2. Por que aprender JavaScript?

Basicamente, são três os principais motivos para aprender JavaScript:

- É a única linguagem 100% fullstack;
- É a única linguagem que roda nativamente em todos os browsers;
- É fácil de aprender.

Vejamos:

Única linguagem 100% fullstack

Fullstack significa dominar todas as tecnologias que a aplicação utiliza.

Na maioria dos casos, um negócio precisa de mais de uma aplicação pra existir, como vemos na **Figura 1**.

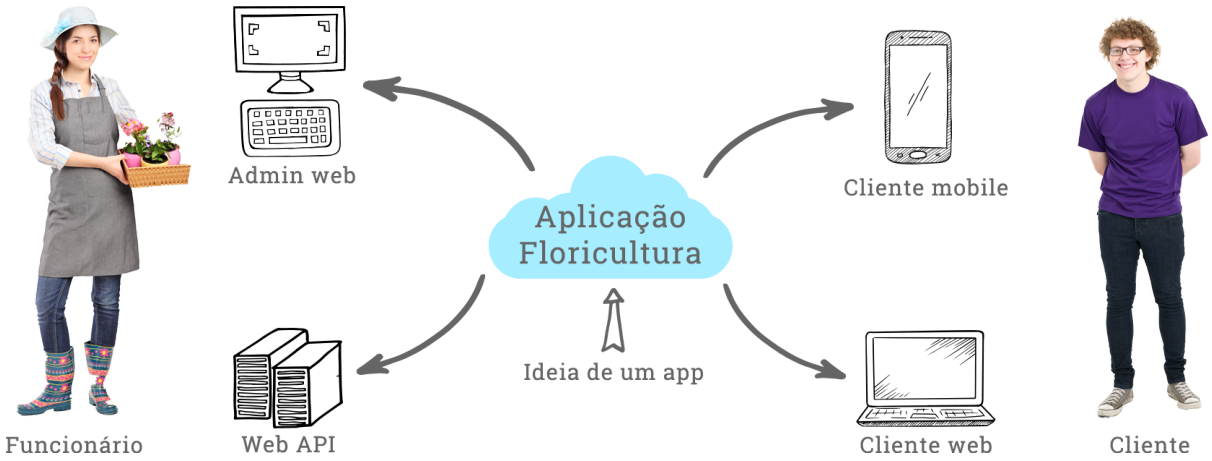


Figura 1. Tipos de aplicações

Cada uma dessas aplicações pode usar uma tecnologia diferente ou apenas o JavaScript.

Três tecnologias diferentes para entregar a mesma aplicação - não é muito legal, né? 🤔

O JavaScript é sensacional, pois permite entregar todas as aplicações que um negócio precisa com uma linguagem.

Ou seja, aprender JavaScript é um caminho mais curto para maiores salários!

Roda nativamente em todos os browsers

Isso é muito, muito importante.

O JavaScript é simplesmente a única linguagem interpretada pelos browsers (tanto desktop quanto mobile).

- **Sou obrigado a saber JavaScript para programar web?**
- Sim. Toda aplicação web roda em algum browser. O JavaScript é a única linguagem interpretada pelo browser então, em algum momento você vai ser OBRIGADO a aprender JavaScript. Simples assim. 😂
- **E se minha aplicação rodar somente no mobile, ou somente desktop?**
- Bom, se sua aplicação rodar somente no mobile (por exemplo, Uber), ou somente no Desktop, essa obrigatoriedade não vai existir. Mas, na nossa carreira de programador, é esperado que em algum momento tenhamos que entregar um cliente Web (mesmo que seja a parte admin da aplicação).

Então, aprender JavaScript vai estar no caminho da sua carreira, em algum momento. 🙂

É fácil

Essa é a boa notícia. 😉

Vamos lá. JavaScript é fácil por dois principais motivos:

A sintaxe é simples

JavaScript é baseado na linguagem C e é fracamente tipada. Isso significa que o código JavaScript é de fácil compreensão, como vemos no exemplo no **Código 1.**

```
var nome = "João";

var idade = 18;

var estaMatriculado = true;
```

Código 1. Exemplo de código de fácil compreensão.

A biblioteca é pequena

JavaScript possui uma biblioteca com poucas funções nativas.

