BRUNNA CROCHES

PARTE 1

JAVA SCRIPT

Artist Image :Descourtilz, Jean-Théodore | Dates:179?-1855

Guia iniciante: JavaScript

ABOUT ME

Brunna Croches

Developer Full Stack

Brunna Croches é Dev FullStack, advogada e empreendedora.

Apaixonada por tech, vem adquirido vasto conhecimento na área.

Desenvolveu projetos ricos em diversidade, buscando captar as próximas tendências e necessidades do mercado.

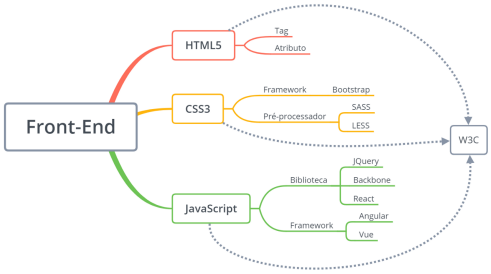
Neste e-book você aprenderá ou recapitulará de forma simplificada e otimizados conceitos de programação feito por ela.

let's share

SUMMARY

1.0 Funções e Variáveis

Característica do JavaScript



O JavaScript é considerado uma Linguagem Flexível

é correto programar em JavaScript :

- 1. De forma estruturada;
- 2. Utilizando orientação a objetos;
- 3. Com o paradigma funcional.

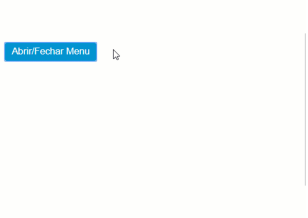
Programação Estruturada	var primeiroNumero = 9;Tabela var segundoNumero = 4; console.log(primeiroNumero + segundoNumero);

Programação utilizando funções	var primeiroNumero = 9;
	var segundoNumero = 4;
	var somarNumero =
	function(primeiroNumero, segundoNumero){
	return primeiroNumero + segundoNumero;
	}
	console.log(somarNumero (primeiroNumero,
	segundoNumero));

Programação utilizando classes com o framework react native	class Calculadora extends Component {
	constructor(props){
	super(props);
	this.somar=this.somar.bind(this);
	}
	somar(n1,n2){
	return n1+n2;
	}
	}

A importância do JavaScript no browser

JavaScript manipula o DOM	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**	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.**

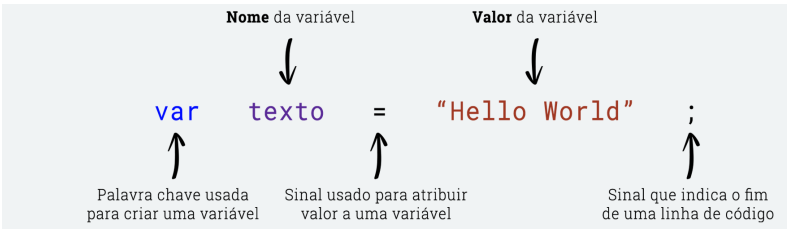
<table>
<tr><td></td><td> 
</td><td></td></tr>
<tr><td>DOM</td><td>DOM é a sigla para Document Object Model
Em palavras simples, quando falamos DOM queremos dizer o seguinte: que o JavaScript permite manipular elementos HTML como se eles fossem objetos.
Por exemplo, ao clicar no objeto botão (HTML = elemento <button>elemento</button>) um menu deve ser aberto (HTML = elemento <nav>)</td><td> 

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.</td></tr>
</table>

Variáveis e funções

<table>
<tr><td>var

VARIÁVEIS</td><td>As variáveis são usadas para armazenar dados.

No JavaScript, para criar uma variável usamos a palavra var, seguida do nome da variável, depois um sinal de igual =, para indicar que vamos atribuir um valor para ela, e o valor.</td><td> 

var texto = "Hello World";</td></tr>
<tr><td>function

FUNÇÕES</td><td>As funções são blocos de código que realizam uma tarefa. Com elas reaproveitamos um código, ou seja, escrevemos uma rotina e utilizamos mais de uma vez.</td><td>Em seguida, vamos criar uma função que imprime um texto. Utilizar funções evita repetições de código e o deixa mais organizado.
Para criar uma função utilizamos a palavra function, seguida do nome da função, parênteses ( ) e chaves { }.
Veja como vai ficar:

function imprimirTexto() { }</td></tr>
</table>

( )

PARA IMPRIMIR

FUNÇÕES

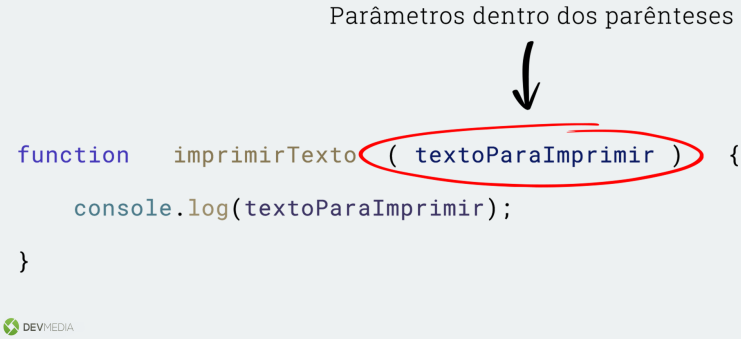
Palavra usada para criar uma função

Precisamos **enviar o texto que queremos imprimir para a função** e, para isso, vamos passar um parâmetro para ela, que deve ser colocado entre os parênteses ( ).

function imprimirTexto(textoParaImprimir) {}

Nome da função

Parâmetros dentro dos parênteses



console.

CODIGO DAS FUNÇÕES

Agora precisamos **criar o código que vai ser executado quando chamarmos esta função.**

console.log("Hello World");

Mas no lugar do texto, vamos colocar o parâmetro que a função vai receber. Vai ficar assim:

```
function imprimirTexto(textoParaImprimir) {
  console.log(textoParaImprimir);
}
```

```
function   imprimirTexto  ( textoParaImprimir )   {
    console.log(textoParaImprimir);
}
```

Código da função

DEVMEDIA

Determina o início da função

```
function   imprimirTexto  ( textoParaImprimir )  {
    console.log(textoParaImprimir);
}
```

Determina o fim da função

DEVMEDIA

Declarando uma Variável

var **Declarar uma VARIÁVEL**	Para declarar uma variável usamos a palavra chave var. É bem simples: **var programador = "Eduardo";** **var pontuação = 10;**
var **Acessar o valor de uma VARIÁVEL**	Para acessar o valor de uma variável basta utilizar o nome dela. **console.log(programador);** **// Imprime Eduardo** **console.log(pontuação);** **// Imprime 10** 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.

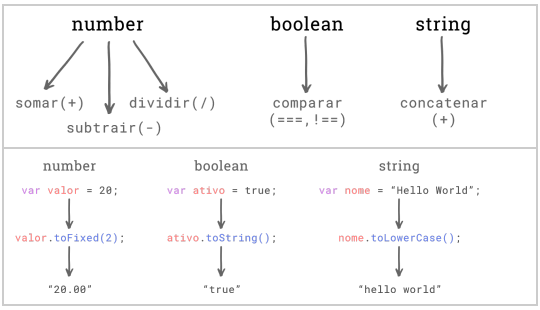
Outros exemplos de declaração de variáveis	var idade = 32;
	console.log(idade + "anos");
	var valor =25.99;
	var desconto = 5;
	var precoFinal = valor - desconto;
	console.log("Valor final = " +precoFinal);

let	Outra forma de **criar uma**

Criando uma VARIÁVEL com LET	**variável é utilizando a palavra chave let.** Veja um exemplo no código abaixo. **let tecnologia = "JavaScript";** **let anoAtual = 2021;**
Diferença entre VAR e LET	**Ao utilizar var conseguimos re-declarar uma mesma variável.** var nome = "José"; var nome = "Pedro"; **Re-declarando 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

---

Tipos de dados: string, number e boolean



```
var nome = "Willian";
console.log(nome);
```


```
var idade = 32;
console.log(idade + "anos");
```


```
var valor = 25.99;
var desconto = 5;
var precoFinal = valor - desconto;

console.log("Valor final = " + precoFinal);
```

DEVMEDIA

Tipos de dados array, undefined, null

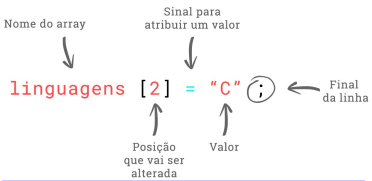
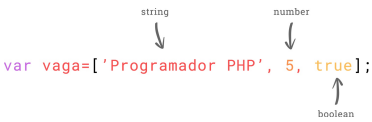
Array

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"];**
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);**	
console. O JEITO ERRADO **CODIGO DAS FUNÇÕES**	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"] // ...
console. O JEITO CERTO **CODIGO DAS FUNÇÕES**	**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.**	Todo array começa com o índice 0, portanto para acessarmos o seu primeiro valor utilizamos nome_do_array[0]

console. **CODIGO DAS FUNÇÕES**		

	Para imprimir o texto "Rio de Janeiro", valor da variável na posição 0 do array, usamos o **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[0]);
		
		var estados = ["Rio de Janeiro", "São Paulo", "Bahia"];
		console.log(estados[1]);

Alterando um valor no Array

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;**
Alterando um valor de uma posição	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";**
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;** 

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

ARRAY?		**Código 10**. Array com dois telefones **var telefones = ['(11) 98899 - 8787', '(22) 3455 - 8819', '(91) 95620 - 0000'];**

Outros exemplos de ARRAY	
	Armazenar os meses do ano para uma consulta
	`var meses= [ 'janeiro', 'fevereiro', 'março', 'abril', 'maio', 'junho', 'julho', 'agosto', 'setembro', 'outubro', 'novembro', 'dezembro' ];`
		
		Armazenar as redes sociais que um usuário possui
		`var redesSociais= [ 'Facebook', 'Twitter', 'Linkedin', 'Instagram' ];`
		Armazenar os itens de um menu
		`var menu = [ 'Todas', 'Videos', 'Noticias', 'Maps', ];`

Tipos de dados undefined

Undefined

UNDEFINED	**Variável criada sem receber um valor** Quando for esse o caso, o JavaScript dará a variável o valor undefined.	**var nome; var nome; console.log(nome); //vai imprimir undefined**
TIPO STRING usando o valor undefined	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);**	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**
RESOLVENDO O ERRO	Uma das formas de **resolver esse erro é inicializando a variável** (**Código 5**). **var nome = '';**	
OPERAÇÃO MATEMATICA usando o valor undefined	**var idade;** **console.log(idade + 1);**	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;**

Tipos de dados null

Null

Null	É possível iniciar uma variável com null, o que **significa que queremos adiar intencionalmente ou não atribuir um valor a ela**, como mostra o **Código 1**. `var nome = null;` **Código 1**. Atribuindo o valor null a uma variável

Null	Se imprimirmos esta variável teremos o valor null (**Código 2**).
	
