



JavaScript



Aula 1



Olá

Rafael Rodrigues

Engenharia de Computação

1 ano e 8 meses na CJR

5º Semestre

<https://www.linkedin.com/in/rafael2903>

<https://github.com/rafael2903>



O que veremos



- [Plano de ensino](#)
- [Conteúdo programático e umas anotações bem bagunçadas](#)

Acordos



- O conteúdo é bastante extenso, então vou ter que passar rápido em algumas partes, principalmente as coisas mais básicas de lógica de programação, mas caso não entenda algo, pergunte!
- Podem me interromper a qualquer momento, abrindo o microfone ou levantando a mão.
- Sintam-se livres para tirar dúvidas pelo slack fora do horário da aula.
- Tentei sempre colocar referências e links para mais conteúdo nos slides.
- Se eu travar ou vocês não entenderem o que eu falar, me avisem :)

Sites para praticar



[beecrowd](#)

[Exercism](#)

[Codewars](#)

[HackerRank](#)

[leetcode](#)

[DevChallenge](#)



JavaScript, o que é? Onde vive? O que come?



JavaScript é uma linguagem de programação que permite implementar funcionalidades mais complexas em páginas web. Sempre que uma página web faz mais do que apenas mostrar informações estáticas para você - ela mostra em tempo real conteúdos atualizados, mapas interativos, animações gráficas em 2D/3D, vídeos, etc. - você pode apostar que o Javascript provavelmente está envolvido.

[Uma breve história sobre Javascript ↗](#)





JavaScript, o que é? Onde vive? O que come?



Exemplos:

[Meu jogo da velha ↗](#)

[Trello ↗](#)





JavaScript, o que é? Onde vive? O que come?



Mais conhecida como a linguagem de script para páginas Web, mas usada também em vários outros ambientes sem browser, como `node.js`.



JavaScript, o que é? Onde vive? O que come?



- Linguagem leve
- Interpretada
- Baseada em objetos e em protótipos
- Com funções de primeira classe (HOF)
- Tipagem dinâmica e fraca
- Case sensitive
- Multi-paradigma, suportando estilos de orientação a objetos, imperativos e declarativos (como por exemplo a programação funcional).





JavaScript, o que é? Onde vive? O que come?



- Interpretada

"Linguagem interpretada é uma linguagem de programação em que o código fonte nessa linguagem é executado por um programa de computador chamado interpretador, que em seguida é executado pelo sistema operacional ou processador." Diferente de uma linguagem compilada.



JavaScript, o que é? Onde vive? O que come?



- Tipagem dinâmica

Na Tipagem Dinâmica, os tipos são verificados durante a execução, o que significa que os tipos são associados a valores encontrados em tempo de execução e não a nomes pré definidos no momento de escrita do código fonte.

```
let nome = "Rafael"  
typeof nome // 'string'
```

```
nome = 3  
typeof nome // 'number'
```



JavaScript, o que é? Onde vive? O que come?



- Tipos

Na TI
durante
são a
de ex
mom

```
let nome  
typeof
```

```
nome  
typeof
```

Obs: por causa desse "problema",
foi criada a linguagem TypeScript.

Ela é um superconjunto sintático
estrito de JavaScript que adiciona
tipagem estática opcional à
linguagem.

Talvez vocês aprenderam ela mais
pra frente no PT.

```
nome // number
```



JavaScript, o que é? Onde vive? O que come?



- Tipagem fraca

Na Tipagem fraca, o tipo do dado não é bem definido, a linguagem pode alterar o tipo por alguma ação, sem intervenção direta do programador.

```
const numberOne = "5"  
const numberTwo = 5
```

```
numberOne + numberTwo
```

```
// 55
```





JavaScript, o que é? Onde vive? O que come?



- Case sensitive

Linguagens case sensitive são linguagens que diferenciam maiúsculo de minúsculo.

```
const a = "a"  
const A = "a"
```



Atenção

Apesar de terem nomes parecidos, JavaScript não tem nada haver com Java. São linguagens de programação que possuem sintaxe, semânticas e usos muito diferentes.