Basicamente o que aprendemos com a linguagem é manipular textos, datas e fazer cálculos com funções matemáticas.

Depois de ter essa base, você consegue trabalhar com bibliotecas fornecidas pela comunidade e assim expandir as funcionalidades do seu projeto.

Com isso, aprendendo as funções nativas, você consegue dominar a linguagem rapidamente.

3. Característica: Tipagem mutável

Tipagem mutável significa poder alterar o tipo da variável ao longo do código, como no exemplo do **Código 1**.

```
var idade = 18;

idade = "18 anos";
```

Código 1. Exemplo de tipagem mutável

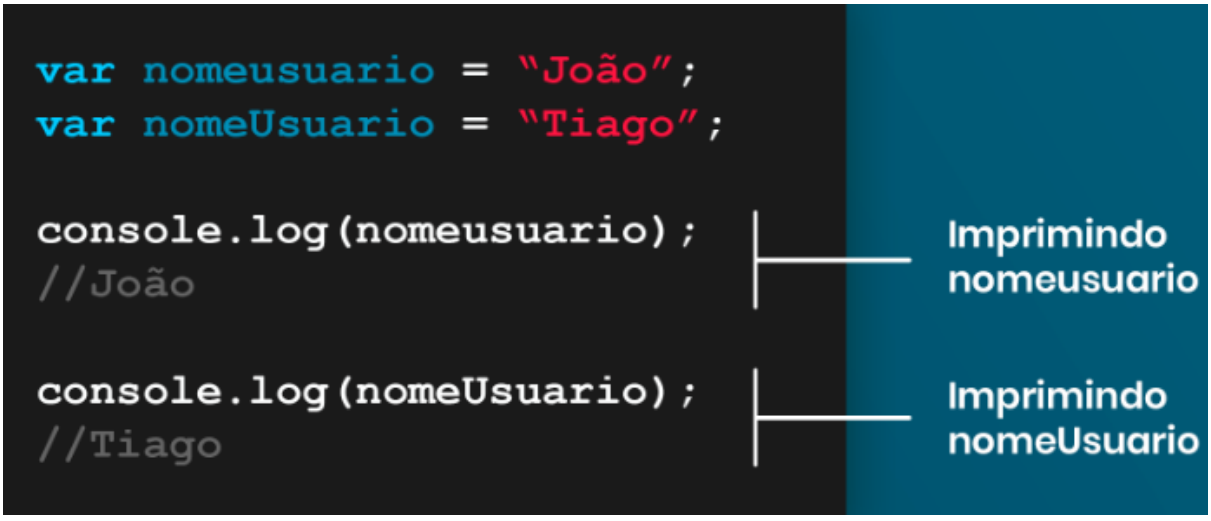
Como podemos ver, mudamos o valor da variável `idade`, que era do tipo numérico para o tipo texto.

A tipagem mutável acelera a curva de aprendizado porque deixa o programador iniciante com uma preocupação a menos na hora de escrever seus primeiros códigos.

Case Sensitive

O JavaScript vai diferenciar letras maiúsculas de minúsculas no código.

nome é diferente de Nome



6. Característica: A importância do JavaScript no browser

Nesse momento precisamos entender uma coisa importante sobre o JavaScript.

A palavra JavaScript, de acordo com o contexto, pode trazer significados diferentes. Vejamos:

1) JavaScript é uma linguagem 👍

Um primeiro cenário é o fato do JavaScript ser apenas uma linguagem de programação, ou seja, uma espécie de idioma que utilizamos para escrever códigos.

Por exemplo, existem outros idiomas, tais como as linguagens C#, PHP, Python, Pascal, etc.

Essa matéria trata justamente desse ponto: aqui vamos aprender a linguagem/idioma JavaScript - somente.

2) JavaScript manipula o DOM 👍

O segundo cenário é o fato de o JavaScript ser a única linguagem embutida nos browsers web.

Ou seja, se você quer programar um comportamento no browser, a única linguagem disponível para isso é o JavaScript.

Como assim "*comportamento no browser*"?

Por exemplo, digamos que você quer fazer algo mais ou menos assim: quando o usuário clicar em um determinado botão na página, um menu deve ser aberto, como vemos na **Animação 1**.