	`var nome  = null;`
	`console.log(nome);`
	`// vai imprimir null`
	Código 2. Imprimindo uma variável null
	
	Caso você ainda não saiba qual deve ser o valor de uma variável, utilize null em sua inicialização para comunicar essa incerteza.

Manipulando uma variável Null	**Devemos ter cuidado** ao lidar com uma variável cujo valor é null.
	Por exemplo, se presumirmos o tipo de uma variável, tentando acessar algum método ou atributo dela enquanto o seu valor for null, um erro como o do **Código 3** ocorrerá:
	
	`var nome = null;`
	`console.log(nome.length);`
	
	TypeError: Cannot read property 'length' of null
	
	Código 3. Erro gerado por tentar acessar a propriedade length de uma variável null
	Nesse caso **o erro é tentar acessar a propriedade length de null, assumindo que nome é uma string**.

Outro exemplo Null	`var preço = null;`
	`console.log(preco.toFixed(2));`
	
	TypeError: Cannot read property 'toFixed' of null
	Código 4. Erro gerado por tentar acessar a propriedade toFixed de uma variável null.

2.0 Declarando uma Constante

Protegendo Valores	como proteger valores que nunca deveriam mudar utilizando a palavra-chave const.	
Valores que nunca devem mudar	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/";
const **Dando um Valor**	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***
const **Dando um Valor**	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.***

const
Vantagens
	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/registos";
		
		// código
		// código
		// e mais código
		
		console.log(url);
		Código 5. Imprimindo uma variável que foi alterada

const	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

Vantagens		url = “https://api.com.br/registos”; // 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.

3.0 - Operadores

Conceito inicial

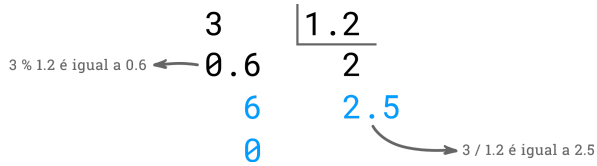
Definição de operadores **Tipos de operadores**	Os operadores são **símbolos usados para modificar** ou até mesmo **gerar um novo valor**. ☒ 1- Aritméticos ☒ 2- Incremento e decremento ☒ 3-Comparação, ou relacionais ☒ 4-Lógicos ☐ 5- Atribuição	var atendeAClassificacaoEtária = idade >= 18

Tipos de operadores

1- Aritméticos

são eles :	exemplos	

o **adição** (+), **subtração** (-), **multiplicação** (*), **divisão** (/) e **módulo** (%)	**var** quantidadeVagas = 2 **+** 5; // resultado: 7
	
	var contratados = 7 **-** 2; // resultado: 5
	
	var valorContribuicao = 2500 ***** 0.10; // resultado: 250
	
	var primeiraParcela = 2500 **/** 2; // resultado: 1250	

Operador módulo (%)	Esse operador **retorna o resto de uma divisão**.
	Exemplos:
	var resto1 = 15 % 4; // resultado: 3
	var resto2 = 3 % 1.2; // resultado: 0.6
	 
	
	O resultado é 0.6 porque na última divisão entre inteiros restou o valor 0.6, então fique ligado!
		Este operador pode ser **usado para verificar se um número é par.** Para fazer isso, basta verificar se o número dividido por 2 gera um resultado inteiro e resto zero (0).
		
		Veja no código abaixo:
		var verificaSeEPar = 20 % 2 == 0;

	Além de ser usado para somar número, **o operador de adição (+) também pode juntar**	Quando a gente **usa o operador de adição com tipos diferentes**, digamos

Operador de adição (+)

strings.

```
console.log("texto" + " e complemento");
// Vai imprimir "texto e complemento"

console.log("5" + "6");
// Vai imprimir "56"
```

uma string e um number, o operador de**adição primeiro converte todos os valores para string e depois as une.**

```
console.log("5" + "6");// Vai imprimir "56"
console.log("texto" + 20); // "texto20"
console.log("texto" + true); // "textotrue"
console.log("texto" + null); // "textonull"
console.log("texto" + undefined); // "textoundefined"
console.log( [1,2,3] + 4 ) // "1,2,34"
console.log( {nome:'José'} + 1 ); // "[object Object]1"
```

(+) Conversão para string

Conversão para string

Valor original	Valor em string
true	"true"
false	"false"
null	"null"
undefined	"undefined"
array	"valor1, valor2, valor3"
objeto	"[object Object]"

Outros exemplos de conversão de tipos:

```
console.log(5 + 20); // 25
console.log(5 + true); // 6
console.log(5 + null); // 5
console.log(true + true); // 2
console.log(true + null); // 1
console.log(null + null); // 0
console.log(null + false); // 0
console.log(5 + undefined); // NaN
console.log(null + undefined); // NaN
```

(+) Conversão para number

Conversão para number

Valor original	Valor em number
true	1
false	0
null	0
undefined	NaN (não é número)

```
console.log("5" + "6");//  "56"
console.log("texto" + 20); // "texto20"
console.log("texto" + true); // "textotrue"
console.log(5 + 20); //  25
console.log(5 + true); //  6
console.log(5 + undefined); // NaN
```

Operação aritmética e atribuição

Existem situações nas quais precisamos **gerar um novo valor e atribuí-lo a mesma variável**.
Nesses casos podemos fazer de duas formas:

Exemplo 1:

```
var preco = 10;
preco = preco + 20;
```

Código 7 - Somando o valor da própria variável.

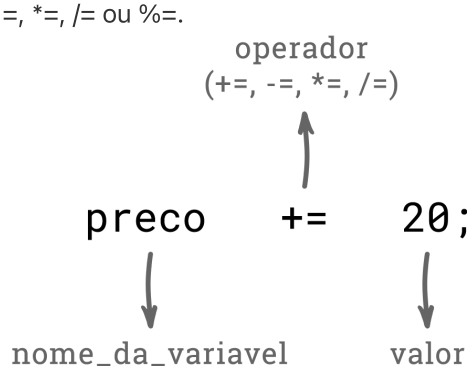
Exemplo 2:

```
var preco = 20;
preco += 1;
```

Código 8 - Utilizando o operador aritmético e de atribuição juntos.

Ambos os exemplos 1 e 2 **geram o mesmo resultado**, atribuindo o valor 30 no primeiro exemplo e 21 no segundo exemplo a variável preco, mas no segundo caso escrevemos menos código.

Podemos usar qualquer operador aritmético em conjunto com o operador de igualdade nas formas +=, -=, *=, /= ou %=.



Em qual ordem o JavaScript executa as operações aritméticas?	Como resolver uma subtração ou adição primeiro que as demais operações?	O que é NaN?
Assim como nas operações matemáticas, os primeiros cálculos feitos **são os que possuem os operadores *, / e % e só então + e -.**	Caso seja necessário mudar essa ordem, **colocamos o que queremos priorizar dentro de parênteses** e ele será efetuado primeiro. **Exemplo: (5 + 9) * 5**	NaN significa "**não é um número**", ele é o resultado de uma **operação matemática que falhou** ou por causa de uma operação com**tipos incompatíveis** (string * string, string / boolean, array * boolean) ou pela divisão 0 / 0.

Tipos de operadores

2- Incremento e decremento

O que são?	Exemplos

Os operadores de incremento ++ e decremento --, são **usados para adicionar ou subtrair o valor 1 de uma variável.**	var contador = 5;
	contador++;
	console.log(contador);// contador agora vale 6
	
	contador--;
	console.log(contador);// contador agora vale 5

Efeitos iguais	Perceba que **contador = contador + 1** ou **contador += 1 ou contador++ tem o mesmo efeito**, em **todos os casos o valor em contador será aumentado em 1.** Assim como contador **-= 1 ou contador = contador - 1 ou contador --**, **também são iguais, porque em todos**

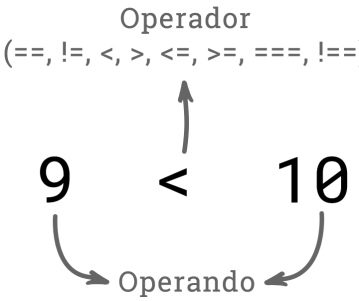
	os casos o valor em contador será diminuído em 1.
Ordem das operações de incremento ou decremento	Esses operadores **podem ser colocados antes ou após o nome de uma variável.**

Colocados os operadores antes	Quando **colocados o operador antes do nome, o valor muda imediatamente**, como mostra o **Código 2**.
	`var numero = 9;`
	`console.log(++numero);`
	Código 2. Utilizando o operador de incremento antes do nome da variável
	Neste caso, **primeiro o valor será alterado de 9 para 10 e depois ele será impresso**.

Colocados os operadores depois	Quando o **operador é colocado após o nome**, **o valor muda após a operação atual ser executada**.
	`var numero = 9;`
	`console.log(numero++);`
	Código 3. Utilizando o operador de incremento após o nome da variável
	Neste caso, **primeiro o valor 9 será impresso e depois ele será modificado para 10**.

Tipos de operadores

3- Comparação, ou relacionais

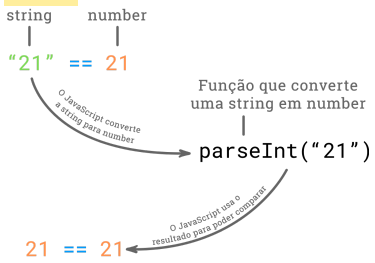
O que são?	Exemplo	
Os operadores de comparação servem para **comparar dois valores**, retornando um booleano (**true ou false**). Os operadores de comparação são: **==, !=, <, >, <=, >=, === e !==.**	 Neste caso **verificamos se o valor 9 é menor que o valor 10.**	==, !=, <, >, <=, >=, ===, !== comparam dois dados e retorna um valor booleano true ou false.

	Após esta verificação, nesse caso, o JavaScript vai nos retornar o valor true.	

Operadores == !=	O operador de igualdade **== compara dois valores e retorna true se eles forem iguais.**
	`console.log( 21 == 21 ); // vai imprimir true`
	`console.log( 120 == 100); // vai imprimir false`
	Código 1 - Comparando dois valores do tipo number utilizando o operador **==**.
		Quando precisamos **verificar se um valor é diferente do outro**, utilizamos o **operador de desigualdade !=**.
		`console.log( 11 != 21 ); // vai imprimir true`
		`console.log( 100 != 100); // vai imprimir false`
		Código 2 - Comparando dois valores do tipo number utilizando o operador **!=**.
		