≠





Dúvidas?

**Vamos
começar!**



Como adicionar o JS ao HTML?

[Por que ele foi adicionado no final do body? ↗](#)

Ele é adicionado por meio da tag `<script>`. Existem algumas formas de utilizá-la:

```
<body>
  <p>parágrafo qualquer</p>
  <script>
    console.log("Hello World!")
  </script>
</body>
```

```
<body>
  <p>parágrafo qualquer</p>
  <script src="./script.js"></script>
</body>
```

```
<head>
  <title>Título</title>
  <script src="./script.js" defer></script>
</head>
```



0 Console

Alguns métodos globais

- `console.log('mensagem')` => mostra uma mensagem no Console
- `alert('mensagem')` => mostra um pop-up com uma mensagem
- `prompt('mensagem')` => mostra um pop-up com uma mensagem e um input em que o usuário pode inserir um texto
- `confirm('mensagem')` => mostra um pop-up com uma mensagem em que o usuário pode confirmar ou cancelar uma ação



Variáveis

Uma variável é um local nomeado para armazenar um valor. Dessa forma, um valor pode ser acessado através de um nome predeterminado.

No JavaScript, temos 3 formas de declarar variáveis:

let

const

~~**var**~~

Primeiro, uma diferenciação

- Declaração de uma variável
`let variavel`
- Inicialização de uma variável
`variavel = 2`
- Declaração e inicialização
`let variavel = 2`



let

- Permite declarar uma variável no escopo do bloco atual.
- Pode ou não ser inicializada ao ser declarada.
- Pode ter seu conteúdo alterado.

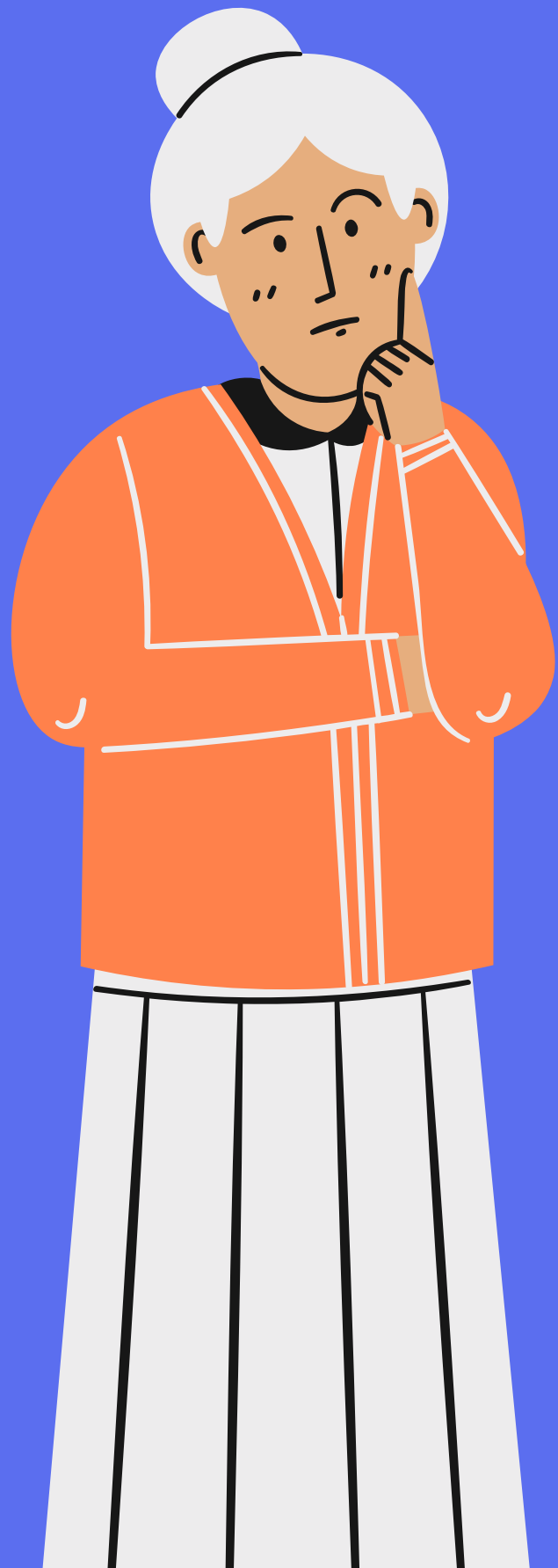
Só se utiliza quando precisamos ter uma variável com valor mutável, caso contrário, utiliza-se const