Animação 1. Abrindo um menu

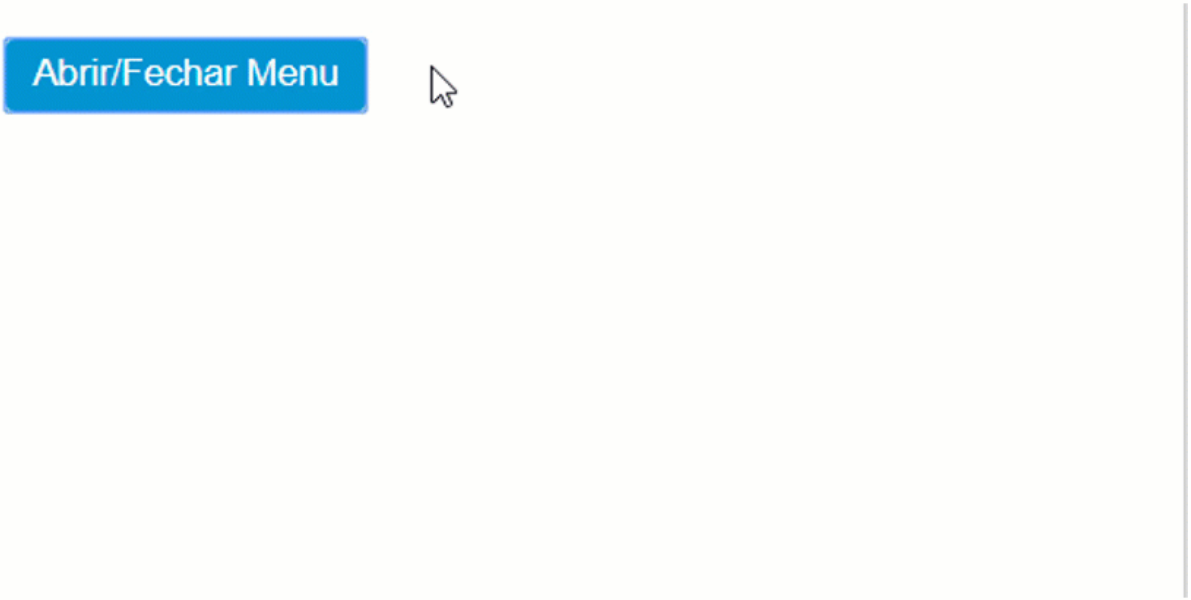
Isso é um evento no browser que necessita de uma programação. Isso só pode ser feito com JavaScript.

Tecnicamente, dizemos a mesma coisa assim: o JavaScript é a única linguagem que pode manipular o DOM.

DOM é a sigla para Document Object Model

Em palavras simples, quando falamos DOM queremos dizer o seguinte: que o JavaScript permite manipular elementos HTML como se eles fossem objetos.

Por exemplo, ao clicar no objeto botão (`HTML = elemento <button>elemento</button>`) um menu deve ser aberto (`HTML = elemento <nav>`), como vemos na **Animação 2**.



Animação 2. Abrindo e fechando um menu

Com JavaScript e DOM conseguimos:

- Alterar os elementos da página;
- Alterar os estilos, ou seja, cores, tamanho de fonte, …
- Excluir ou acrescentar um novo elemento;
- Criar SPAs, ou aplicações de página única, onde alteramos toda estrutura, sem recarregar a página.

Aprender a manipular o DOM não é assunto dessa matéria - aqui veremos apenas a sintaxe JavaScript.

Para se aprofundar nesse assunto recomendamos o uso de um framework, tais como o Angular, Vue.js ou React.

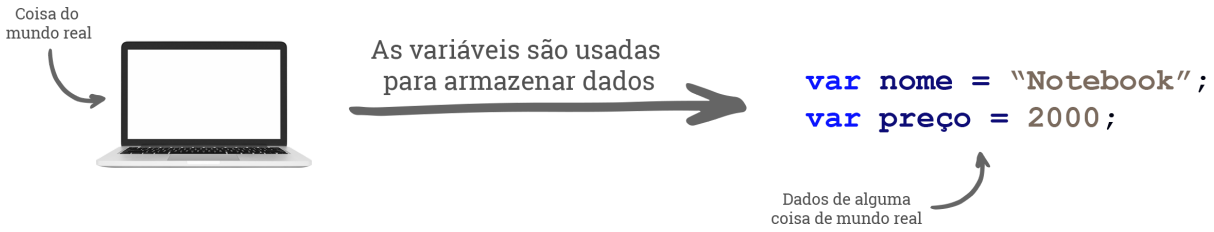
Através do DOM, o JavaScript consegue alterar toda estrutura da página.

