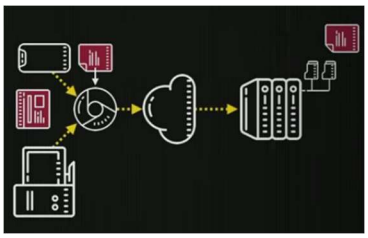
O QUE É JAVASCRIP

DIFERENÇA ENTRE UM CLIENTE E UM SERVIDOR.

Existem dois lados de aplicação da programação, o lado back-end (serverside) e o front-end (clientside), o Javascrip é mais utilizado no lado front-end, mas é também é utilizado no lado do servidor.



PARA QUE SERVE O HTML, CSS E JAVASCRIPT.

Na construção de um site existem 3 tipos de tecnologias, HTML, CSS e JavaScrip,

HTML significa HiperText Markup Language, traduzindo ao português: Linguagem de Marcação de Hipertexto. Os hipertextos são conjuntos de elementos conectados. Esses podem ser palavras, imagens, vídeos, documento, etc. Quando conectados, formam uma rede de informações que permite a comunicação de dados, organizando conhecimentos e guardando informações. Ele serve para dar significado e organizar as informações de uma página na web.

CSS é a sigla para o termo em inglês Cascading Style Sheets que, traduzido para o português, significa Folha de Estilo em Cascatas. é usado para estilizar elementos escritos em uma linguagem de marcação como HTML. O CSS separa o conteúdo da representação visual do site.

O JavaScript, diferentemente do HTML e do CSS, é uma linguagem de programação, e em conjunto com esses, ele é capaz de dar vida, gerar movimento ao site. Ele é o que tornará os elementos mais dinâmicos, pois é o JavaScript que permite a execução de scripts na página da web.

JAVASCRIPT E ELEMENTOS HTML

O Javascript pode mudar o conteúdo de um elemento HTML através do método “getElementbyId()”. O exemplo abaixo vai “procurar” no HTML o elemento com o id=”demo”, e mudar o conteúdo desse elemento (innerHTML) para “Hello JavaScript” quando o botão for clicado(onclick):

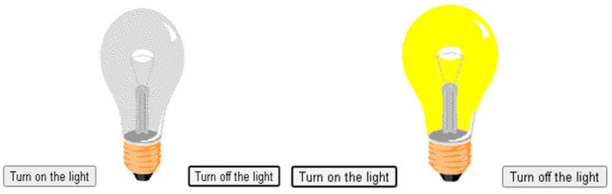
```
<h2>What Can JavaScript Do?</h2>
<p id="demo">JavaScript can change
HTML content.</p>
<button type="button"
onclick='document.getElementById("demo
").innerHTML = "Hello
JavaScript!"'>Click Me!</button>
```

O Javascript pode mudar os atributos de um elemento HTML. No próximo exemplo os valores de atributo src(source) de um elemento <img> são alterados quando um dos botões for clicado:

```
<button
onclick="document.getElementById('myImage').
src='pic_bulbon.gif'">Turn on the
light</button>

<img id="myImage" src="pic_bulboff.gif"
style="width:100px">

<button
onclick="document.getElementById('myImage').
src='pic_bulboff.gif'">Turn off the
light</button>
```



O Javascript pode mudar os atributos de estilo de um elemento HTML.

```
<p id="demo">JavaScript can change the
style of an HTML element.</p>
<button type="button"
onclick="document.getElementById('demo
').style.fontSize='35px'">Click Me!
</button>
```

O Javascript pode ocultar ou mostrar um elemento HTML. Através da alteração da propriedade de estilo “display”:

```
document.getElementById("demo").style.display = "none";

document.getElementById("demo").style.display = "block";
```

ONDE ESTA O JAVASCRIPT

No HTML, O código JavaScript é inserido entre as tags <script> e </script>.

```
<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>
document.getElementById("demo").innerH
TML = "Saudades do Bob";
</script>
```

JavaScript in Body

Saudades do Bob

É possível inserir qualquer quantidade de scripts em um documento HTML. Scripts podem ser inseridos dentro da seção <body>, ou dentro da seção <head>, ou em ambos ao mesmo tempo.