[referência ↗](#)

const

- Permite declarar uma variável no escopo do bloco atual.
- Obrigatoriamente tem de ser inicializada ao ser declarada.
- Seu conteúdo é fixo, não pode ser alterado depois de inicializada.

[referência ↗](#)



Blocos e Escopo

Blocos

- Blocos são utilizados para agrupar uma ou mais expressões.
- No JavaScript, são delimitados por chaves (`{...}`).
- É equivalente a indentação utilizada no python.

Escopo

- Escopo se refere ao conjunto de funções e variáveis que estão disponíveis para uso (ou seja, que podem ser referenciadas).

var

- Não é recomendada a sua utilização, pois pode levar a bugs.
- No dia a dia, só utilizamos let ou const
- Era o padrão nas versões antigas do JS, quando não existia let e const.
- É parecida com o let.

[A diferença entre var e let ↗](#)
[referência ↗](#)





Case Styles (no JS)

camelCase

nome de variáveis e funções

PascalCase

nome de classes
nome de componentes (React)
nome de tipos e interfaces (TypeScript)

kebab-case

URLs
nomes de arquivos
CSS

snake_case

não se utiliza muito em JS
comum em C, ruby e python



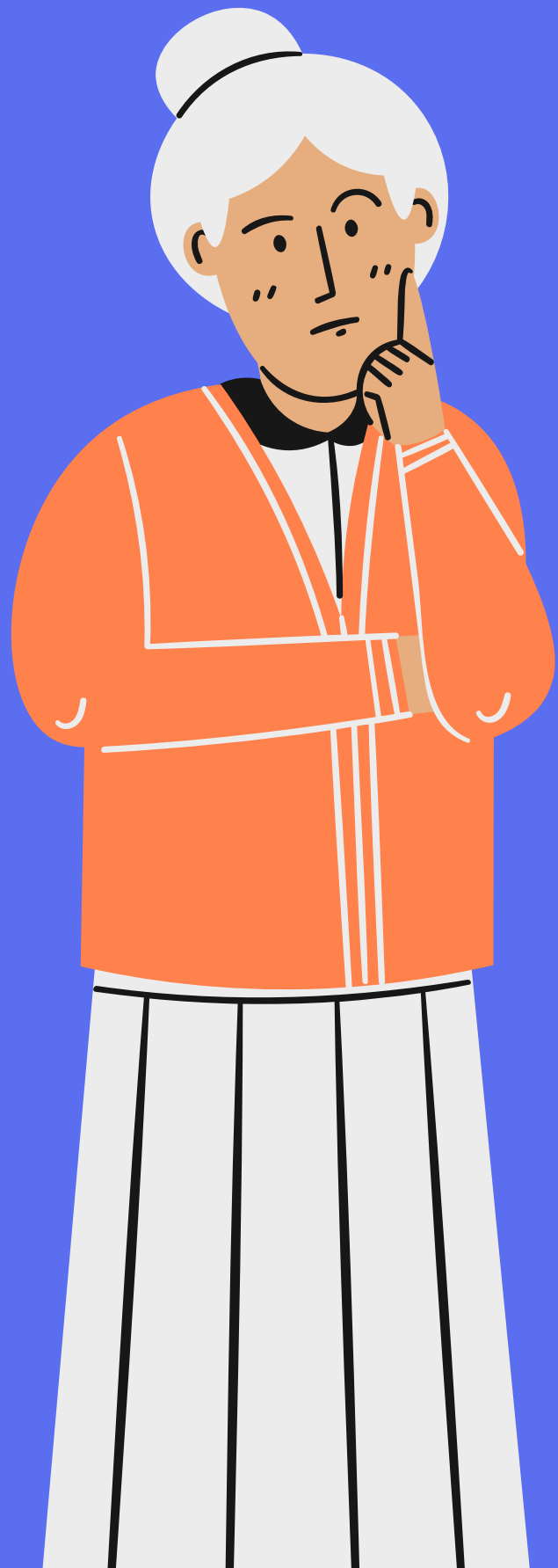
Case Styles (no JS)