1. Declarando uma variável

Voltar Suporte ao aluno Anotar Marcar como concluído

Muito bom te ver de volta 😊

Agora que você já tem um ambiente de programação configurado e funcionando, vamos aprender sobre as variáveis e os tipos de dados do JavaScript.



Var é de variável

Para declarar uma variável usamos a palavra chave `var`.

É bem simples:

```
var programador = "Eduardo";

var pontuacao = 10;
```

Código 1. Declarando uma variável

Para acessar o valor de uma variável basta utilizar o nome dela (**Código 2**).

```
console.log(programador);
// Imprime Eduardo
console.log(pontuacao);

// Imprime 10
```

Código 2. Acessando o valor de uma variável

Lembrando que a função `console.log()`, vista no curso "Hello World com a linguagem JavaScript", é usada para imprimir uma mensagem para o usuário no terminal.

Outros exemplos de declaração de variáveis

Criando uma variável com let

Outra forma de criar uma variável é utilizando a palavra chave let. Veja um exemplo no código abaixo.

```
let tecnologia = "JavaScript";

let anoAtual = 2021;
```

Código 3. Criando uma variável utilizado let.

Mas qual a diferença entre var e let?

Ao utilizar `var` conseguimos redeclarar uma mesma variável.

```
var nome = "José";

var nome = "Pedro";
```

Código 4. Redeclarando uma variável criada com var.

Já utilizando `let` isso não é possível e um erro vai ocorrer.

```
let nome = "José";
let nome = "Pedro";

Uncaught SyntaxError: Identifier 'nome' has already been
declared
```

Código 5. Erro gerado por tentar redeclarar uma variável criada com let.

Outra diferença está relacionada com o escopo que faz mais sentido com exemplos que empreguem `if`, `for`, `while` ou funções. Por esse motivo não abordaremos sobre isso neste momento.

Variáveis são usadas para armazenar dados

2. Tipos de dados: array

O array é uma coleção de dados e com esse recurso podemos colocar mais de um valor em apenas uma variável.

Declaração

Podemos declarar um array da seguinte forma (**Código 1**).

```
var estados = ["Rio de Janeiro", "São Paulo", "Bahia"];
```

Código 1. Exemplo de variável do tipo array

Vamos entender o código (**Figura 1**):

Figura 1. Explicando array

Acessando um valor do array

Na aula anterior, vimos que basta usar o nome de uma variável para acessar o seu valor (**Código 2**).

```
var produto = "Notebook";


console.log(produto);
```

Código 2. Acessando o valor de uma variável.

Utilizar `console.log()` dessa forma com um array não surtirá o efeito esperado, como vemos no **Código 3**.

```
var estados = ["Rio de Janeiro", "São Paulo", "Bahia"];


console.log(estados);




// (3) ["Rio de Janeiro", "São Paulo", "Bahia"]


// ...
```

Código 3. Utilizando a função console.log( ) em um array

🤔 interessante, mas não é o que planejamos 🤷‍♀️

Em um array, cada variável está numa posição específica, que é representada por um índice numérico. Sendo assim, para acessar um valor

específico usamos o índice da posição em que esse valor está. Fazemos isso com a sintaxe apresentada na **Figura 2**.

Figura 2. Posições no array

Por exemplo, para imprimir o texto "Rio de Janeiro", valor da variável na posição 0 do array, usamos o **Código 4**.

```
var estados = ["Rio de Janeiro", "São Paulo", "Bahia"];


console.log(estados[0]);
```

Código 4. Acessando o valor na posição 0 do array estados

Veja no **Código 5** outro exemplo, que agora imprimirá "São Paulo", valor da variável na posição 1 do array.

```
var estados = ["Rio de Janeiro", "São Paulo", "Bahia"];


console.log(estados[1]);
```

Código 5. Acessando o valor na posição 1 do array estados

Pegou o jeito? Vamos ver mais um exemplo no **Código 6**.

```
var linguagens = ["JavaScript", "PHP", "JAVA", "C#"];




console.log(linguagens[0]); //resultado impresso: JavaScript


console.log(linguagens[3]); //resultado impresso: C#
```

Código 6. Exemplo de array

Todo array começa com o índice 0, portanto para acessarmos o seu primeiro valor utilizamos nome_do_array[0]

Alterando um valor no array

Lembra que em um array cada variável está em uma certa posição? Então, uma vez que acessamos essa variável podemos fazer com ela operações comuns com variáveis, tais como modificar o valor dela.