		Esses operadores conseguem fazer a comparação, mesmo que os valores sejam de tipos diferentes.
		`console.log( "20" == 20 ); // true`
		`console.log( true == 1 ) ; // true`
		Código 3 - Comparando dois valores de tipos diferentes utilizando o operador ==.
		Nesse contexto do operador ==, a **string** "20" é igual ao number 20 e o booleano true é igual ao number 1.

Operadores de igualdade e desigualdade estrita	O operador de **igualdade estrita (===) assim como o operador de igualdade (==), vai comparar se dois valores são iguais.**
	A diferença entre os dois é que o operador === não faz conversão de tipo, ou seja, ele vai comparar se os dois valores são iguais tanto em valor quanto em tipo.
		O mesmo vale para o operador de desigualdade estrita **(!==)**.
		`"21" !== 21 // o resultado será true`
		`1 !== true // o resultado será true`
		Código 5 - Comparando dois valores de tipos diferentes utilizando o operador !==

	Por exemplo:
	"21" === 21 // o resultado será **false**
	1 === true // o resultado será **false**
	Código 4 - Comparando dois valores de tipos diferentes utilizando o operador ===.	

Conversão de tipo	Ao utilizar o operador de igualdade o **JavaScript vai verificar se os dois valores possuem o mesmo tipo. Se não for esse o caso, o JavaScript vai converter os valores da seguinte forma** >>>
	 
	
		• **null** e **undefined são iguais**:
		null == undefined // o resultado será true
		null != undefined // o resultado será false
		Código 6 - null é igual a undefined.
		• **true é convertido para 1 e false para 0**
		true == 1 // o resultado será true
		true != 1 // o resultado será false
		false == 0 // o resultado será true
		false != 0 // o resultado será false
		Código 7 - true é igual a 1 e false igual a 0.
		Mesmo que usar os operadores == e != pareç a ser mais fácil, podemos ter alguns problemas em nossas validações, já que para eles não importa se os tipos são diferentes. O mesmo não vai ocorrer para os operadores === e !== já que com eles conseguimos ter precisão na nossa comparação.

Conhecendo os	**Menor que (<) - verifica se um valor é menor que o outro** 13<100 //true 34<5 //false	**Menor ou igual a (<=) - verifica se um valor é menor ou igual ao outro** 13<=100 //true 13<=13 //true

operadores <, >, <=, >=	Maior que (>) - verifica se um valor é maior que o outro 20>10 //true 20>50 //false	34<=5 //false 34<=34 //true Maior ou igual a (>=) - verifica se um valor é maior ou igual ao outro 48>=48 //true 30>=31 //false

Só pra lembrar!

=	==	===
significa que uma variável recebe ou é atribuída de um valor.	verifica se uma condição é igual a outra, fazendo conversão de tipo para isso.	verifica se uma condição é exatamente igual a outra tanto em tipo quanto em valor.

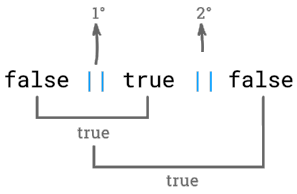
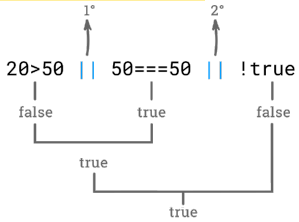
Tipos de operadores

4- Lógicos

O que são?	Exemplos	

Os operadores lógicos comparam dois valores booleanos e retorna um valor true ou false. São eles:
 	
&&, || e !
	Veja alguns exemplos:
	true && false // false
	false || false // false
	!true // false
	
	Código 2 - Exemplos com os operadores lógicos.
		É comum usarmos estes operadores lógicos, para combinar operações relacionais como as que vemos a seguir:
		1 == '1' || "a" == "a" // true
		35 === '35' && 1 == 10 // false
		!('a' == 1) // true
		
		Código 3 - Operadores lógicos combinando operações lógicas.

	A ordem que o JavaScript vai interpretar uma operação lógica será da esquerda para direita.	Caso haja uma operação relacional, ela será resolvida e o seu valor

A ordem dos fatos		**será usado pelo operador lógico.** 
E lógico (&&) E	Através do **operador E(&&) é possível comparar dois valores e retorna true apenas se os dois valores comparados forem true.**	true && true // true true && false // false false && false // false **Código 4 -** Exemplo com o operador &&.
Ou lógico (||)	Ao **comparar dois valores usando o operador OU (||), o resultado será true se pelo menos um dos dois valores for true.** true || false // true false || true // true false|| false // false **Código 5 -** Exemplo com o operador ||.	Um ponto importante desse operador é que **caso o primeiro valor seja true, ele simplesmente vai ignorar tudo que vier depois dele.** Veja um exemplo: true || false &&false ||false &&false || false && false&&false ||false || false &&false || false && false **Código 6 -** Outro exemplo com o operador ||. O resultado **true** já foi definido neste ponto aqui: true || **Código 7 -** Simplificando o Código 6. **Tudo que vier depois não importa.**
Não lógico (!)	O operador **lógico NÃO (!) vai mudar o valor posterior a ele, ou seja, true será false e false será true.** !false // true !true // false	Com este operador também **é possível negar o resultado de uma expressão inteira, para isso colocamos a expressão que queremos**

	Código 8 - Exemplo com o operador !.	**negar dentro de parênteses**:
		!(true || false) // false
		!(true || true && false) // false
		Código 9 - Negando uma expressão completa.

Tipos de operadores

5- Atribuição

O que são?	Exemplos

4.0 - Estruturas Condicionais ? ou :

1- Introdução

5- IF ternário ?: / ?:

? ou :

=?

= ?

= :

=:

: ou ?

: se sim

? senão

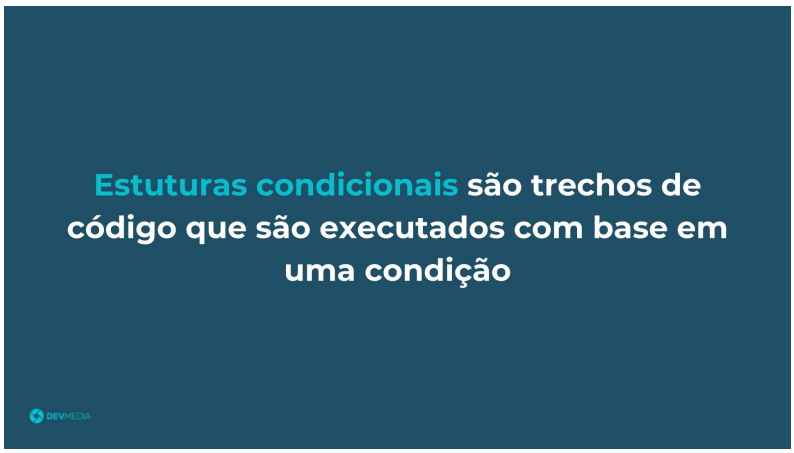
var mensagem = (mes == 11 || mes == 12) ? "Promoção" : "Preço normal";

1- Introdução

em js tô na parte de estruturas condicionais com if ternário com múltiplas condições

Neste curso você vai aprender a trabalhar com as seguintes estruturas condicionais:

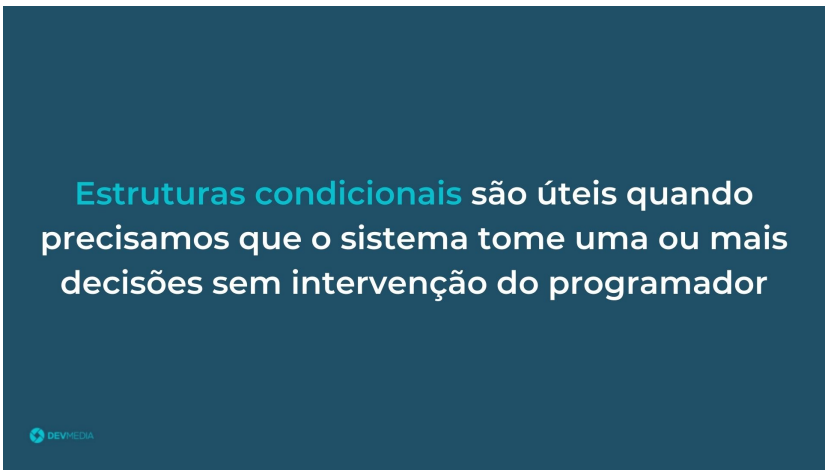
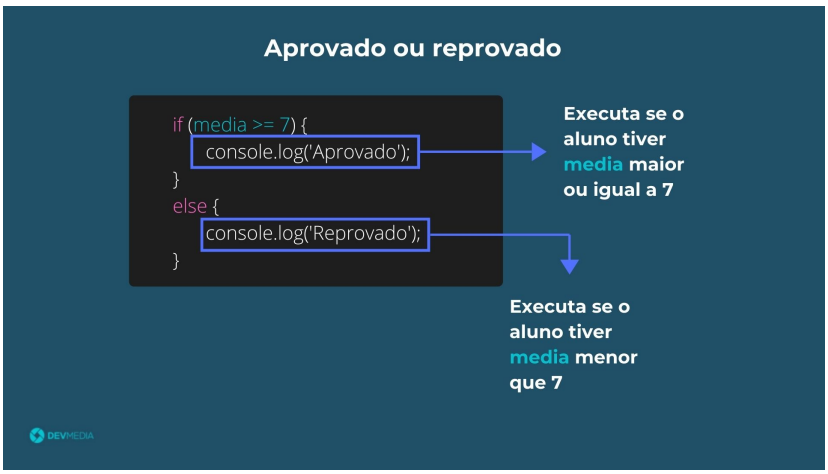
- if / else;
- else if;
- switch

O que são estruturas condicionais?	