SCREAMING_SNAKE_CASE

(ou UPPER_SNAKE_CASE)

constates e variáveis de ambiente

-
- Esse é o padrão mais utilizado, mas existem vários padrões, e não tem certo e errado, o mais importante, é definir um padrão com a equipe e segui-lo.



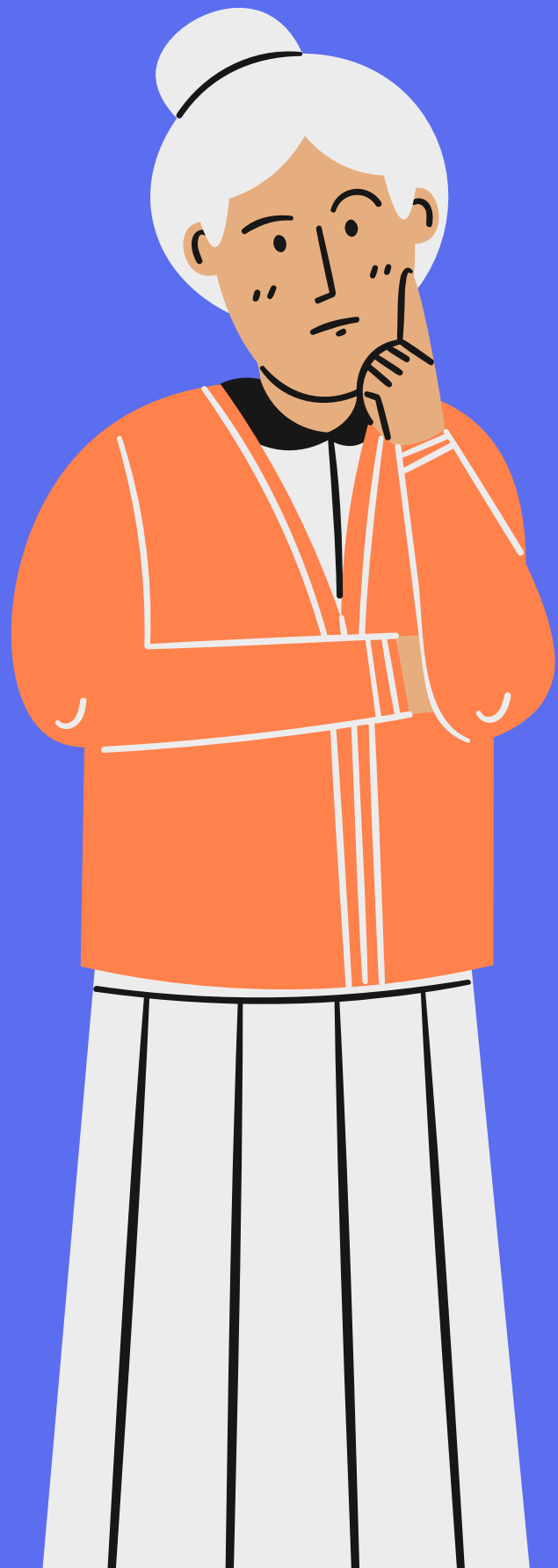
Comentários

- Os comentários são usados quando queremos que o navegador não interprete partes do código, ou para adicionar notas ou documentação.
- No JS, temos duas formas de comentar:
 - uma única linha: `// código legal`
 - múltiplas linhas:

```
/*  
código  
mais código  
outra linha de código  
*/
```



Você pode usar o atalho do VS Code *Ctrl + ;* (Windows) ou *Ctrl + .* (Linux) para comentar.