Por exemplo, para modificar a linguagem "JAVA" para "C" no array linguagens utilizamos o **Código 7**.

```
var linguagens = ["JavaScript", "PHP", "JAVA", "C#"];

var linguagem = "C";

linguagens[2] = linguagem;
```

Código 7. Alterando o valor de uma posição no array

O que podemos fazer assim também, escrevendo menos como vemos no **Código 8** e na **Figura 3**.

```
var linguagens = ["JavaScript", "PHP", "JAVA", "C#"];

linguagens[2] = "C";
```

Código 8. Alterando o valor de uma posição no array

Execute o código

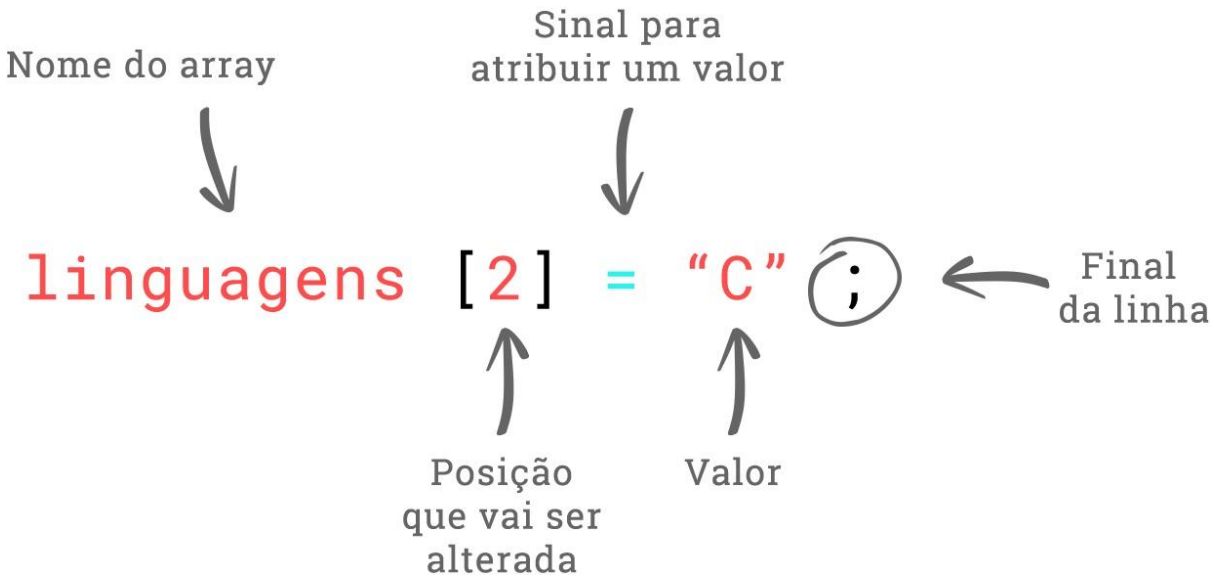


Figura 3. Acessando array com chaves

Fácil, né?

Armazenando tipos diferentes

Um array pode ser usado para armazenar tipos de dados diferentes.

Imagine que você precisa armazenar os dados de uma vaga (**Código 9**).

```
var titulo = "Programador";

var quantidadeDeVagas = 5;
```

```
var vagaAtiva = true;
```

Código 9. Armazenando tipos diferentes em um array

Veja na **Figura 4** como ficaria em um array.