O exemplo a seguir altera um parágrafo invocando (chamando) uma função quando o botão for clicado, o script pode ser inserido tanto no <head>, quanto <body>do documento HTML.

```
<!DOCTYPE html>
<html>
<head>
<script>
function myFunction() {
 document.getElementById("demo").innerHTM
L = "Paragraph changed.";
}
</script>
</head>
<body>
<h2>Demo JavaScript in Head</h2>
<p id="demo">A Paragraph.</p>
<button type="button"
onclick="myFunction()">Try it</button>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<h2>Demo JavaScript in Body</h2>
<p id="demo">A Paragraph.</p>
<button type="button"
onclick="myFunction()">Try it</button>
<script>
function myFunction() {
 document.getElementById("demo").innerHTM
L = "Paragraph changed.";
}
</script>
</body>
```

O elemento HTML <head> providencia informações gerais (metadados) sobre o documento HTML, incluindo seu título e link, o elemento <body> do HTML representa o conteúdo de um documento HTML. **Inserir scripts na parte inferior do elemento <body> melhora a velocidade de exibição**, já que, posicionar ele no <head> faz o navegador interpretar esse script primeiro, tornando a exibição mais lenta.

Os scripts também podem ser posicionados em arquivos externos, scripts externos são mais práticos quando alguns códigos são usados por diferentes páginas web. Arquivos Javascript possuem a extensão “.js”. Para utilizar um script externo, deve ser inserido o nome do arquivo do script no atributo src(source) da tag <script>:

External file: myScript.js

```
function myFunction() {
  document.getElementById("demo").innerHTML = "Paragraph changed.";
}
```

```
<!DOCTYPE html>
<html>
<body>
<h2>Demo External JavaScript</h2>
<p id="demo">A Paragraph.</p>
<button type="button"
onclick="myFunction()">Try it</button>
<p>This example links to
"myScript.js".</p>
<p>(myFunction is stored in
"myScript.js")</p>
<script src="myScript.js"></script>
</body>
</html>
```

Colocar scripts em arquivos externos tem algumas **vantagens**: separa HTML e código, torna o HTML e o JavaScript mais fáceis de ler e manter, arquivos JavaScript em cache podem acelerar o carregamento da página. Um script externo pode ser **referenciado** de 3 maneiras diferentes: Com um URL completo (um endereço web completo); com um caminho de arquivo (como /js/); sem nenhum caminho (como no exemplo acima).

EXIBIÇÃO JAVASCRIPT

O JavaScript pode “exibir” dados de diferentes formas: escrevendo dentro de um elemento HTML, usando “innerHTML”. Escrevendo dentro da saída do HTML usando “document.write()”. Escrevendo em uma caixa de alerta no navegador, usando “window.alert()”. Escrevendo no console do navegador, usando “console.log()”.

USANDO innerHTML

Para acessar um elemento HTML, o Javascript pode usar o método “document.getElementById(id)”. O atributo “id” define o elemento HTML, **a propriedade “innerHTML” define o conteúdo desse elemento**. Alterar a propriedade “innerHTML” de um elemento HTML é uma **maneira comum** de exibir dados em HTML.

```
<body>
<h2>My First Web Page</h2>
<p>My First Paragraph.</p>
<p id="demo"></p>
<script>
document.getElementById("demo")
.innerHTML = 5 + 6;
</script>
</body>
```

USANDO document.write()

Para propósitos de testes, é conveniente usar o método document.write(), pois, utilizá-lo após um documento HTML ser carregado, vai deletar todo o HTML existente.

```
<!DOCTYPE html>
<html>
<body>
<h2>My First Web Page</h2>
<p>My first paragraph.</p>
<button type="button"
onclick="document.write(5 + 6)">Try
it</button>
</body>
</html>
```

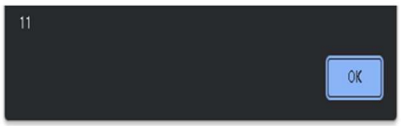
USANDO window.alert()

É possível utilizar uma caixa de alerta no navegador para exibir dados, a palavra-chave “window” pode ser pulada, já que, o objeto “window” é por padrão um objeto de escopo global, isso significa que variáveis, propriedades, e

métodos por padrão pertencem ao objeto “window”. **Isso significa que especificamente a palavra-chave “window” é opcional**.

```
<script>
window.alert(5 + 6);
</script>
```

```
<script>
alert(5 + 6);
</script>
```



USANDO console.log()