Ponto e vírgula (semicolon)

- Antigamente, toda instrução JavaScript tinha que terminar com ponto e vírgula (;).
- Hoje em dia, seu uso é opcional*, já que o navegador consegue determinar sozinho onde coloca-lás.
- Apesar disso, alguns style guides, como o da [airbnb](#) e o do [google](#), recomendam sua utilização.
- O importante é definir com o time se vocês usarão ou não.

* Opcional só quando temos somente 1 instrução por linha. Se tiver mais de 1, seu uso é obrigatório.



Tipos primitivos





- "Um tipo de dados primitivo é um dado que não é representado através de um Objeto e, por consequência, não possui métodos."
- "Na maior parte do tempo, um valor primitivo é representado diretamente através do mais baixo nível da implementação de uma linguagem."
- Resumindo: são tipos básicos que não podem ser representados por outros tipos.
- No JavaScript, existem 6 tipos primitivos:
 - String
 - Number
 - Boolean
 - Null
 - undefined
 - ~~Symbol~~



String

- Uma string é uma sequência de caracteres usados para representar texto.
- Ela é envolvida por aspas simples ('string') ou duplas ("string").

Template Strings

- Template Strings são strings que permitem expressões embutidas. Você pode utilizar string multi-linhas e interpolação de string com elas.
- São envolvidas por acentos graves (`string`). Utiliza-se o cifrão seguido de chaves para inserir expressões.

```
`Seu nome é: ${nome}` // sem template strings: "Seu nome é " + nome
```

```
`${string1}${string2}` // sem template strings: string1 + string2
```



Number

- Em outras linguagens de programação, diferentes tipos numéricos podem existir, por exemplo:
 - Integers (Inteiros). Ex: 44, -15
 - Floats (Pontos Flutuantes), tem pontos e casas decimais. Ex: 12.523
 - Doubles (Dobros), igual float, mas com mais casas decimais possíveis
- Em JavaScript, todos os números são representados como doubles:

```
typeof -15 // 'number'  
typeof 12.5 // 'number'  
typeof 26.8765434567 // 'number'
```



Boolean

- Um booleano, é um tipo de dado lógico que pode ter apenas um de dois valores possíveis: `true` (verdadeiro) ou `false` (falso).
- São usados em condicionais.



Null

- Representa um valor nulo ou "vazio".
- **null** é utilizado quando se quer intencionalmente dizer que há uma ausência de qualquer valor para uma variável.
- É o equivalente ao nil, None ou NULL de outras linguagens.



undefined

- É o valor atribuído automaticamente para variáveis que foram declaradas, mas não foram inicializadas.
- Indica que o valor da variável não foi definido.

[Qual a diferença entre null e undefined? ↗](#)



Symbol

[referência ↗](#)



Dúvidas?

Operadores

[referência ↗](#)





Operadores aritméticos

Significado	Operador
Soma (ou concatenação)	+
Subtração	-
Multiplicação	*
Divisão	/
Resto	%
Exponenciação	**



Operadores aritméticos

Operador	Significado	Uso
++	Adiciona um ao seu operando	++x ou x++
--	Subtrai um de seu operando.	--x ou x--

Qual será a saída no console para o seguinte código?

```
console.log('rafael' / 2)
```





NaN

- Not-A-Number (não é um número)
- Valor retornado quando se tenta fazer operações matemáticas inválidas.

[referência ↗](#)



Operadores de atribuição

Nome	Operador encurtado	Significado
Atribuição	$x = y$	$x = y$
Atribuição de adição	$x += y$	$x = x + y$
Atribuição de subtração	$x -= y$	$x = x - y$
Atribuição de multiplicação	$x *= y$	$x = x * y$
Atribuição de divisão	$x /= y$	$x = x / y$
Atribuição de resto	$x \% = y$	$x = x \% y$
Atribuição exponencial	$x ** = y$	$x = x ** y$



Operadores de comparação

- Um operador de comparação compara seus operandos e retorna **true** ou **false** baseado em se a comparação é verdadeira.
- Na maioria dos casos, se dois operandos não são do mesmo tipo, o JavaScript tenta convertê-los para um tipo apropriado.