Exemplo estruturas	`if (idade >= 16) {`
	`  console.log('Pode votar');`
	`}`

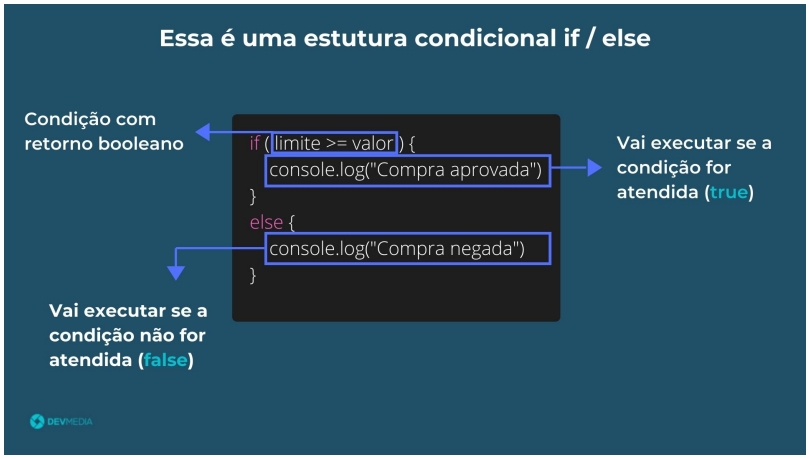
condicionais
if
	else {
	console.log('Não pode votar');
	}

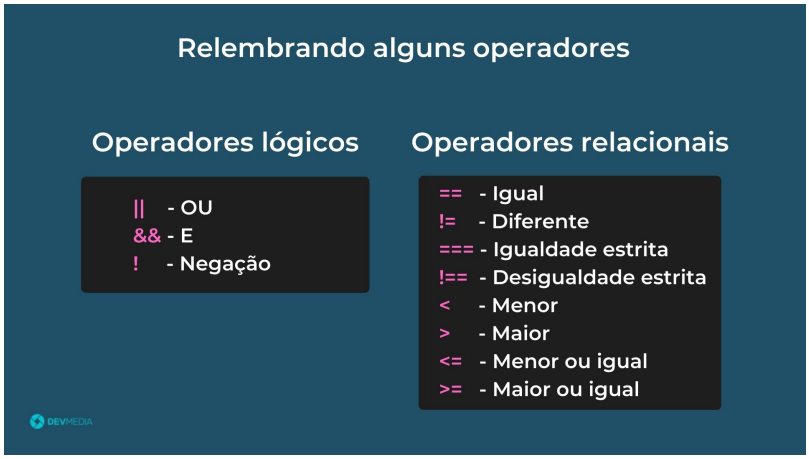
Exemplo estruturas condicionais
switch
	switch (ddd) {
	case 11:
	console.log("São Paulo");
	break;
	case 21:
	console.log("Rio de Janeiro")
	break;
	}

Estruturas Condicionais Por que elas são úteis?	
Exemplo	

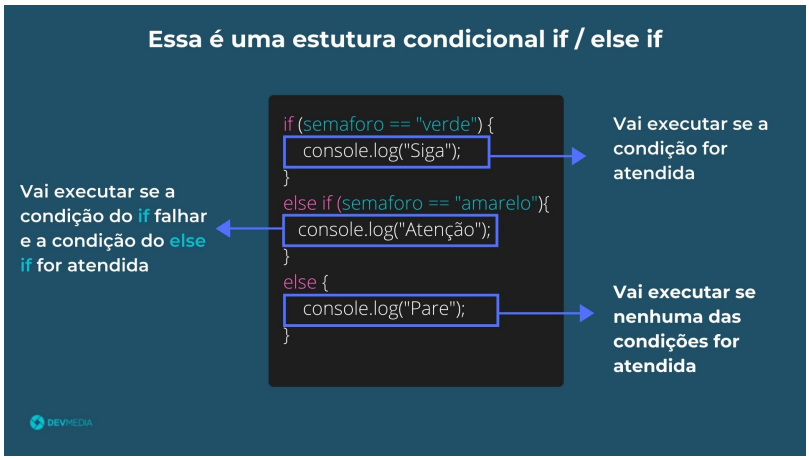
Conceitos

2- IF/ ELSE

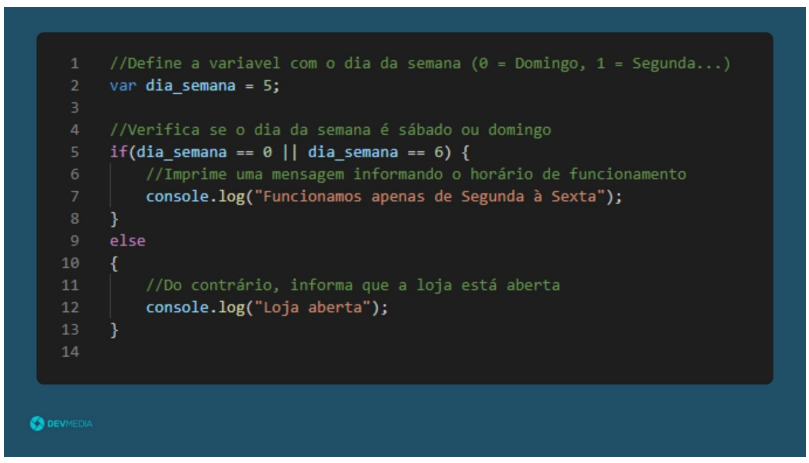
if / else	a estrutura condicional if/elf trabalha com condições booleanas que indicam qual caminho o código deve seguir
Estrutura Condicional if/else	
Exemplos de uso de condições if / else	
Exemplos de uso de condições if / else	
Relembrando alguns operadores	



Segunda Condição else if



Exemplo prático



```
//Define a variavel com o dia da semana (0 = Domingo, 1 = Segunda...)
var dia_semana = 5;

//Verifica se o dia da semana é sábado ou domingo
if(dia_semana == 0 || dia_semana == 6) {
    //Imprime uma mensagem informando o horário de funcionamento
    console.log("Funcionamos apenas de Segunda à Sexta");
}
else
{
    //Do contrário, informa que a loja está aberta
    console.log("Loja aberta");
}
```

3- IF e às anomalidades do tipo booleano

O que é booleano?	Variáveis booleano
Em ciência da computação, **boolean**, ou lógico, é um tipo de dado primitivo que possui dois valores, que podem ser considerados como 0 ou 1, falso ou verdadeiro. Chamado **boolean** em homenagem a George Boole, que definiu um sistema de lógica	**Variáveis booleanas são variáveis** capazes de conter apenas 1 de 2 valores: verdadeiro ou falso. Como as condições no IF retornam verdadeiro ou falso **são** chamadas **booleanas**.

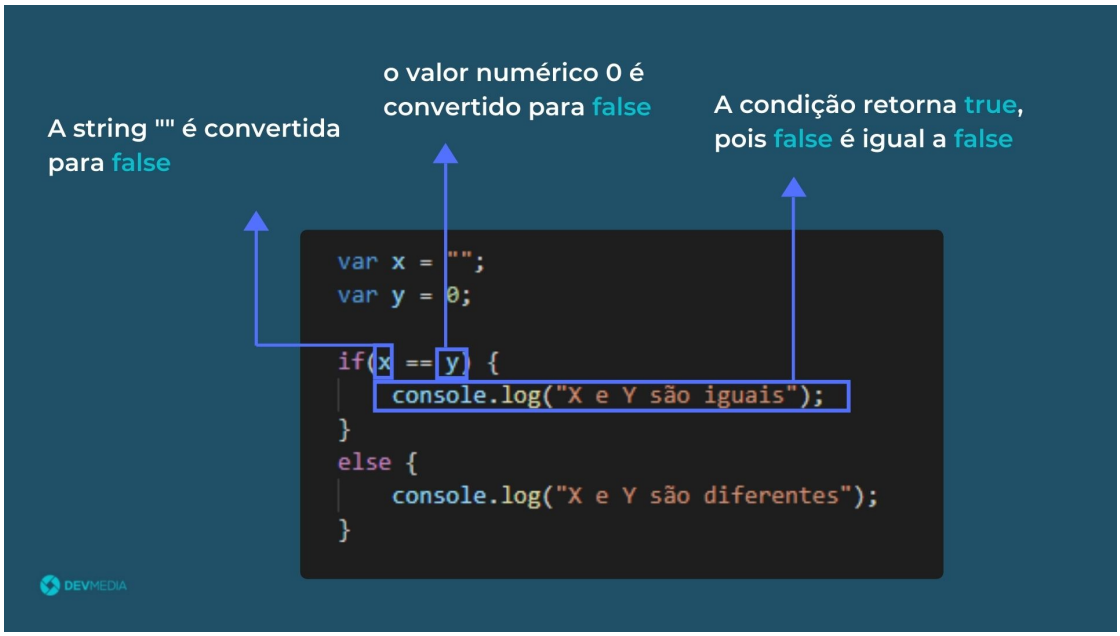
algébrica pela primeira vez na metade do século XIX.	

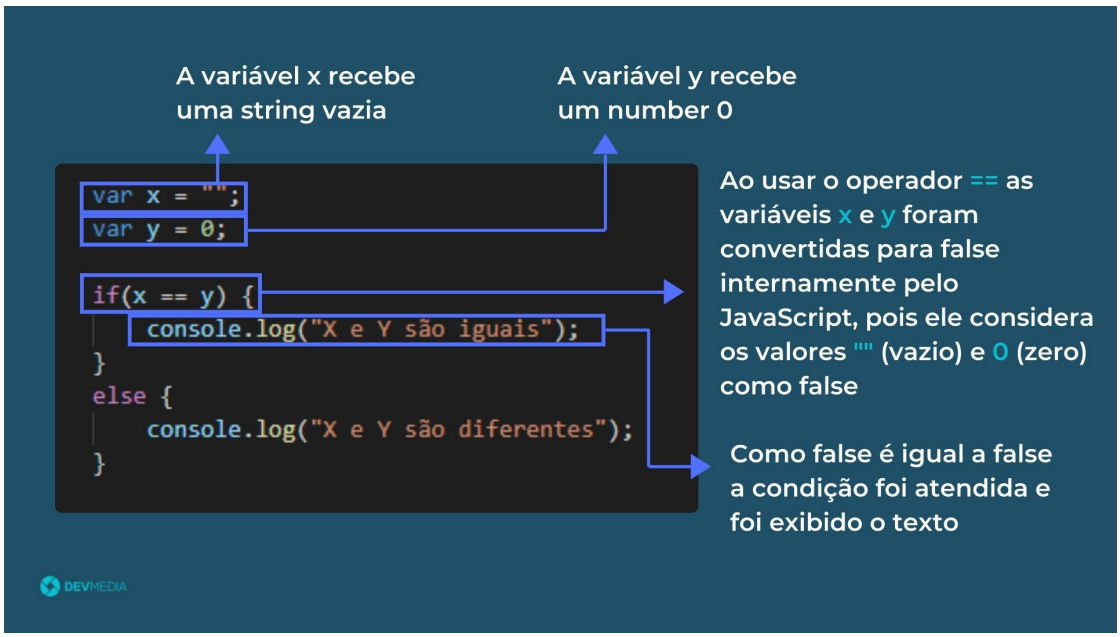
if / else com dados booleano

O JavaScript possui algumas anomalidades com tipo de dado booleano devido a um sistema interno de conversão de tipo

Essa conversão ocorre quando utilizamos o operador ==

EXEMPLO





OBS

Perceba que por causa da conversão de tipo feita pelo operador == a condição retornou true, mesmo comparando valores diferentes