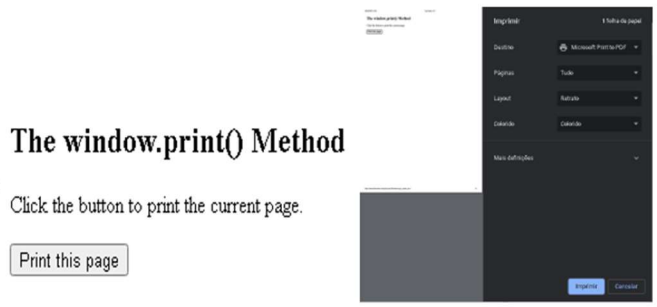
Para propósito de depuração de erros (debugging), é possível chamar o método “console.log()” em um navegador para exibir dados no console.

```
<!DOCTYPE html>
<html>
<body>
<h2>Activate Debugging</h2>
<p>F12 on your keyboard will activate debugging.</p>
<p>Then select "Console" in the debugger menu.</p>
<p>Then click Run again.</p>
<script>
console.log(5 + 6);
</script>
</body>
</html>
```

JavaScript Print

JavaScript não tem nenhum objeto de impressão ou métodos de impressão. Você não pode acessar dispositivos de saída do JavaScript. A única exceção é que você pode chamar o método window.print() no navegador para imprimir o conteúdo da janela atual.

```
<!DOCTYPE html>
<html>
<body>
<h2>The window.print() Method</h2>
<p>Click the button to print the current page.</p>
<button onclick="window.print()">Print this page</button>
</body>
</html>
```



DECLARAÇÕES JAVASCRIPT

Um programa de computador é uma lista de “instruções” a serem “executadas” por um computador. Em uma linguagem de programação, essas instruções são chamadas de **statements (declarações)**. Um programa JavaScript é uma lista de declarações programáveis. Programas JavaScript e **declarações** JavaScript são frequentemente **chamados por códigos JavaScript**. No HTML, os programas JavaScript são executados pelo navegador.

As declarações JavaScript podem ser compostas por: Valores, Operadores, Expressões, Keywords e comentários. Muitos programas Javascript contém muitas declarações Javascript, essas declarações (instruções) são executadas, uma por uma, seguindo a ordem em que elas foram escritas.

PONTO E VÍRGULA ;

O sinal gráfico ponto e vírgula separa as declarações JavaScript, ele deve ser adicionado no final de cada declaração executável.

```
let a, b, c;  // Declare 3 variables
a = 5;        // Assign the value 5 to a
b = 6;        // Assign the value 6 to b
c = a + b;    // Assign the sum of a and b to c
```

Quando separadas por ponto e vírgula, múltiplas declarações podem ser alocadas em uma única linha para facilitar a leitura.

ESPAÇOS EM BRANCO

O JavaScript ignora múltiplos espaços, é possível adicioná-los para tornar o script mais legível. Os seguintes exemplos são equivalentes, porém o primeiro apresenta um código mais limpo.

```
let person = "Hege";
let person="Hege";
```

Também é uma boa prática adicionar espaços ao redor dos operadores (= + = * /).

LARGURA DE LINHA E QUEBRA DE LINHA

Para tornar os códigos mais legíveis, programadores frequentemente evitam linhas de códigos maiores do que 80 caracteres. Se uma declaração JavaScript não couber em uma linha, a melhor maneira é quebra-la após um operador.

BLOCOS DE CÓDIGO

As declarações JavaScript podem ser agrupadas juntas em blocos de código dentro de chaves {...}. O propósito dos blocos de código é definir as declarações que vão ser executadas juntas.

PALAVRAS-CHAVE

As declarações JavaScript frequentemente começam após uma palvra-chave (**keyword**) para o JavaScript identificar a ação que está sendo realizada.

Exemplo de algumas Keywords:

- **var:** Declara uma variável
- **let:** Declara uma variável de bloco
- **const:** Declara uma constante de bloco
- **if:** Marca um bloco de instruções a serem executadas em uma condição
- **switch**: Marca um bloco de instruções a serem executadas em diferentes casos
- **for:** Marca um bloco de instruções a serem executadas em um loop
- **function:** Declara uma função
- **return:** Sai de uma função
- **try:** Implementa o tratamento de erros para um bloco de instruções

SINTAXE JAVASCRIPT

A sintaxe em JavaScript é um conjunto de regras que define como os programas em JavaScript devem ser construídos.