Operadores de comparação

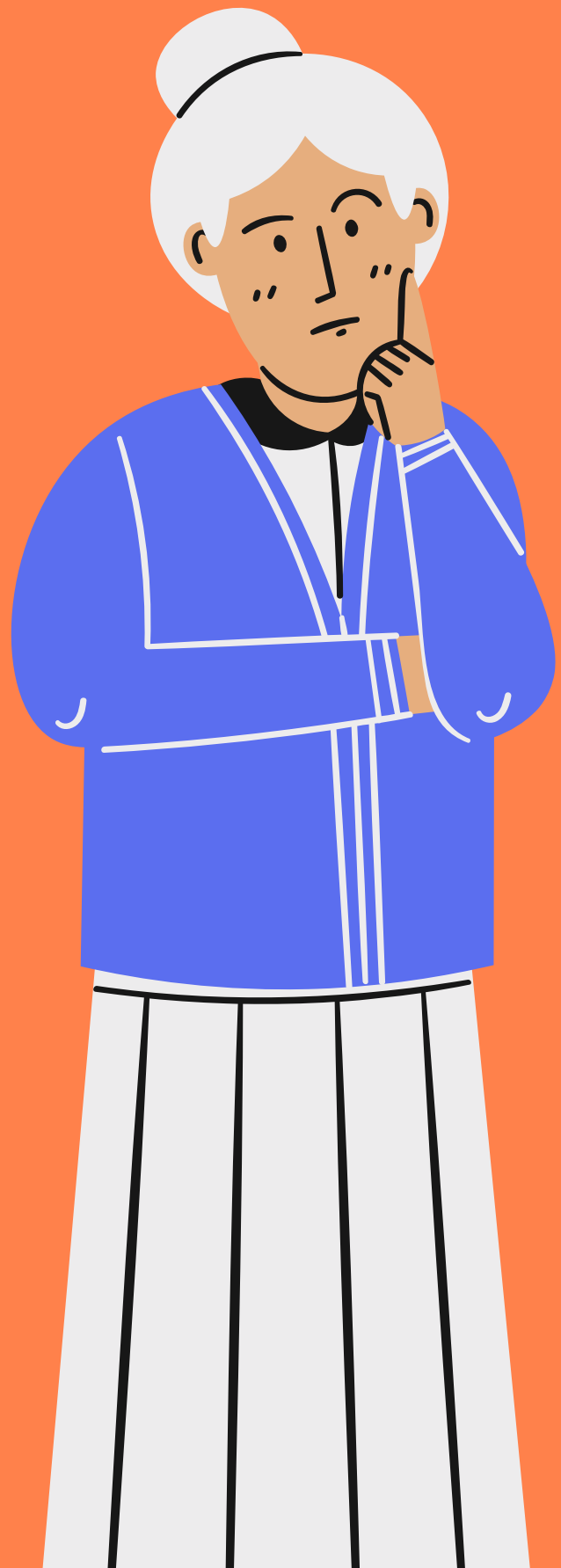
Operador	Descrição	Exemplos que retornam true
Igual (==)	Retorna verdadeiro caso os operandos sejam iguais.	3 == var1 3 == "3"
Não igual (!=)	Retorna verdadeiro caso os operandos não sejam iguais.	var1 != 4 var2 != "3"
Estritamente igual (===)	Retorna verdadeiro caso os operandos sejam iguais e do mesmo tipo.	3 === 3
Estritamente não igual (!==)	Retorna verdadeiro caso os operandos não sejam iguais e/ou não sejam do mesmo tipo.	3 !== "3"



Operadores de comparação

Operador	Descrição	Exemplos que retornam true
Maior que (>)	Retorna verdadeiro caso o operando da esquerda seja maior que o da direita.	<code>var2 > var1</code> <code>"12" > 2</code>
Maior que ou igual (>=)	Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita.	<code>var2 >= var1</code> <code>'4' >= 3</code>
Menor que (<)	Retorna verdadeiro caso o operando da esquerda seja menor que o da direita.	<code>"alou" < "bola"</code> <code>2 < 4</code>
Menor que ou igual (<=)	Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita.	<code>var1 <= var2</code> <code>var2 <= 5</code>





