor controle assíncrono, vamos nos aprofundar mais nisso em aulas futuras.
- ✓ `Object.entries`, `Object.values`, `Object.keys` para melhor controle interno sobre as propriedades de um objeto.





## EXEMPLO AO VIVO

Utilização de Object.entries, Object.values, Object.keys

```
1 let impostos = {
2   imposto1: 2341,
3   imposto2: 341,
4   imposto3: 4611,
5   imposto4: 111
6 };
7
8 let partChaveValor = Object.entries(impostos);
9 console.log(partChaveValor); //resultado: [ [ 'imposto1', 2341 ], [ 'imposto2', 341 ], [ 'imposto3', 4611 ], [ 'imposto4', 111 ]
10
11 /**
12  * Aqui vemos que Object.entries retorna um array de arrays de chave e valor e em caso de necessidade
13  * podemos utilizar separado
14  */
15
16 let apenasPropriedades = Object.keys(impostos);
17 console.log(apenasPropriedades); // resultado: [ 'imposto1', 'imposto2', 'imposto3', 'imposto4' ]
18 /**
19  * Aqui vemos que Object.keys retorna um array com as propriedades do objeto, é um método muito útil
20  * em códigos profissionais, porém, por questões de complexidade este assunto será abordado no futuro.
21  */
22
23 let apenasValores = Object.values(impostos);
24 console.log(apenasValores); // resultado: [ 2341, 341, 4611, 111 ]
25
26 /**
27  * obtendo apenas os valores do objeto, podemos utiliza-los para obter um total(Neste exemplo utilizaremos
28  * um método já existente reduce)
29  */
30 let impostosTotais = apenasValores.reduce((valorInicial, valorAcumulado)=>valorAcumulado+valorInicial);
31 console.log(impostosTotais); // 7404, total de todos os impostos
```

# JavaScript e ES9

# O que há de novo?

As principais características desta versão são:

- ✓ Resolvedores de promessa com `.finally()`, para atender uma promessa, seja ela cumprida ou não.
- ✓ Maior controle para objetos com operadores rest e spread (também aplicável a arrays)



## EXEMPLO AO VIVO

Utilização básica de operador rest e operador spread nos objetos

```
JS spread_restjs > ...
1 // Dados os seguintes objetos
2 let objeto1 = {
3   propriedade1: 2,
4   propriedade2: "b",
5   propriedade3: true
6 };
7 let objeto2 = {
8   propriedade1: "c",
9   propriedade2: [2,3,5,6,7]
10 };
11
12 //SPREAD OPERATOR serve para fazer uma desestruturação do objeto, para usarmos apenas as propriedades que queremos
13 let {propriedade1, propriedade2} = objeto1; // pegamos o objeto1 e "quebramos" apenas as 2 primeiras propriedades
14 let objeto3 = {...objeto1,...objeto2} // podemos utilizar o spread operator para pegar propriedades de outros objetos e criar um novo
15
16 console.log(objeto3); // resultado { propriedade1: 'c', propriedade2: [ 2, 3, 5, 6, 7 ], propriedade3: true }
17 /**
18  * Notamos que no objeto3, como é uma junção de 2 objetos(objeto1 e objeto2) e ambos possuem propriedades com o mesmo nome
19  * no objeto3 as informações da propriedade1 e propriedade2, foram sobrepostas pelos valores do objeto2 por estar por último
20  */
21
22 let objeto4 = {
23   a: 1,
24   b: 2,
25   c: 3
26 };
27
28 let {a,...rest} = objeto4; // indicamos que queremos trabalhar com a propriedade "a" e guardar em um objeto as outras propriedades
29 //objeto, caso precisemos no futuro.
30 console.log(rest); // resultado: { b: 2, c: 3 }
```



# Para praticar

Duração: 15 minutos



ATIVIDADE EM SALA

# Para praticar

Utilização ES6-ES9

**Descrição da atividade:** Dados os objetos indicados no próximo slide, faça uma nova lista (array) contendo todos os tipos de produtos (não quantidades), dica: use `Object.keys` e `Array.includes`. Mostrar array por console. Posteriormente, obtenha o total de produtos vendidos para todos os objetos (use `Object.values`)

**Duração:** 15 minutos



```
const objetos = [  
  {  
    macas:3,  
    peras:2,  
    carne:1,  
    frango:5,  
    doces:2  
  },  
  {  
    macas:1,  
    cafes:1,  
    ovos:6,  
    frango:1,  
    paes:4  
  }  
]
```

# Novas implementações com ES10 e ES11

# O que consideramos aqui?

A partir dessas implementações, temos que fazer um esclarecimento importante. Muitas das mudanças nessas versões estão intimamente ligadas a outros conceitos que você aprenderá ao longo do curso. Portanto, a ênfase será colocada apenas nos elementos com os quais podemos trabalhar neste momento do curso.