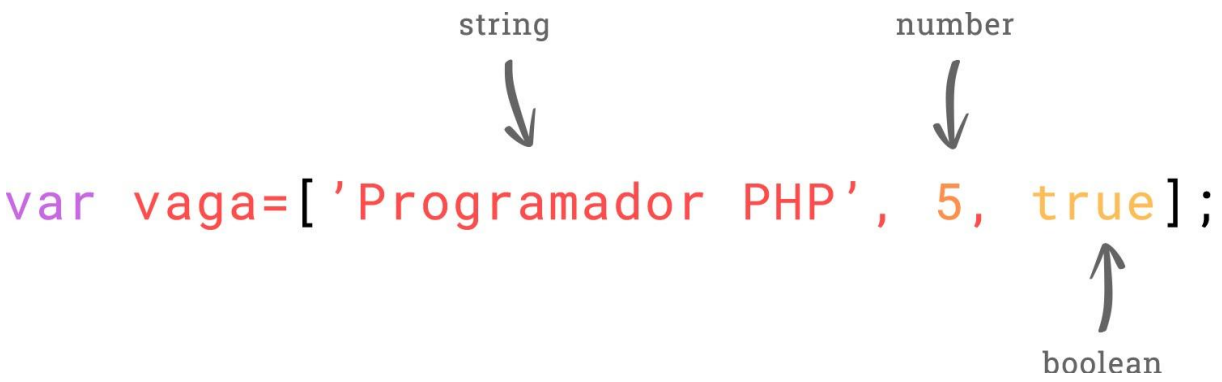


Figura 4. Exemplo em um array

Apesar de agrupar valores de tipos diferentes utilizando arrays ser possível, conseguimos fazer isso melhor com objetos, o que aprenderemos no próximo curso.

Contando a quantidade de elementos

Existem casos em que precisamos saber quantos elementos tem dentro do array. Isso é feito utilizando `.length` depois do nome do array. Veja um exemplo no **Código 10**:

```
var linguagens = ["JavaScript", "PHP", "JAVA", "C#"];



console.log(linguagens.length);

                    // Vai imprimir 4
```

Código 10. Contanto elementos de um array

No código acima utilizamos `.length` para contar a quantidade de elementos do array `linguagens`.

Quando usar arrays?

Um exemplo: digamos que um usuário possui dois telefones. Como armazenar esses dados? Uma primeira ideia seria usar duas variáveis:

Figura 5. Armazenando dois telefones em variáveis

Isso resolve. Mas e se agora ele tiver três telefones? Precisaríamos mudar nosso código-fonte criando uma terceira variável, o que não é uma solução elegante.

O uso de array nesse caso é perfeito, veja:

Cenário 1 (**Código 11**):

```
var telefones = [

    '(11) 98899 - 8787',
```

```
    '(22) 3455 - 8819'

                                    ];
```

Código 11. Array com dois telefones

Cenário 2 (**Código 12**):

```
var telefones = [

    '(11) 98899 - 8787',

    '(22) 3455 - 8819',

    '(91) 95620 - 0000'

                                    ];
```

Código 12. Array com três telefones

Observe que com o mesmo código atendemos as duas situações, modificando apenas os valores.

Veja outros exemplos de uso de array:

Com o array conseguimos armazenar em uma variável mais de um valor em comum.

4. Tipos de dados:

undefined

No JavaScript, `undefined` e `null` podem ser utilizados para representar a ausência de valor de uma variável.

Vamos ver como eles funcionam começando com undefined. Veremos null na próxima aula.

Undefined

Anteriormente, vimos que uma variável é do tipo `number` quando atribuímos a ela um número, `string` quando ela contém um texto e `boolean` se seu valor é `true` ou `false`. Então, o que acontece no caso abaixo? Qual é o tipo e o valor da variável `nome`?

```
var nome;
```

Código 1. Variável criada sem receber um valor

Quando for esse o caso, o JavaScript dará a variável o valor undefined.

Veja um exemplo no **Código 2**.

```
var nome;


console.log(nome); //vai imprimir undefined
```

Código 2. Imprimindo uma variável que não recebeu um valor

O que é undefined?

É tanto o nome de um tipo quanto o nome do único valor que esse tipo pode receber.

Mas qual é o problema de uma variável ser undefined?