DEVMEDIA

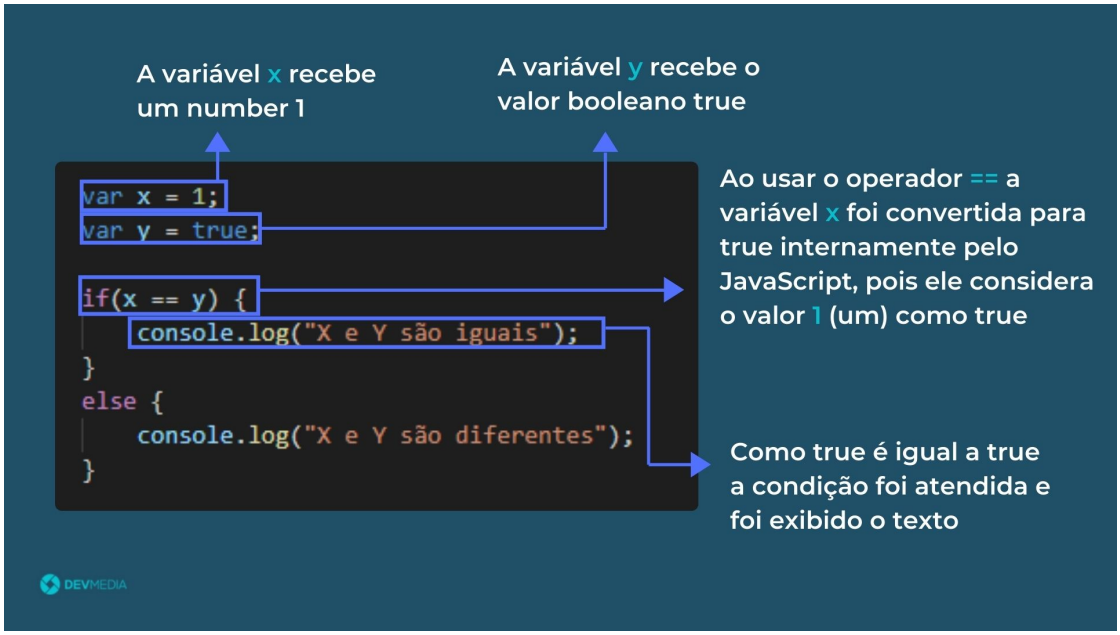
OUTRO EXEMPLO

O valor 1 é convertido para true

A condição retorna true, pois true é igual a true

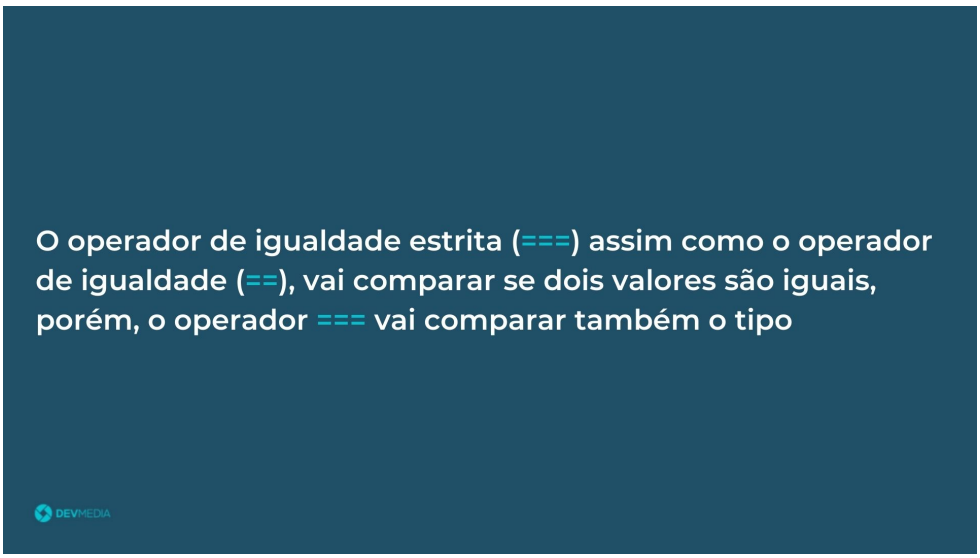
```
var x = 1;
var y = true;

if(x == y) {
    console.log("X e Y são iguais");
}
else {
    console.log("X e Y são diferentes");
}
```

DEVMEDIA

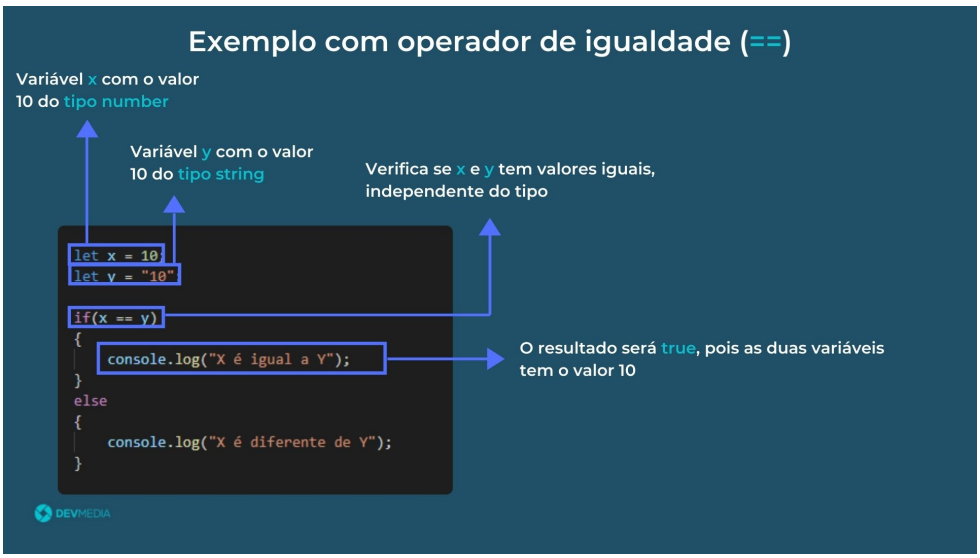


4- IF com igualdade estrita

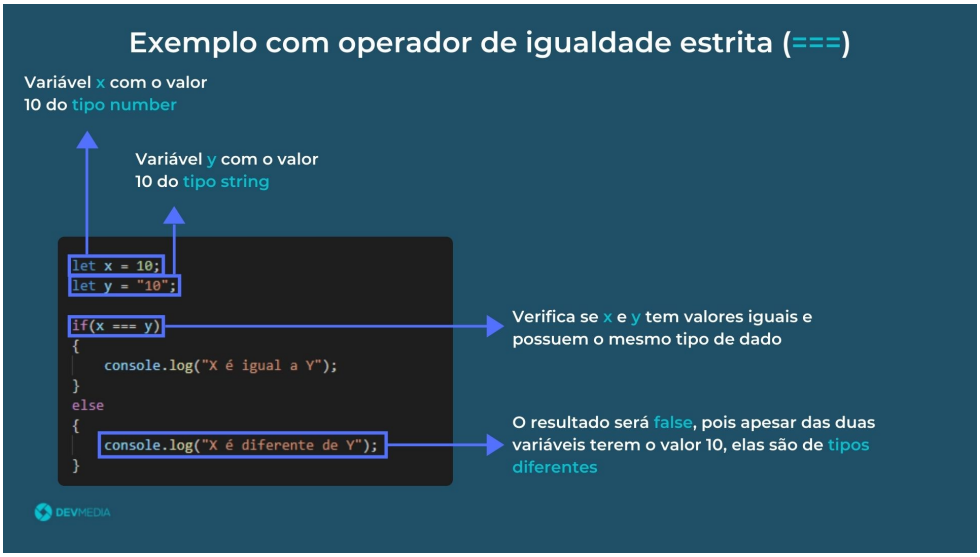
Condições com igualdade estrita



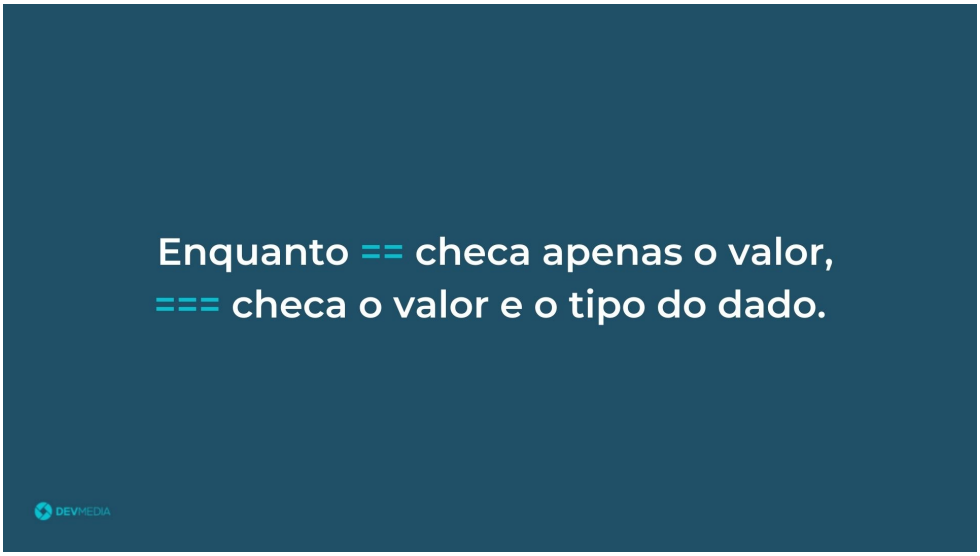
Exemplo com operador de igualdade (==)



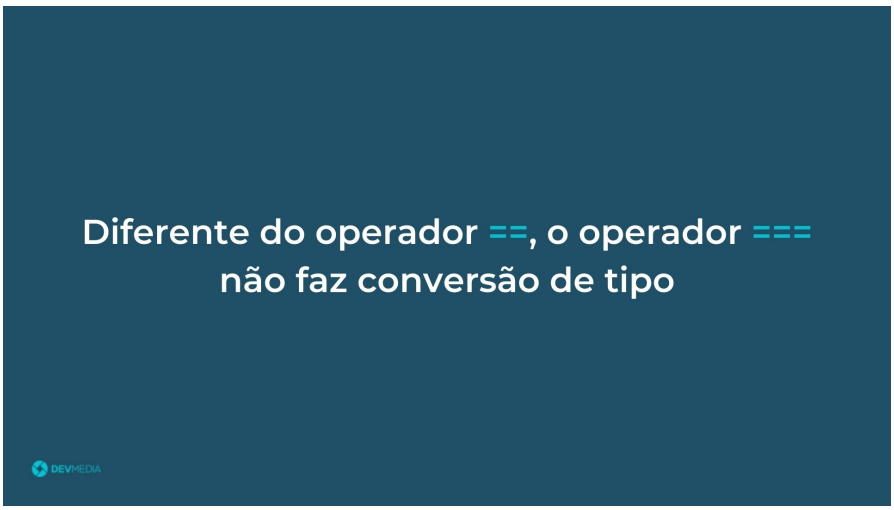
Exemplo com operador de igualdade restrita (===)