VALORES

Existem dois tipos de valores em JavaScript: Valores fixos e Valores variáveis. Valores fixos sã chamados de literais (**Literals**) e valores variáveis chamados de variáveis (**Variables**).

As regras mais importantes para **valores fixos (Literals)** são: números são escritos com ou sem o seu decima; strings são texto, escrito com aspas duplas ou aspas simples.

Em uma linguagem de programação, **variáveis (variables)** são usadas para armazenar valores. JavaScript usa as keywords **var**, **let** e **const** para declarar variáveis. O sinal gráfico de igualdade (=) é usado para atribuir valores a uma variável. No seguinte exemplo, x é definido como uma variável, após isso, é atribuído (“x recebe”) o valor 6:

```
let x;
x = 6;
```

EXPRESSÕES

Uma expressão é uma combinação de valores, variáveis e operadores que calculam um valor. Na computação isso é chamado de evaluation (estimativa ou avaliação). Como por exemplo, cinco vezes dez é igual a cinquenta, ou em uma expressão: 5 * 10. Expressões também podem conter valores de variáveis: x * 10. Os valores podem ser de vários tipos como números e strings, por exemplo, a expressão “John” + “ “ + “Doe”, ser igual a “John Doe”.

COMENTÁRIOS

Nem todas as declarações são “executadas”, códigos escritos após barras duplas **//** ou entre **/*** e ***/** são considerados como comentários. Comentários são ignorados e não será possível executá-los.

IDENTIFICADORES / NOMES

Identificadores são nomes em JavaScript, eles são usados para nomear variáveis, palavras-chave e funções. As regras para nomes permitidos são as mesmas para a maioria das linguagens de programação. Em JavaScript o nome deve começar com:

- Uma letra (A-Z ou a-z)
- Um sinal de cifrão ($)
- Ou um sublinhado (_)

Números não podem ser alocados para o primeiro caractere em um nome, para facilitar o JavaScript distinguir um número de um nome.

OUTRAS SINTAXES

JavaScript é uma linguagem de programação **case sensitive**, ou seja, ela identifica como diferentes os caracteres maiúsculos e caracteres minúsculos. No exemplo a seguir as variáveis lastName e lastname são diferentes variáveis:

```
let lastname, lastName;
lastName = "Doe";
lastname = "Peterson";
```

Historicamente, programadores têm usado diferentes maneiras de adicionar **múltiplas palavras** em um único nome de uma variável. Uma das maneiras é utilizando o sinal de hífen (-), porém em JavaScript esse sinal é reservado para ser utilizado como sinal de subtração. Algumas das maneiras são: utilização do underscore (_) “first_name”; **Camel Case** maiúsculo (Pascal case) “FirstName”; Camel Case minúsculo “firstName”. Programadores JavaScript tendem a utilizar camel case iniciado com letras minúsculas.

JavaScript usa o conjunto de caracteres **Unicode**. Unicode cobre (quase) todos os caracteres, pontuações e símbolos do mundo.

https://www.w3schools.com/charsets/ref_html_utf8.asp

VARIÁVEIS JAVASCRIPT

Variáveis são containers para armazenamento de dados (armazenar valores de dados). Existem 4 maneiras de declarar uma variável em JavaScript, usando “var”, usando “let”, usando “const” ou não usando nada. Porém uma variável deve sempre ser declarada usando var, let ou const. A keyword “var” é usada em códigos JavaScript de 1995 até 2015. As keywords “let” e “const” foram adicionadas em 2015, códigos que devem rodar em navegadores mais antigos devem usar “var”.

IDENTIFICADORES

Todas as variáveis JavaScript devem ser identificadas com nomes exclusivos. Esses nomes exclusivos são chamados de identificadores .Os identificadores podem ser nomes curtos (como x e y) ou nomes mais descritivos (idade, soma, volume total).As regras gerais para construir nomes para variáveis (identificadores exclusivos) são:

- Os nomes podem conter letras, dígitos, sublinhados e cifrões.
- Os nomes devem começar com uma letra
- Os nomes também podem começar com $ e _ (mas não o usaremos neste tutorial)
- Os nomes diferenciam maiúsculas de minúsculas (y e Y são variáveis diferentes)
- Palavras reservadas (como palavras-chave JavaScript) não podem ser usadas como nomes

