JAVASCRIPT - MÓDULO 1 - AULA 2

1. FORMATAÇÃO DE DADOS

Já vimos alguns parâmetros de formatação de dados, por exemplo o **tofixed** (escolher quantas casas decimais deseja ter em um número) e o **tolocalestring** (para converter valores numéricos comuns em valores de moeda). Vamos conhecendo mais deles a cada aula.

```
> var n = 200
undefined
> n
200
> typeof n
'number'
> n = "Google"
'Google'
> typeof n
'string'
> typeof 6
'number'
> typeof 6.5
'number'
> typeof "6.5"
'string'
> typeof []
'object'
> typeof {}
'object'
> typeof function(){}
'function'
> typeof undefined
'undefined'
> typeof NaN
'number'
> typeof Infinity
'number'
> typeof null
'object'
>
```

Mas antes de falarmos sobre formatadores de dados, vamos conhecer um comando muito interessante, que nos diz qual o **TIPO** do dado que estamos trabalhando.

Esse comando é o **typeof.** Observe a imagem ao lado:

Nesta imagem podemos observar que basta digitar o comando seguido de uma variável, texto, numeral e até comandos (como o caso da função).

Com isso temos esse comando como um auxiliar para validação dos tipos de dados e utilizá-lo pode ser de grande valia quando recebemos um dado que por vezes não temos certeza do que se trata.

Agora que conhecemos esse comando tão importante, podemos continuar conhecendo mais formatadores de dados.

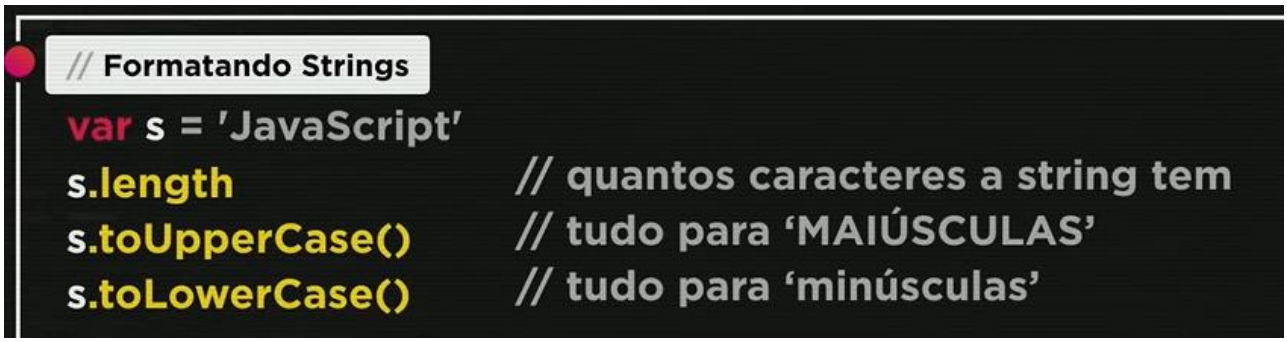
1.1. Valores numéricos: Vamos validar mais possibilidades para formatadores de valores numéricos.

- **replace:** O replace é um comando utilizado para substituir valores. É claro que pode ser utilizado para vários casos, mas vamos observar ele em conjunto com o **tofixed.**

```
> var n1 = 1545.5
undefined
> n1
1545.5
> n1.toFixed(2)
'1545.50'
> n1
1545.5
> n1.toFixed(2)
'1545.50'
> n1.toFixed(2).replace('.' , ',')
'1545,50'
>
```

Como pode observar, ele tem por função substituir um valor por outro, nesse caso ele substituiu o “.” pela “,” pois nas expressões de valores é utilizado vírgula para separar os centavos por exemplo, e não o ponto. (que é por padrão usada nas linguagens de programação.

1.2. **Valores string:** Na formatação de strings viemos com 3 importantes comandos que vão auxiliar bastante no desenvolvimento de textos.