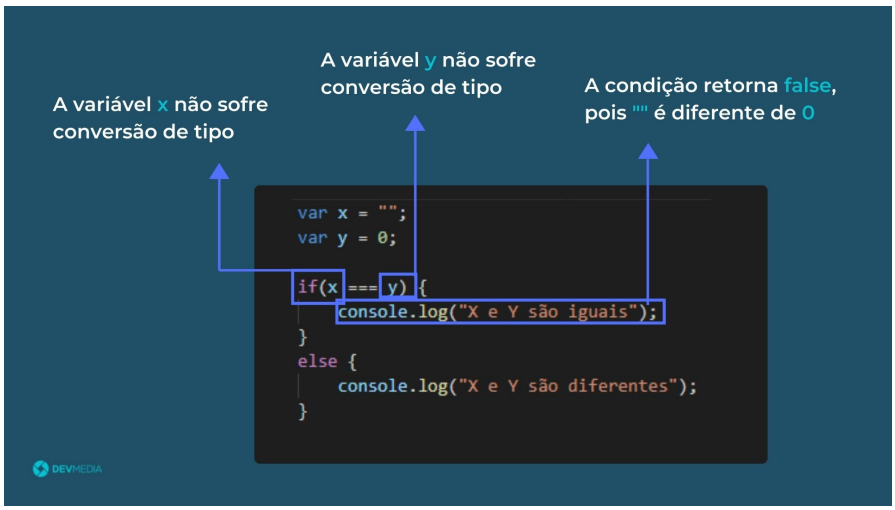
Diferenças (==) para (===)



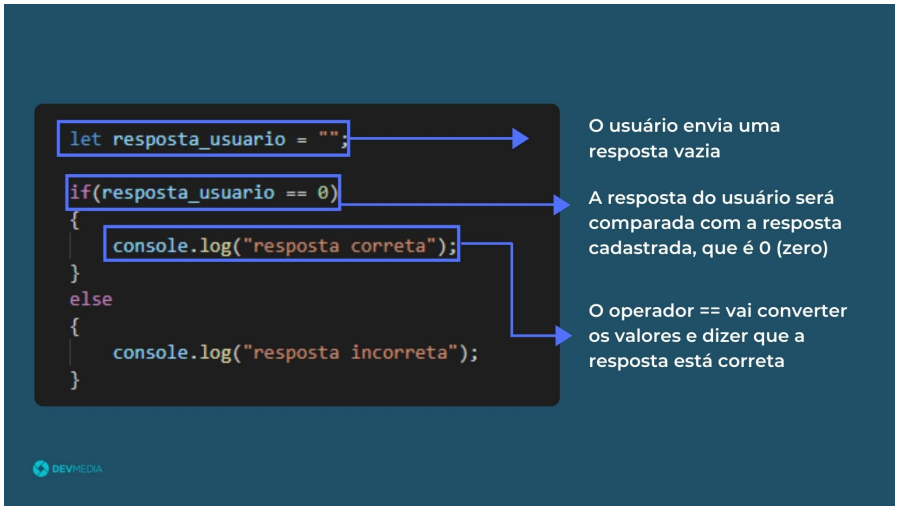
Corrigindo às anomalias com ===



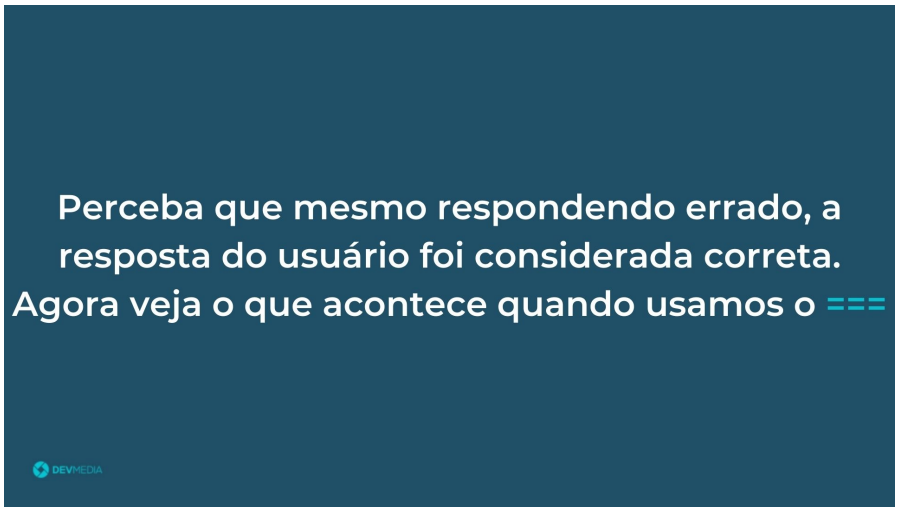
Corrigindo às

anomalias com ===	Por esse motivo, ele pode ser utilizado para resolver a anomalia da aula anterior. Veja o exemplo
Corrigindo às anomalias com ===	
Corrigindo às anomalias com ===	Perceba que agora com o === a condição retornou corretamente, informando que as variáveis eram diferentes
Exemplo Prático do uso da igualdade estrita	

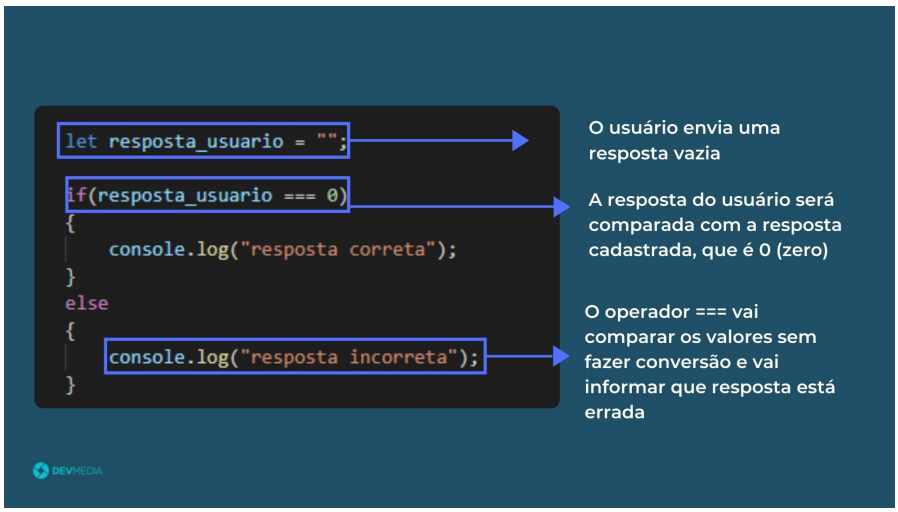
(===)



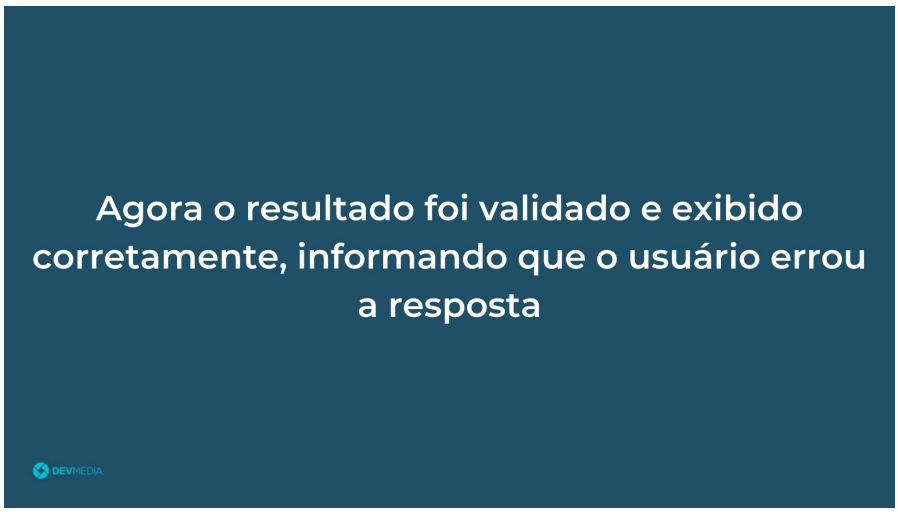
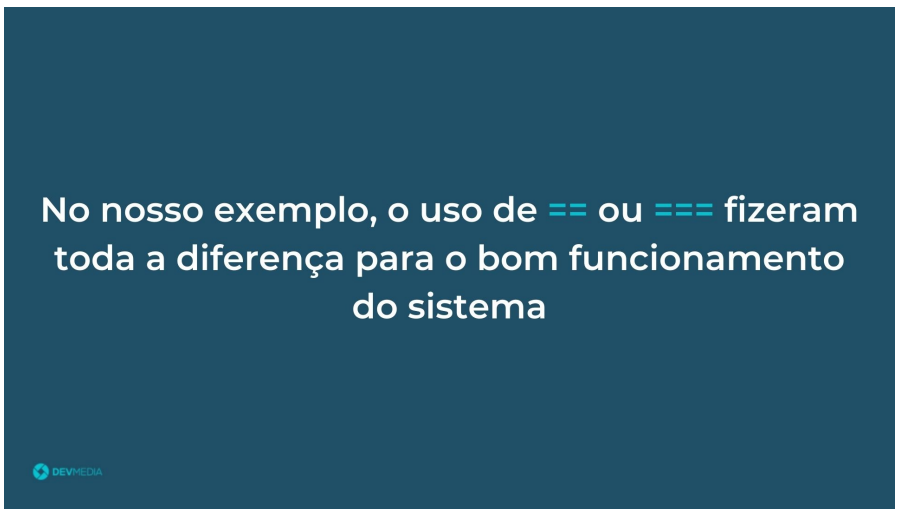
Exemplo Prático do uso da igualdade estrita (===)



Exemplo Prático do uso da igualdade estrita (===)



Exemplo Prático do uso da igualdade estrita (===)

	
Exemplo Prático do uso da igualdade estrita (===)	

5- IF ternário ?: / ?:

? ou :

=?

= ?