DECLARAÇÃO E ATRIBUIÇÃO

A ação de criar uma variável em JavaScript é chamada de “declarar” uma variável. Uma variável é declarada com as keywords var ou let (**var** carName; ou **let** carName;). Após a declaração, a variável não terá um valor (tecnicamente ela tem um valor “udenfined”). Para atribuir um valor a uma variável é utilizado o sinal de igual (`carName = "Volvo";`). Em JavaScript, o sinal de igual (=) é um operador de “atribuição”, não um sinal de “igual a”. É possível também atribuir valor a variável quando ela for declarada (`let carName = "Volvo";`), para declarar uma variável com const deve-se sempre atribuir valores durante a declaração (`const price1 = 5;`). É uma boa prática de programação sempre **declarar as variáveis no início do script.**

UMA DECLARAÇÃO, MUITAS VARIÁVEIS

É possível declarar muitas variáveis em uma única declaração, apenas iniciando a declaração com “let” e separando as seguintes com ponto e vírgula:

```
let person = "John Doe", carName = "Volvo", price = 200;
```

A declaração também pode ser feita em múltiplas linhas para uma leitura mais agradável do código.

VALUE = UNDEFINED

Na programação, variáveis são frequentemente declaradas sem um valor. O valor pode ser algo que ainda vai ser calculado, ou fornecido mais tarde, como por exemplo, através de input do usuário. **Uma variável declarada sem um valor vai ter um valor “undefined”**.

VARIÁVEIS DECLARADAS COM LET

A keyword let foi introduzida no ES6 (2015), variáveis definidas com let não podem ser redeclaradas, devem ser declaradas antes do uso e possuem escopo de bloco (Block Scope). Uma variável declarada com let pode ter seu valor reatribuído, porém ela não pode ser redeclarada.

ESCOPO DE BLOCO (BLOCK SCOPE)

Antes do ES6(2015), JavaScript tinha apenas os chamados “escopo global” e “escopo de função”. ES6 introduziu duas importantes novas keywords: “let” e “const”. Essas keywords providenciaram o “escopo de bloco” (**Block Scope**) no JavaScript. Variáveis declaradas dentro de um sinal de chaves {...}(bloco) não podem ser acessadas fora desse bloco. Variáveis declaradas com a keyword var não pode ter um escopo de bloco, pois elas podem ser acessadas fora do bloco.

```
{
  var x = 2;
}
// x CAN be used here
```

Redeclarar uma variável com let dentro de um bloco não redeclarará a variável fora do bloco, evitando problemas que aconteceriam ao utilizar var.

```
let x = 10;
// Here x is 10
{
let x = 2;
// Here x is 2
}
// Here x is 10
```

Redeclarar uma variável com var é possível, porém, **não é possível redeclarar uma variável com let dentro do mesmo bloco**.

```
var x = 2;// Allowed
let x = 3;// Not allowed
{
let x = 2;// Allowed
let x = 3 // Not allowed
}
{
let x = 2;// Allowed
var x = 3 // Not allowed
}
```

LET HOISTING (IÇAMENTO)

Variáveis definidas com var são içadas (hoisted) para o topo e podem ser inicializadas a qualquer momento. Isso significa que é possível usar uma variável antes de ela ser declarada. **Variáveis definidas com let e const** também são içadas até o topo de um bloco, porém não podem ser inicializadas. Isso significa que usar uma variável (atribuir um valor) antes de ela ser declarada vai resultar em um erro (`ReferenceError`).

```
carName = "Volvo";
var carName;
```

```
carName = "Saab";
let carName = "Volvo";
```

ReferenceError: Cannot access 'carName' before initialization

VARIÁVEIS DECLARADAS COM CONST

A keyword const foi introduzida em ES6 (2015). Variáveis definidas com const não podem ser redeclaradas. Variáveis definidas com const não podem ser reatribuídas. As variáveis definidas com const têm o escopo do bloco. As variáveis declaradas com const não podem ter seus valores reatribuídos e não pode ser redeclarada.

QUANDO USAR CONST?

As variáveis const devem ter seus devidos valores atribuídos durante a sua declaração. De regra geral uma variável é atribuída com const sempre que você souber que seu valor não será alterado mais tarde.