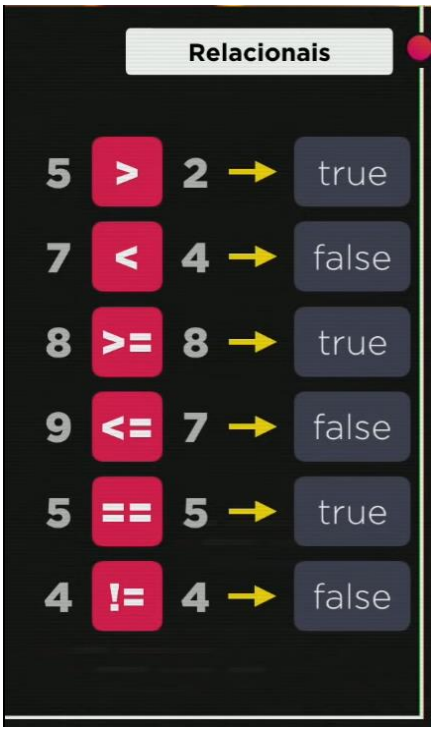
Como pode observar temos 3 comandos muito importantes. O primeiro **.length** conta a quantidade de caracteres que tem dentro de uma variável **STRING** (lembrando que espaços em branco, caracteres especiais e até números também são contados). O segundo **.toUpperCase** faz com que a variável escolhida tenha todo o seu conteúdo convertido para caixa alta. E por fim o terceiro **.toLowerCase** que faz justamente o inverso, converte os valores para caixa baixa.

2. OPERADORES RELACIONAIS

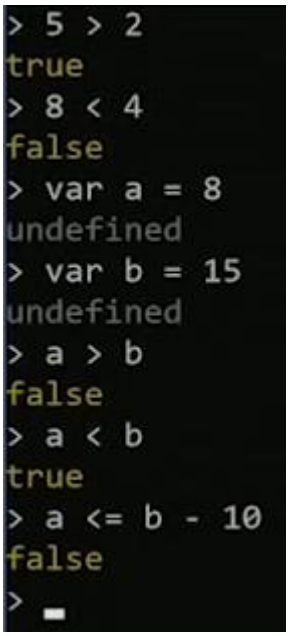
Da mesma maneira que utilizamos operadores relacionais em outras linguagens é feito também no JavaScript. Inclusive são idênticas a maioria das linguagens de programação, inclusive da linguagem Python. São eles então:

==	Igual
!=	Diferente
<	Menor
>	Maior
<=	Menor Igual
>=	Maior Igual

Na prática esses operadores são utilizados para confirmar a veracidade de uma variável em relação a outra, ou seja eu preciso validar e o valor um é alguma coisa em relação ao valor 2. Veja os exemplos a seguir:

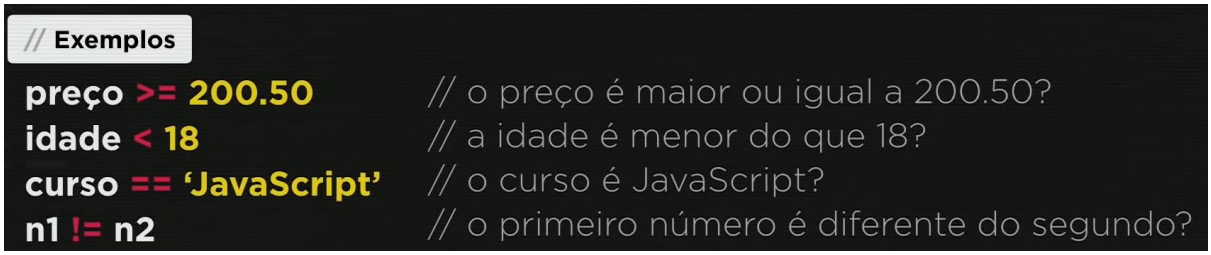


Como podemos observar no exemplo acima, o primeiro valor (a esquerda) é posto em condição de validação com o segundo valor (a direita) e isso resulta na confirmação se aquela comparação é verdadeira ou falsa (variáveis booleanas).

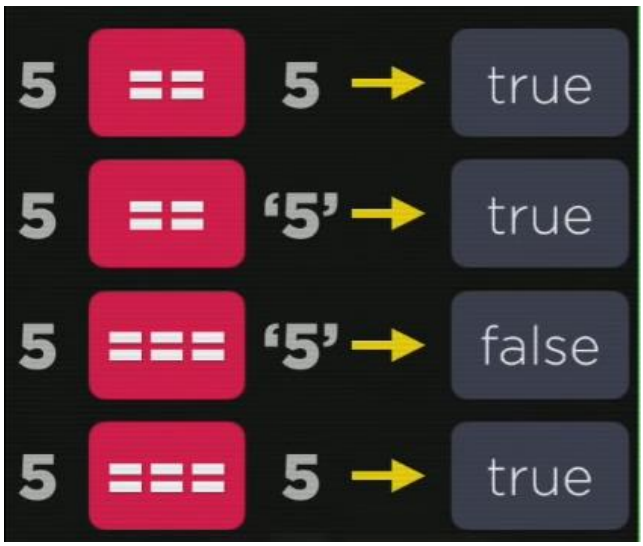


Observe também esses exemplos no Node.js. Vemos na imagem em questão que podemos fazer exemplos mais **COMPLEXOS.** Até uma certa linha comparamos dois valores, depois se observar na penúltima linha consta a seguinte comparação: a <= b - 10. Nesta expressão observamos que queremos comparar se **a** é **menor ou igual** a **b** mas não só isso, antes de executar essa comparação, observe que consta uma operação para b executar. Seria b - 10, b valendo 15 seria 15 - 10. Ou seja a (que vale 8) é menor ou igual a b (que agora vale 5) ? E portanto temos uma resposta diferente da sentença anterior.