= :

=:

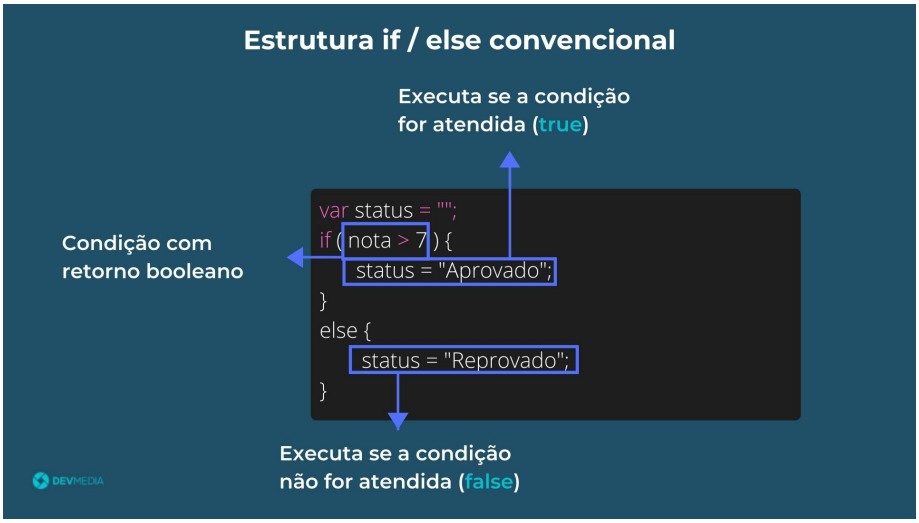
: ou ?

: se sim

? senão

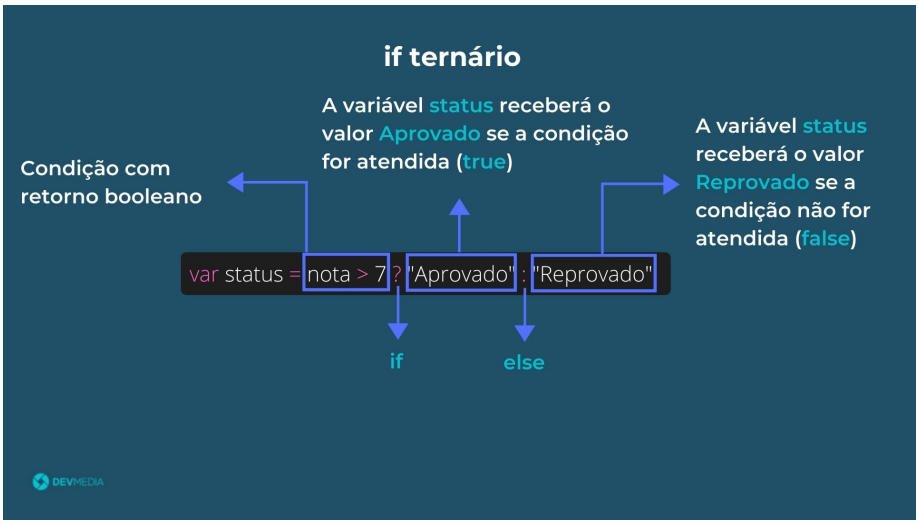
```
var mensagem = ( mes == 11 || mes == 12 ) ? "Promoção" : "Preço normal";
```

Estrutura if / else convencional



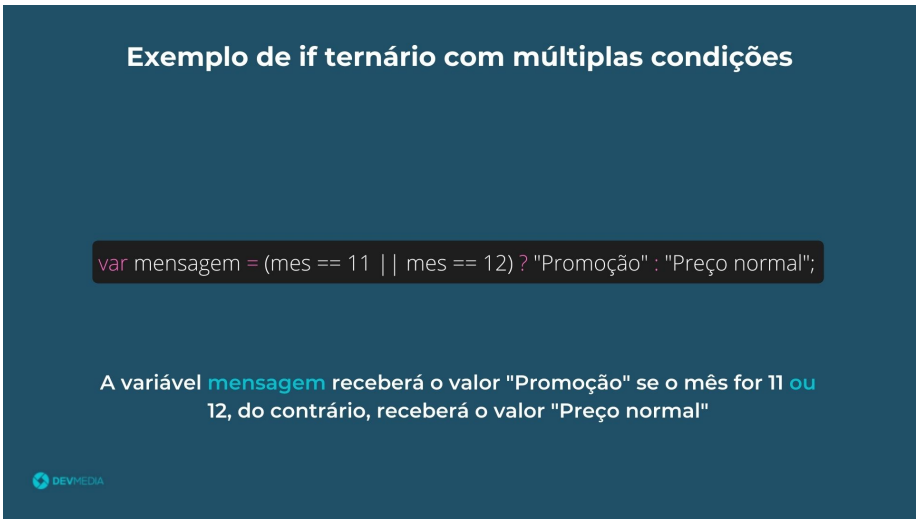
Simplificando a estrutura if / else = if ternário

var statu = nota > 7 ? "Aprovado ": "Reprovado"
? = if
: = else



if ternário também suporta múltiplas condições

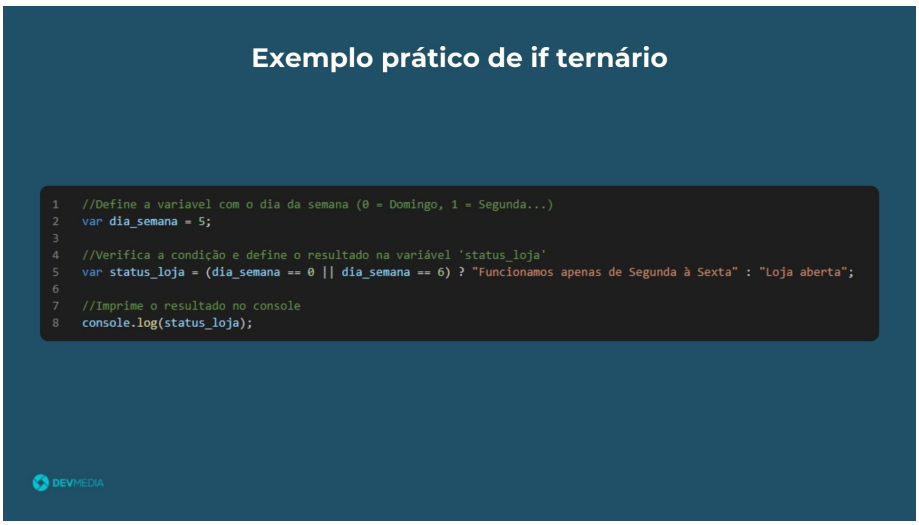
if ternário com múltiplas condições



Exemplo Prático

var mensagem = (mes == 11 || mes == 12) ? "Promoção" : "Preço normal";

if ternário

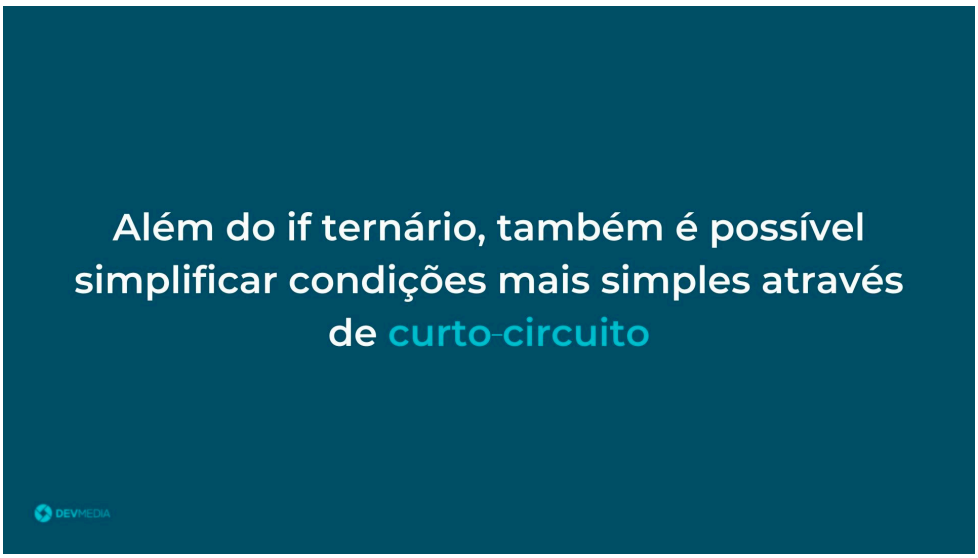


fazer um cadastro - entrevista

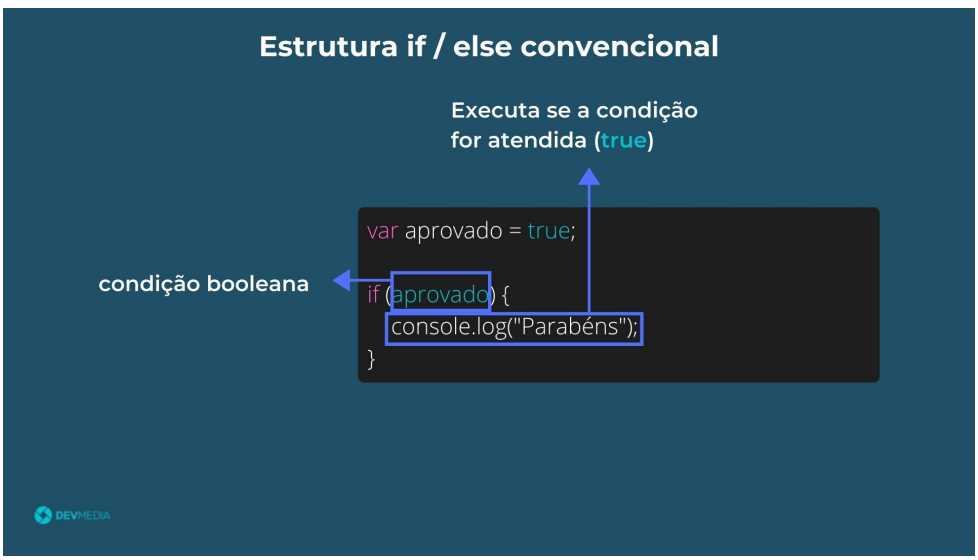
6- Curto-circuito

Utilizar curto-circuito permite ao desenvolvedor implementar condicionais if com muito menos linhas de código