CONSTANT OBJECTS AND ARRAYS

A keyword const possuí uma pegadinha, ela é uma abreviação para constant (constante), porém, ela não define um valor constante, ela define uma referência constante a um valor. Por conta disso não é possível reatribuir um valor constante, reatribuir um array constant e reatribuir um objeto constante. Mas é possível mudar elementos de um array constante e mudar propriedades de um objeto constate.

```
// You can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];
// You can change an element:
cars[0] = "Toyota";
// You can add an element:
cars.push("Audi");
```

```
const cars = ["Saab", "Volvo", "BMW"];
cars = ["Toyota", "Volvo", "Audi"];    // ERROR
```

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};
// You can change a property:
car.color = "red";
// You can add a property:
car.owner = "Johnson";
```

```
const car = {type:"Fiat", model:"500", color:"white"};
car = {type:"Volvo", model:"EX60", color:"red"};    // ERROR
```

ESCOPO DE BLOCO (BLOCK SCOPE)

É semelhante a declaração de uma variável com const e let quando se trata de escopo de bloco. No uma variável declarada com const dentro de um bloco não é a mesma declarada fora do bloco.

TIPOS DE DADOS

Na programação, é importante entender o conceito dos tipos de dados para escrever expressões corretamente e evitar possíveis erros.

STRING

Uma string é uma série de caracteres como “John Doe”. Strings são escritas com aspas simples ou aspas duplas. É possível utilizar aspas dentro de uma string desde que elas não sejam as mesmas que definem essa string. (`"He is called 'Johnny'"`).

NUMBER

JavaScript possui apenas um tipo de número, eles podem ser escritos com ou sem seus decimais, e números muito grandes podem ser escritos com notação cientifica (`123e5, 34.00`).

BOOLEAN

Tipos booleanos possuem apenas dois valores: true (verdadeiro) ou false (falso), eles são frequentemente usados após testes condicionais (`let x = 5; let y = 5; let z = 6;(x == y// Returns true (x == z// Returns false`).

UNDEFINED

Em JavaScript, uma variável sem um valor tem o valor **undefined**. Qualquer variável pode ser considerada vazia configurando o valor para undefined. Isso significa que a variável ainda não foi preenchida com algum outro tipo de valor, logo um valor indefinido (undefined) preenche essa variável. Porém é importante lembrar que como os valores são indefinidos, o tipo dessa variável também é indefinido.

NULL

Em JavaScript null é “nada”. Supostamente isso torna inexistente algo com esse valor. Porem, em javascript null é considerado um objeto. É possível tornar um objeto vazio definindo-o como null. Undefined e null são iguais em valor, mas diferentes em tipo.

```
typeof undefined //undefined typeof null// object
```

OPERAÇÕES ARITMÉTICAS

Operadores aritméticos executam cálculos aritméticos em números.

OPERADORES

Os números são chamados de operandos, os símbolos que executam as operações entre os operandos são chamados de operadores.

- **ADIÇÃO (+):** O operador de adição (+) realiza uma soma entre os números.
- **SUBTRAÇÃO (+):** O operador de subtração (-) realiza a subtração dos números.
- **MULTIPLICAÇÃO (*):** O operador de multiplicação (*) multiplica os números
- **DIVIZÃO (/):** O operador de adição (+) realiza uma soma entre os números.
- **RESTO (%):** O operador de módulo (%) retorna o restante da divisão dos números.
- **INCREMENTO (++):** O operador de incremento (++) adiciona o valor 1 a um número.
- **DECREMENTO (++):** O operador de decremento (--) retira o valor 1 de um número.
- **EXPONENCIAÇÃO (**):** O operador de exponenciação (**) eleva o primeiro operando à potência do segundo operando.

OPERADORES DE COMPARAÇÃO

Operadores de comparação são usados em declarações lógicas para determinar a igualdade ou diferença entre variáveis ou valores. No exemplo a seguir o valor 5 foi atribuído a variável x (`x = 5`), a tabela demonstra a utilização desses comparadores e a sua descrição.

OPERADOR	DESCRIÇÃO	EXEMPLO	RETORNO
==	Igual a	x == 8	false
		x == 5	true
		x == "5"	true
===	Igual valor e tipo (idêntico)	x === 5	true
		x === "5"	false
!=	Diferente	x != 8	true
!==	Diferente em valor ou tipo	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	Maior que	x > 8	false
<	Menor que	x < 8	true
>=	Maior ou igual a	x >= 8	false
<=	Menor ou igual a	x <= 8	true

OPERADORES LÓGICOS

Operadores lógicos são usados para determinar uma lógica entre variáveis ou valores. No exemplo a seguir foi atribuído o valor 6 a variável x (x = 6) e o valor 3 a variável y (y = 3), a tabela demonstra a lógica dos operadores:

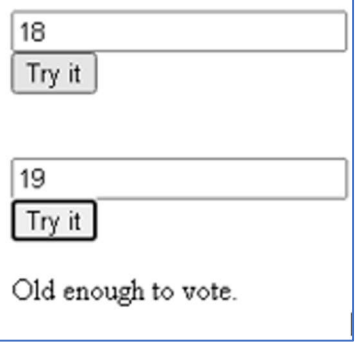
OPERADOR	DESCRIÇÃO	EXEMPLOS
&&	e	(x < 10 **&&** y > 1) is true
||	ou	(x == 5 **||** y == 5) is false
!	negação	**!**(x == y) is true (**!**false y) is true (**!**true y) is false

OPERADOR CONDICIONAL (TERNÁRIO)

JavaScript também contém um operador condicional que atribui um valor a uma variável baseado em uma condição. Esse operador é representado pelo sinal de interrogação (**?**) em conjunto com o sinal de dois pontos (**:**). Ou seja, eles podem ser interpretados como “if” e “else” (se e senão).

nomedavariavel = (condição) **?** valor1 **:** valor2