Exemplos do dia a dia:



Indo ainda mais fundo nos operadores relacionais, observe a imagem abaixo:



Pode parecer um pouco confuso no início, mas observe inicialmente as duas primeiras sentenças. Temos a **comparação** do valor **cinco** sendo **igual** a ele mesmo. Porém existe uma diferença, na primeira sentença temos dois valores **numéricos**

inteiros, já na segunda, temos a comparação de um **valor inteiro** comparando a um **valor string.** Mas como isso é possível? E eu lhe digo, o operador de igualdade, no JavaScript, vai validar se um valor possui a mesma grandeza do outro, isto é, não importando portanto o tipo desse valor.

Agora observe as últimas duas sentenças. Foi adicionado ao operador de igualdade mais um igual, dessa maneira ele se transforma em outro operador, que chamamos de **IDENTIDADE (ou operador de igualdade restrita)**. Esse operador é muito semelhante ao igual, com a pequena diferença que ele compara também o TIPO do valor que necessitamos de validação, trazendo a resposta se o valor é igual ao outro tanto na grandeza quanto no tipo de variável.

```
> 5 == '5'
true
> var x = 5
undefined
> var y = '5'
undefined
> typeof x
'number'
> typeof y
'string'
> x == y
true
> x === y
false
> x != y
false
> x !== y
true
>
```

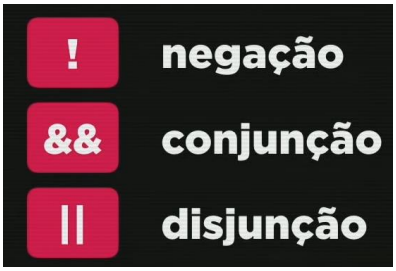
Observe nos exemplos da imagem ao lado. No começo comparamos o 5 novamente, apenas com o operador igual. Depois definimos duas variáveis (inteira e string) e validamos os tipos dela. Logo mais abaixo comparamos elas tanto com o operador igual, quanto o de identidade, e perceba a diferença nas respostas.

Além disso mais abaixo também comparamos se ele é diferente, e logo após adicionamos **MAIS UM IGUAL** ao operador diferente, portanto criamos mais um operador relacional, que vai o mesmo processo do identidade só que validando o oposto, ou seja, se é realmente diferente tanto em valor quanto em tipo do valor.

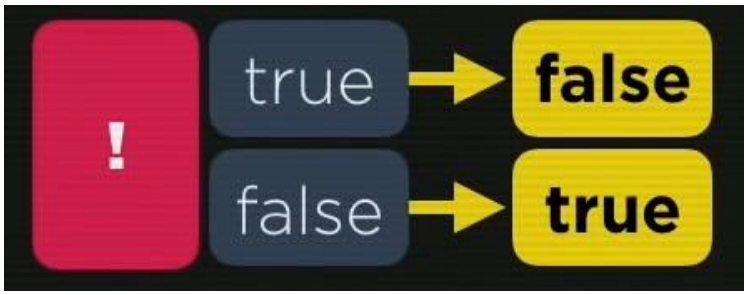
3. OPERADORES LÓGICOS

Os operadores lógicos, têm por função negar, validar duas ou mais afirmações tendo elas ser verdadeiras e validar duas ou mais afirmações tendo que pelo menos uma delas ser verdadeira. Ficou confuso? vamos validar a fundo.