# Javascript e ES10

## O que podemos destacar?

As principais características desta versão são:

- ✓ **String.trim():** Remove espaços desnecessários em uma string. Ele é usado para validar strings enviadas vazias ou remover espaços iniciais e finais.
- ✓ **Array.flat():** Remove o aninhamento interno em arrays para deixar um array plano.
- ✓ **Dynamic import:** permite importar apenas os módulos necessários, economizando espaço e memória.



# Dynamic import

Um dos principais problemas das importações tradicionais é que acabamos importando **TODOS OS MÓDULOS**, mesmo quando não estamos utilizando todos ao mesmo tempo.

**Com a importação dinâmica, isso muda.**

A importação dinâmica permite importar apenas os módulos que preciso de acordo com uma determinada situação, o que permite otimizar o uso de recursos, pedindo ao computador apenas o que vou usar.

É usado principalmente em código que usa o padrão de design Factory *(falaremos mais sobre isso mais tarde)*.

JS calculadora.js X

JS calculadora.js > calculadora

```
1 //Esta calculadora só é utilizada
2 //se o programa tiver no modo "cálculos"
3
4 export default class calculadora {
5   somar = (num1, num2) => num1+num2;
6   subtrair = (num1, num2) => num1-num2;
7 };
```

JS index.js X

JS index.js > exemploImport > Calculadora

```
1 let modo = 'cálculos';
2
3 async function exemploImport() {
4   if (modo === 'cálculos') {
5     const { default: Calculadora } = await import('./calculadora.js');
6     let calculadora2 = new Calculadora();
7     console.log(calculadora2.somar(1,2)); // 3
8   }
9 }
10 exemploImport();
```



## EXEMPLO AO VIVO

Validação de string com trim.

Achatamento de array com aninhamento múltiplo

```
JS trim_flat.js > ...
1 let cadeia1 = '                olá'; // a mensagem não deveria enviar desta forma, consome espaço desnecessário
2 console.log(cadeia1.trim()); //olá
3 // Podemos simular o envio de uma mensagem vazia(simulando um chat)
4 let mensagens = [];
5 let intencaoDaMensagem = '                ';
6 if (intencaoDaMensagem.trim().length>0) { // pelo menos precisa existir 1 caracter para enviar a mensagem
7     mensagens.push(intencaoDaMensagem.trim());
8 } else { // se entrar na condição else, é por que a mensagem está vazia
9     console.log('Mensagem vazia, para enviar uma mensagem, por favor escreva algo');
10 }
11 /**
12  * Sem o método trim permitimos muitas brechas de segurança ao processar cadeias de textos,
13  * de modo que podemos limitar um formato que dominamos ( sem espaços extras em cadeias vazias)
14  */
15
16 //uso do flat
17 let arrayAninhado = [1,32,4,5,6,[1,4,5,1],[3411,3,4]]; // Anteriormente utilizavamos uma técnica chamada recursividade, para conseguirmos acessar
18 //os valores internos de cada array, era muito mais sensível;
19 console.log(arrayAninhado.flat()); // resultado: [1, 32, 4, 5, 6, 1, 4, 5, 1, 3411, 3, 4 ], notamos que não existe mais aninhção
20 // e podemos manipula-los melhor.
```

# Javascript e ES11

## O que podemos destacar?

As principais características desta versão são:

- ✓ **Operador nullish:** É usado para poder colocar um valor padrão para variáveis que podem ser nulas ou indefinidas, ao contrário do operador ||, essas filtram "falsas"
- ✓ **Variáveis privadas dentro das classes,** isso permite que algumas variáveis não sejam acessíveis a partir do ambiente de instância de uma classe e sejam usadas apenas internamente.





## EXEMPLO AO VIVO

Explicação da atribuição de variável de um nulo, para entender sua diferença com o operador OR ||  
Explicação de uma variável privada em uma classe.

```
JS nullish_privateVariables.js X
JS nullish_privateVariables.js >  Persona
1  /** O operador nullish(??) difere do operador ||, pois o operador || também verifica se a variável é "false"
2    * e o operador nullish verifica apenas se a variável é indefinida ou nula.
3    */
4
5  let variavelProva = 0; // Substituir esta variável por diferentes valores undefined, null e falso(0)
6  let variavelPreenchida = variavelProva || 'Sem valor';
7  console.log(variavelPreenchida); // caso precise do valor zero, porém utilizando o operador ||, será retornado o valor default
8  let variavelNullish = variavelProva ?? 'Sem valor';
9  console.log(variavelNullish); // aqui vemos que retronou o valor zero(false), pois utilizou o operador nullish
10
11 /* Variável privada */
12 class Persona {
13   #fullname; // declaramos a varável antes do construtor, para utilizarmos no construtor da classe
14
15   constructor(nome, sobrenome) {
16     this.nome=nome;
17     this.sobrenome=sobrenome;
18     this.#fullname = `${this.nome} ${this.sobrenome}`; // colocamos valor na variável privada
19   }
20
21   getFullname = () => this.#fullname; // Única forma de acessar este valor desta variável privada é através do método getFullname
22   //Permite o uso de variáveis, mas evitando que sejam modificadas de fora(por segurança)
23
24   #metodoPrivado = () => null; // também podemos declarar métodos privados, que são apenas utilizados dentro da classe
25 }
26
27 let instancia1 = new Persona('José', 'da Silva');
28
29 console.log(instancia1.getFullname());
```



# Hands on Lab

Duração: 20 minutos



ATIVIDADE EM SALA

# Hands on Lab

Nesta instância da aula iremos revisar alguns dos conceitos vistos em aula com um aplicativo

## De que maneira?

O professor demonstrará como fazer e você poderá replicar em seu computador. Se surgirem dúvidas, você pode compartilhá-las para resolvê-las junto com a ajuda dos tutores.