Valores falsy

- Um valor falsy é um valor que se traduz em falso quando avaliado como Boolean.
- São valores falsy:
 - `false`
 - `null`
 - `undefined`
 - `0`
 - `NaN`
 - `''`



Valores truthy

- Um valor truthy é um valor que se traduz em verdadeiro quando avaliado como Boolean.
- Todos os valores são truthy a menos que eles sejam definidos como falsy
- São valores truthy:
 - `true`
 - `{}`
 - `[]` e `[1, 2]`
 - `42`
 - `-3.14`
 - `Infinity` e `-Infinity`
 - `'0'` e `'rafael'`



Operadores lógicos (na prática)

Operador	Descrição	Exemplo
&& (AND)	Retorna verdadeiro se todos os operandos forem verdadeiro.	false && false => false true && false => false false && true => false true && true => true
(OR)	Retorna verdadeiro se ao menos um dos operandos forem verdadeiros.	false false => false true false => true false true => true true true => true
!(NOT)	(Negação) Retorna false se o valor for truthy e true se o valor for falsy.	!true => false !false => true !0 => true

Qual será a saída no console para os seguintes códigos?

```
console.log(0 || 'rafael')
```

```
console.log(0 && 3)
```





Operadores lógicos (for real)


- A última tabela na verdade está imprecisa! Os operadores `&&` e `||` são mais complicados que isso.
- Você pode pensar daquele jeito em diversos contextos, como nas condicionais de um `if`, `for loop`, `while`, operador ternário (`? :`).
- Mas em alguns contextos, como em atribuições condicionais ou renderizações condicionais (em React), pensar daquele jeito não funciona.

[Como `&&` e `||` realmente funcionam ↗](#)



Atribuições condicionais

- Podemos usar `&&`, `||` e `??` para atribuir valores à variáveis condicionalmente, com uma sintaxe mais enxuta que um `if`.



Atribuições condicionais com &&

- Nesse exemplo, `firstElement` vai receber o valor de `array` se ele for um valor falsy e `array[0]` caso contrário.
- Essa sintaxe é útil quando queremos acessar uma coisa que não temos certeza se está definida, evitando o lançamento de erros.
- Esse código é equivalente a esse:

```
const array = [1,2,3]
const firstElement = array && array[0]
```

```
const array = [1, 2, 3]
if (array) {
  const firstElement = array[0]
}
```

Atribuições condicionais com ||

- Nesse exemplo, `age` vai receber o valor de `age` (ou seja, vai manter seu valor) se ele for um valor `truthy`, caso contrário, receberá o valor 19.
- Essa sintaxe é útil quando queremos definir valores padrões ou [fallbacks](#) ↗.
- Esse código é equivalente a esse:

```
let age // undefined
age = age || 19
// age == 19
```

```
let age
if (!age) {
  age = 19
}
```



Atribuições condicionais com ??

- ?? é o operador de coalescência nula (nullish coalescing operator).
- É semelhante ao operador ||.

[referência ↗](#)

Qual será a saída no console para o seguinte código?

```
false && console.log( 'mensagem' )
```





Avaliação de curto-circuito

- Como *false* **&&** *qualquercoisa* é sempre **false** e *true* **||** *qualquercoisa* é sempre **true**, o JavaScript não avalia "qualquercoisa".
- Ou seja, se tivermos *false* **&&** *console.log('mensagem')*, a mensagem não é mostrada no console.



Dúvidas?



Condicionais

if e else

- A condicional *if* é uma estrutura condicional que executa a afirmação, dentro do bloco, se determinada condição for verdadeira. Se for falsa, executa as afirmações dentro de *else*.

```
if (price > 25) {  
    discount = 0.1  
} else if (price > 35) {  
    discount = 0.2  
} else {  
    discount = 0  
}
```

[referência ↗](#)



Fim!