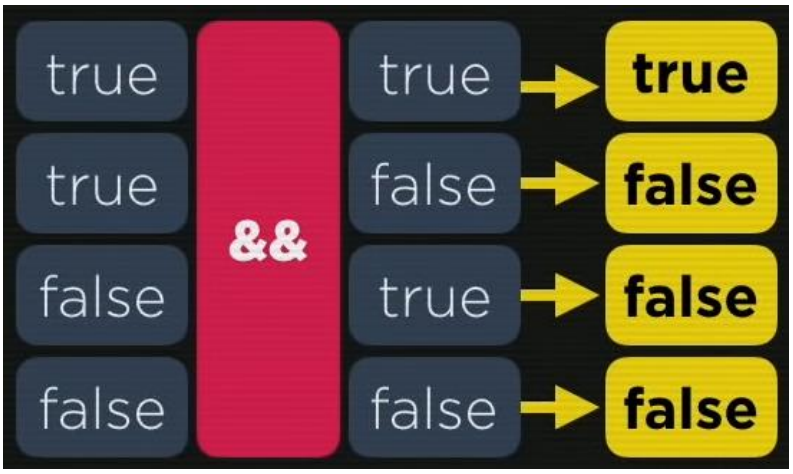
Observe a imagem abaixo, onde temos os três operadores lógicos contidos no JavaScript. Vamos entender 1 por 1.



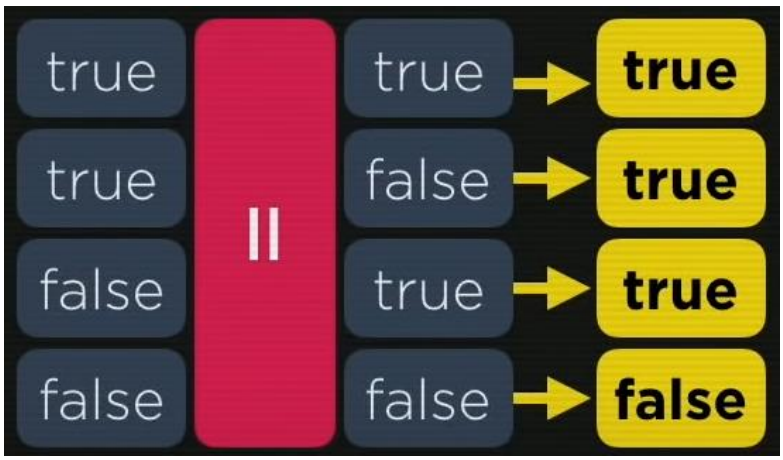
3.1. **Negação:** Esse operador tem por função negar uma informação. Isto é, se aquela informação é tida como verdadeira eu vou negá-la, tornando-a falsa e assim também o contrário. Trazendo para um exemplo mais prático, imagine que eu peço a você uma caneta azul, eu vou usar a expressão **NEGAÇÃO** nela. Agora então eu estou dizendo que a caneta azul que antes era o que eu estava requerendo não é mais o que eu quero, isto é, eu quero canetas de qualquer cor, MENOS da azul.



3.2. **Conjunção:** Esse operador tem por função validar duas ou mais informações e a conclusão dessa validação é que todas devem trazer um resultado verdadeiro. Imagina que agora eu peço a você uma caneta azul e uma vermelha. Se você me trazer só uma vermelha eu não vou ficar satisfeito, e nem se for só uma azul. Portanto só vou aceitar se você me trouxer as duas. (qualquer uma diferente das que eu pedi também não vale.



3.3. **Disjunção:** Esse operador traz por função validar também duas ou mais informações, só que, diferente da conjunção que eu necessito que todas essas validações me tragam um resultado verdadeiro, a disjunção necessita que apenas UMA afirmação seja verdadeira. Voltando ao exemplo das canetas, agora eu peço a você uma caneta azul OU uma vermelha. Não me importa se será uma ou outra, contanto que eu tenha pelo menos uma das duas.



Colocando esses operadores em prática observe o exemplo a seguir:

```
> var a = 5
undefined
> var b = 8
undefined
> true && false
false
> true && true
true
> false || false
false
> true || false
true
> true || true
true
> a > b && b % 2 == 0
false
> a <= b || b / 2 == 2
true
>
```

Como pode observar na imagem ao lado, podemos executar várias comparações utilizando o E, o OU. Observe que podemos fazer operações booleanas, e até um pouco mais complexas. As últimas duas linhas, nos trazem dois exemplos complexos.