**Duração:** 20 minutos



# Registrador de Ingressos de Eventos

Como fazemos? Será criada uma classe que permitirá um gerenciamento completo dos usuários que desejam acessar esses eventos.

- ✓ Defina a classe TicketManager, que terá um array de eventos que começará vazio. A classe deve ter uma variável privada "basePrice", que adicionará um custo adicional ao preço de cada evento.
- ✓ Deve possuir o método "getEvents" que mostrará os eventos salvos.



## ATIVIDADE EM SALA

# Registrador de Ingressos de Eventos

- ✓ Deve possuir o método "addEvent", que receberá os seguintes parâmetros:
  - nome
  - lugar
  - preço (0,15 deve ser adicionado ao valor original)
  - capacidade (50 por padrão)
  - data (hoje por padrão)

O método também deve criar o campo id de auto-incremento e o campo "participants" que sempre iniciará com um array vazio.



ATIVIDADE EM SALA

# Registrador de Ingressos de Eventos

- ✓ Deve ter um método "addUser" que receberá:
  - ID do evento (deve existir, adicione validações)
  - ID do usuário

O método deve avaliar que o evento existe e que o usuário não foi previamente cadastrado (validação de data e lotação será evitada para não alongar o desafio)

Se tudo estiver em ordem, você deve adicionar o id do usuário no array

"participants" daquele evento.



## ATIVIDADE EM SALA

# Registrador de Ingressos de Eventos

- ✓ Deve possuir um método "putEventoEnGira" que receberá:
  - ID do evento
  - nova cidade
  - Nova data

O método deve copiar o evento existente, com um novo local, nova data, novo id e seus participantes vazios (Use o operador spread para o restante das propriedades)

Perguntas?



# Como foi a aula?

1

**Que bom**

O que foi super legal na aula e podemos sempre trazer para as próximas?

2

**Que pena**

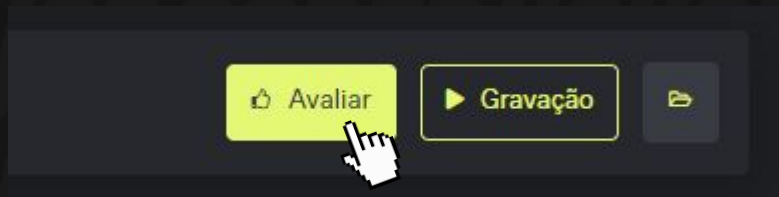
O que você acha que não funcionou bem e precisamos melhorar?

3

**Que tal**

Qual sugestão deveríamos tentar em próximas aulas?

# O que você achou da aula?



Seu feedback vale pontos para o Top 10!! 😎



## Deixe sua opinião!

1. Acesse a plataforma
2. Vá na aula do dia
3. Clique em **Avaliar**



# Classes com ECMAScript e ECMAScript avançado

Desafio obrigatório



DESAFIO OBRIGATÓRIO

# Classes com ECMAScript e ECMAScript avançado

## Objetivo

- ✓ Crie uma classe "ProductManager que gerencia um conjunto de produtos"

## Aspectos a incluir

- ✓ Deve ser criado a partir de seu construtor com o elemento products, que será um array vazio.

Cada produto que você gerencia deve ter as seguintes propriedades:

- ✓ title (nome do produto)
- ✓ description (descrição do produto)
- ✓ price (preço)
- ✓ thumbnail (caminho da imagem)
- ✓ code (código identificador)
- ✓ stock (número de peças disponíveis)



DESAFIO OBRIGATÓRIO

# Classes com ECMAScript e ECMAScript avançado

## Aspectos a incluir

- ✓ Deve ter um método "addProduct" que adicionará um produto ao array inicial de produtos.
- ✓ Valide que o campo "código" não se repete e que todos os campos são obrigatórios
- ✓ Ao adicioná-lo, ele deve ser criado com um id de incremento automático
- ✓ Deve ter um método "getProductById" que deve procurar na matriz o produto que corresponde ao id
- ✓ Caso nenhum id corresponda, exiba um erro "Não encontrado" no console

## Formato de entrega:

- ✓ Arquivo Javascript pronto para ser executado a partir do node.

# Resumo

## da aula de hoje

- ✓ Familiarizar-se com estruturas e conceitos fundamentais ao programar usando JavaScript
- ✓ Conhecer as características do ECMAScript
- ✓ Aplique os conceitos incorporados no desenvolvimento do backend



**Obrigado por estudar  
conosco!**