Vamos voltar a variável nome. Digamos que alguém tente contar quantas letras essa variável possui, presumindo que o seu tipo é string. Usamos a propriedade `length para isso, que toda string possui, como mostra o **Código 3.**`

```
console.log(nome.length);
```

Código 3. Tentando acessar a propriedade length de uma variável undefined

Uma variável undefined não é uma string e não possui a propriedade length, o que vai gerar um erro, como vemos no **Código 4**.

```
TypeError: Cannot read property 'length' of undefined
```

Código 4. Erro gerado por tentar acessar a propriedade length de uma variável undefined

Uma das formas de resolver esse erro é inicializando a variável (**Código 5**).

```
var nome = '';
```

Código 5, Inicializando uma variável do tipo string

Desse jeito seu código vai funcionar.

Um outro problema acontece quando você faz uma operação matemática com um valor undefined (**Código 6**).

```
var idade;
```

```
console.log( idade + 1 );
```

Código 6. Manipulando uma variável undefined com operador matemático

O resultado será NaN (Not a Number), não é um número.

NaN é o resultado de uma operação matemática que falhou.

Esse erro também pode ser evitado se atribuímos um valor ao criar a variável como vemos no **Código 7**.

```
var idade = 0;
```

Código 7. Inicializando uma variável do tipo number

Feito isso você poderá dormir tranquilo 🤣🤣🤣

Uma variável será undefined quando não for atribuído um valor a ela.

5. Declarando uma constante

Nessa aula veremos como proteger valores que nunca deveriam mudar utilizando a palavra-chave const.

Algumas coisas mudam... outras não

No código de uma aplicação é fácil encontrar valores que nunca devem mudar. Uma url, PI, um percentual de desconto, etc.

É uma boa prática declarar esses valores utilizando a palavra-chave const, como no **Código 1**:

```
const url = "https://www.devmedia.com.br/";
```

Código 1. Declarando uma constante

Logo que definimos uma const precisamos dar um valor para ela, porque caso isso não aconteça, será gerado um erro, como vemos no **Código 2**.

```
const nome;
```

```
SyntaxError: Missing initializer in const declaration
```

Código 2. Declarando uma constante sem atribuir valor a ela

Outra diferença é que uma vez definido esse valor não conseguimos mais alterar e se isso for feito também vai gerar um erro (**Código 3**).

```
const aula = "JavaScript";
aula = "JS";
```

```
TypeError: Assignment to constant variable.
```

Código 3. Declarando uma constante sem atribuir valor a ela

const e a ideia de imutabilidade

Uma das vantagens do uso de const é o conceito de imutabilidade, que quer dizer "manter o mesmo valor" ou "não mudar".

Por que é bom que as coisas não mudem? Imagine que você declarou uma variável no início do seu código (**Código 5**) e foi usá-la muitas linhas abaixo. Como ter certeza que ela não foi alterada sem querer?

```
var url = ”https://api.com.br/usuarios”;

// código
// código
// e mais código

url = “https://api.com.br/registros”;

// código
// código
// e mais código
```

```
console.log(url);
```

Código 5. Imprimindo uma variável que foi alterada

Utilizando const teremos a certeza de que isto não vai acontecer, como vemos no **Código 6**.

```
const url = ”https://api.com.br/usuarios”;

// código
// código
// e mais código

url = “https://api.com.br/registros”;

// Neste ponto o código vai quebrar

Uncaught TypeError: Assignment to constant variable.
```

Código 6. Tentando alterar uma const

No JavaScript veremos muito o uso de const por causa desse conceito de imutabilidade.

const é usado sempre que não precisamos fazer alterações a uma variável.

1. Por que eu preciso de operadores?

Parabéns, mais um avanço na sua carreira como programador JavaScript! 😊

Anteriormente você aprendeu como atribuir um valor a uma variável diretamente, dessa forma: `var idade = 18`. Meio sem graça, não é?

O que são operadores!

Os operadores são símbolos usados para modificar ou até mesmo gerar um novo valor. Na programação, em muitos casos, precisamos que um valor seja gerado a partir da análise, combinação ou comparação entre valores e são os operadores que fazem isso:

```
var atendeAClassificacaoEtária = idade >= 18
```

Código - Exemplo do uso de operadores.

O que vamos aprender?

As próximas aulas tratarão dos operadores, agrupando-os nos seguintes conjuntos:

- Aritméticos
- Atribuição
- Incremento
- Comparação, ou relacionais
- Lógicos