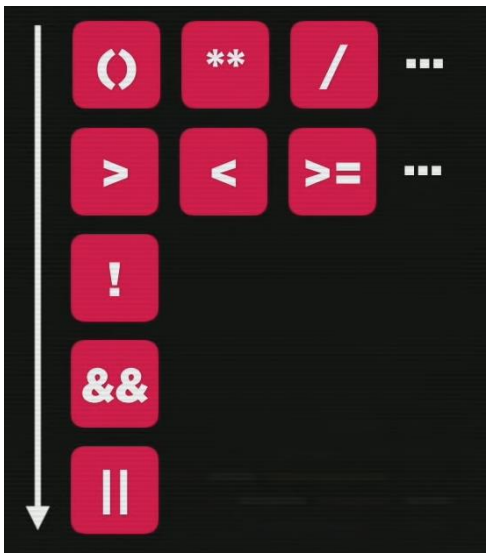
O primeiro deles faz a comparação se o **a** é **maior** que **b**, **E** se o **b resto da divisão por 2** é **igual** a **0**. Perceba, temos uma sequência para seguir para que possamos trazer o resultado esperado. Primeiro resolvemos todos os **operadores aritméticos**, depois os **relacionais** e por último os **lógicos.** Isto é, primeiro b % 2 == 0, neste caso a resposta seria **SIM,** depois validamos se a é maior que b, neste caso a resposta é **NÃO** e por fim validamos a resposta do primeiro caso (que é NÃO) e do segundo (que é SIM) com o operador lógico E, neste caso NÃO && SIM vai dar false (falso). O mesmo se aplica ao último exemplo.

Exemplos do dia a dia:

```
// Exemplos

idade >= 15 && idade <= 17          // a idade está entre 15 e 17?
estado == 'RJ' || estado == 'SP'    // o estado é RJ ou SP?
salário > 1500 && sexo != 'M'       // o salário é acima de 1500 e não é homem?
```

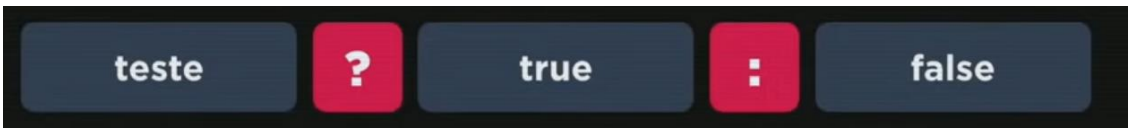
Portanto, como já foi falado na aula anterior. Temos mais coisas a adicionar às ordens de precedência dos operadores. Agora linkando aritméticos, lógicos e relacionais.



Temos na sequência então, primeiro os **aritméticos, relacionais** (ambos não importando a sequência de execução) e por fim os **lógicos** (nesse caso sim importando a sequência, isto é, primeiro **negação**, depois o **conjunção/E** e por último o **disjunção/OU**).

4. OPERADOR TERNÁRIO

O último mas não menos importante dos operadores é o Ternário. Esse operador é muito interessante a o mesmo tempo complexo. Ele trabalha semelhante ao que acontece quando utilizamos a fórmula **SE** no excel. Observe sua sintaxe na imagem abaixo:

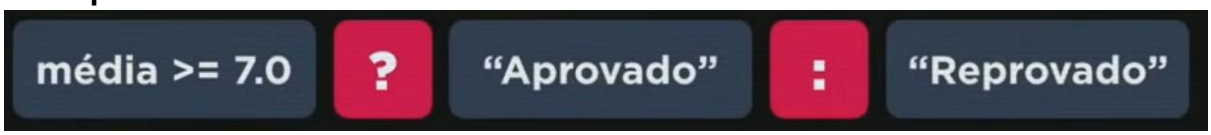


Como podemos observar, primeiro nessa operação validaremos uma afirmação, para constar se ela é verdadeira ou falsa. Em seguida nós traremos o que acontecerá daqui em diante se essa afirmação for verdadeira e por fim o que acontece se ela for falsa.

```
> var x = 8
undefined
> var res = x % 2 == 0 ? 5: 9
undefined
> x
8
> res
5
> var idade = 19
undefined
> var r = idade >= 18 ? 'MAIOR' : 'MENOR'
undefined
> r
'MAIOR'
>
```

Observe os dois exemplos na imagem ao lado. O primeiro eu estou validando se o **x resto da divisão por 2 é igual a 0** (x valendo 8) se sim vou ter a resposta na variável **res igual a 5** senão vou ter a resposta na variável **res igual a 9.** Já no segundo caso eu estou validando se a **variável idade tem o valor maior ou igual a 18**, se sim terei a **resposta 'MAIOR' na variável r** senão terei a **resposta 'MENOR' na variável r.**

Exemplos do dia a dia:



Analisando o exemplo acima, temos a verificação da média de um aluno. Se ela for maior ou igual a sete esse aluno terá como resposta **VERDADEIRA** isto é, a resultante se for verdade o texto "Aprovado". Do contrário, se essa nota for menor que sete ele terá como resposta **FALSA** o texto "Reprovado". Dependendo portanto da validação do teste lógico.