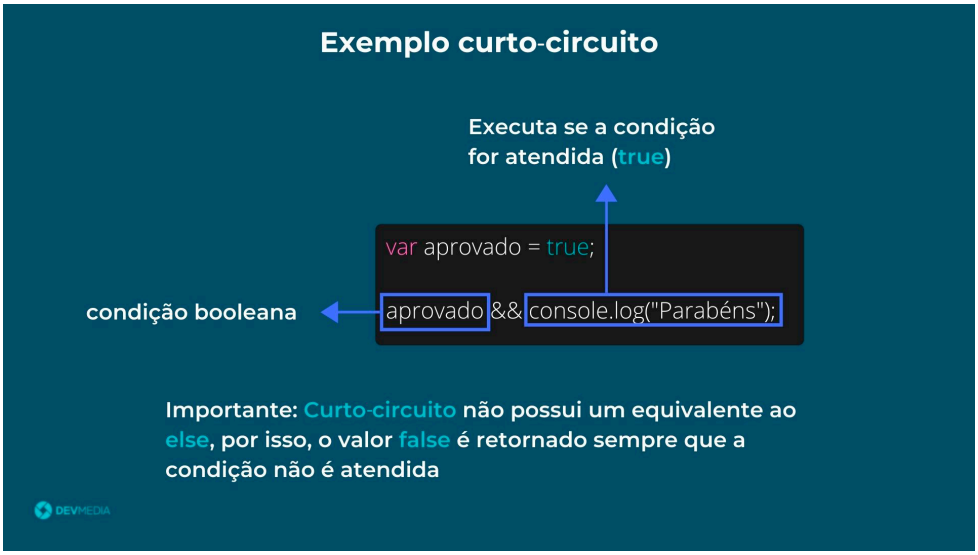
Curto-circuito



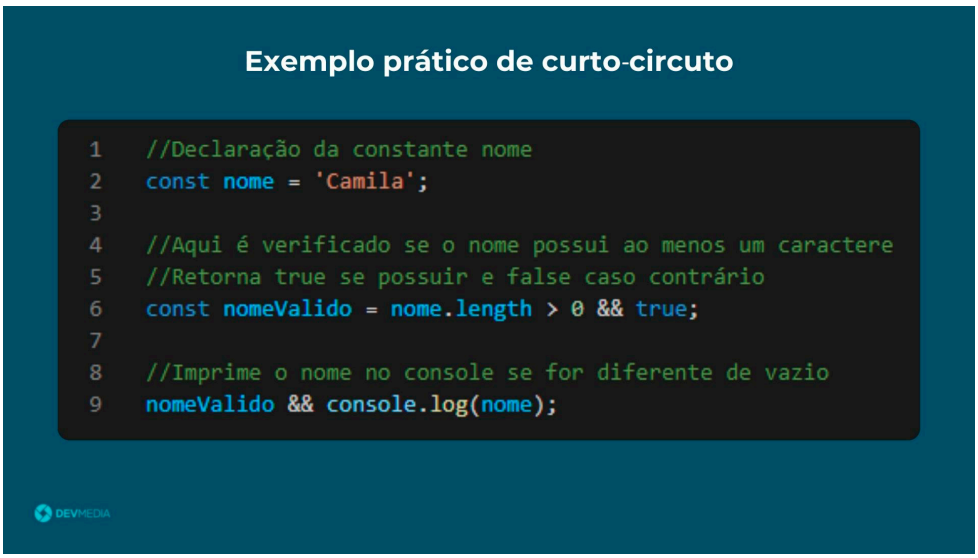
Estrutura if/else convencional



Exemplo curto - circuito



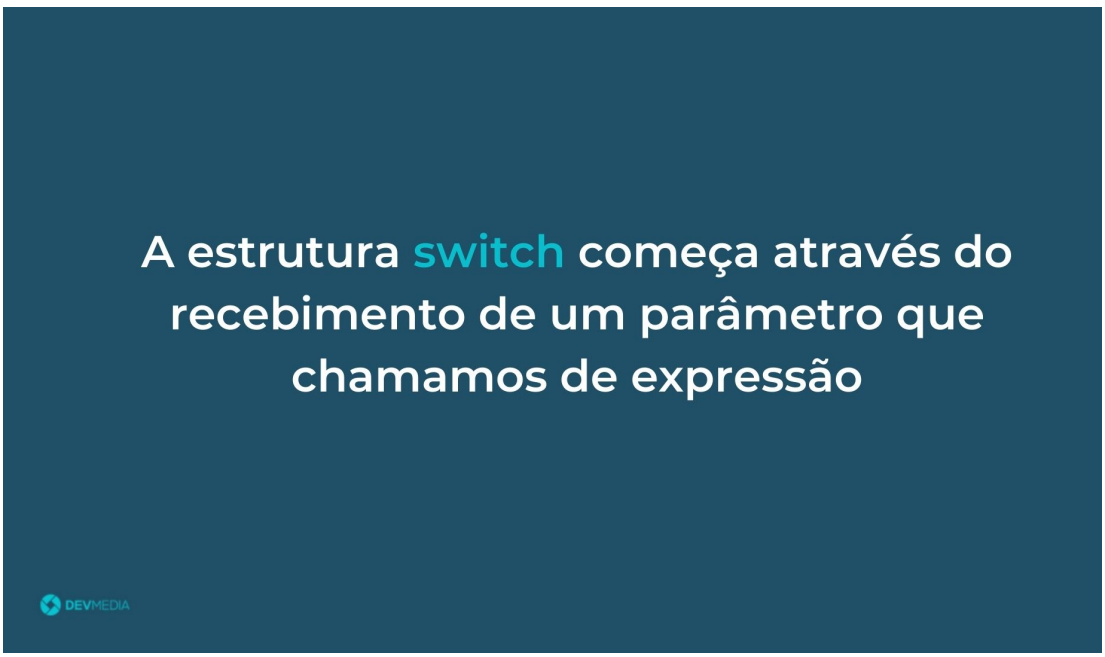
Exemplo Prático



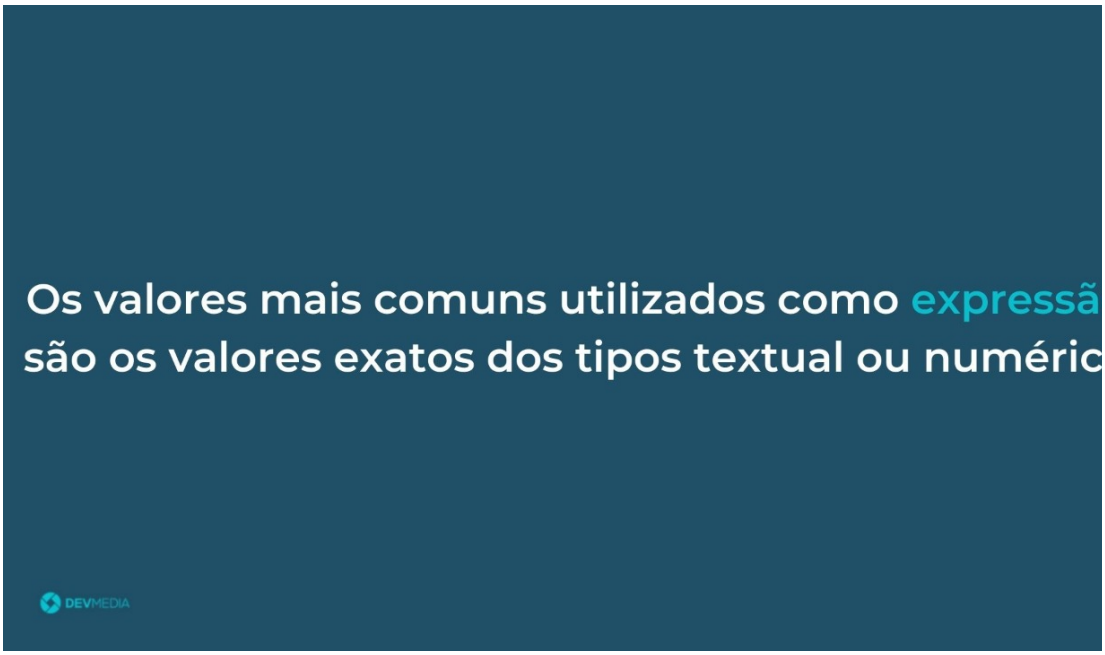
7- Switch

A estrutura condicional switch é um recurso que, assim como o if / else, pode ser utilizado em qualquer tipo de sistema

Switch



Os valores mais usados



Estrutura condicional Switch

Múltiplas clausulas case ao mesmo tempo

Além do formato simples visto, também é possível utilizar múltiplas cláusulas case para um mesmo resultado

DEVMEDIA

Checando múltiplas clausulas case ao mesmo tempo

Checando múltiplas cláusulas case ao mesmo tempo

```
1   var produto = "Smartphone";
2
3   switch(produto)
4   {
5       case "Smartphone":
6       case "Celular":
7       case "Telefone":
8           console.log("Produto: Smartphone");
9           break;
10      case "TV":
11          console.log("Produto: TV");
12          break;
13      default:
14          console.log("Produto inválido");
15          break;
16  }
```

A mensagem "Produto: Smartphone" será impressa se pelo menos uma das três cláusulas case for atendida

DEVMEDIA

Exemplo prático de Switch

Na prática

8- Praticando

corretamente e imprima o valor 'Amazon'?

```
let loja = _____;

_____(loja)
{
    case 1:
        console.log("Americanas");
    _____;
    _____ 2:
        console.log("Amazon");
    break;
}
```

- 2
- switch
- break
- case

Você está em

Guia de linguagem JavaScript » Estruturas condicionais » Exercícios

Complete corretamente o código abaixo corretamente:

```
var status = 1;

 switch  (status)
{
     case  1:
        console.log("Aprovado");
     break ;
     case  0:
        console.log("Reprovado");
```

```
10        break ;
11    }
```

Transforme o código abaixo em um curto-circuito seguindo as orientações da lista:

- O curto-circuito não deve estar associado a nenhuma variável;
- A condição não deve estar entre parênteses.
- Não é necessário declarar a variável status

```
1  var status = true;
2  if(status){
3      console.log("Autorizado");
4  }
```

```
status&&console.log("Autorizado") ;
```

Qual das alternativas abaixo transforma o if ternário em uma estrutura if padrão?

```
1  let resultado = (nota >= 7) ? "Aprovado" : "Reprovado";
```

```
1  if(nota >= 7) {
2      resultado = "Aprovado";
3  }
4  else {
5      resultado = "Reprovado";
6  }
```

Imprimir "menor de idade"

```
let idade = 16;

if ( idade >= 18 ){
  console.log("É maior de idade");
}
else {
  console.log("É menor de idade");
}
```

Compromisso

Qual das alternativas completa o código abaixo para que o resultado seja "Hoje" se a variável compromisso for 0, "Amanhã" se for 1 e "Sem compromisso" se for qualquer valor diferente?

```
var compromisso = 1;

siwtch (compromisso) {
    case 0:
        console.log("Hoje");
        break;
    case 1:
        console.log("Amanhã");
        break;
    defalt
        console.log("Sem compromisso");
}
```

Completando um if

Complete o código abaixo para que a estrutura if fique correta:

```
let  sinal   = "verde";

 if     (sinal == "  verde   ") {
    console.log("Siga");
}
 else if   ( sinal    == "amarelo") {
    console.log("Atenção");
}
 else   {
    console.log("Pare");
}
```

Completando o switch

Complete o switch corretamente:

```
1  var status = 0;
2
3   switch   (status)
4  {
5       case   1:
6          console.log("Cadastrado");
7       break  ;
8       case   0:
9          console.log("Excluído");
10      break  ;
11 }
```

CONTACT

Brunna Croches

Developer Full Stack

brunnacroches.dev

linkedin.com/brunnacroches

github.com/brunnacroches

@brunnacroches.dev

discord.com/brunnacroches

brunnacroches@gmail.com

let's share