```
<!DOCTYPE html>
<html>
<body>
<input id="age" value="18" />
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction() {
  let age = document.getElementById("age").value;
  let voteable = (age < 18) ? "Too young":"Old
enough";
  document.getElementById("demo").innerHTML =
voteable + " to vote.";
}
</script>
</body>
</html>
```



OPERADORES BITWISE

OPERADOR	NOME	DESCRIÇÃO
&	AND (E)	Sets each bit to 1 if both bits are 1
|	OR (OU)	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT (NÃO)	Inverts all the bits
<<	Zero fill left shift (Deslocamento à esquerda de preenchimento zero)	Shifts left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift (Deslocamento à direita)	Shifts right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off
>>>	Zero fill right shift (Deslocamento à direita de preenchimento zero)	Shifts right by pushing zeros in from the left, and let the rightmost bits fall off

Exemplos:

OPERAÇÃO	RESULTADO	MESMO QUE	RESULTADO
5 & 1	1	0101 & 0001	0001
5 | 1	5	0101 | 0001	0101
~ 5	10	~0101	1010
5 << 1	10	0101 << 1	1010
5 ^ 1	4	0101 ^ 0001	0100
5 >> 1	2	0101 >> 1	0010
5 >>> 1	2	0101 >>> 1	0010

OPERADORES DE ATRIBUIÇÃO

Os operadores de atribuição atribuem valores a uma variável. Eles são utilizados para deixar o código mais “limpo”. São eles:

OPERADOR	EXEMPLO	MESMO QUE
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y

TYPEOF

Em JavaScript existem 5 diferentes tipos de dados que podem contar valores:

- string
- number
- boolean
- object
- function

Existem 6 tipos de objetos:

- Object
- Date
- Array
- String
- Number
- Boolean

E 2 tipos de que não podem conter valores:

- null
- undefined

O operador **typeof** pode ser usado para verificar o tipo de dado de uma variável em JavaScript.

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined" *
typeof null                   // Returns "object"
```

É possível observar que:

- O tipo de dados de NaN é número
- O tipo de dados de uma array é objeto
- O tipo de dados de uma data é objeto
- O tipo de dados de null é objeto
- O tipo de dados de uma variável indefinida é undefined *
- O tipo de dados de uma variável que não recebeu um valor também é undefined*

CONVERSÃO DE TIPO

Em JavaScript uma variável pode ser convertida em uma nova variável e em outro tipo de dado.

CONVERTER EM NUMBER

O método global **Number()** pode converter dados em numbers (método global pois ele pode ser usado para todos os tipos de dados, não somente strings), strings contendo números (como “3.14”) podem ser convertidas em números (como 3.14). Strings vazias serão convertidas para 0.

O método **parseInt()** analisa (tradução do verbo parses é “analisar uma palavra”) uma string e retorna um número inteiro. Espaços são permitidos. Somente o primeiro valor é convertido.

```
parseInt("-10");          -10
parseInt("-10.33");       -10
parseInt("10");           10
parseInt("10.33");        10
parseInt("10 20 30");     10
parseInt("10 years");     10
parseInt("years 10");     NaN
```

Do mesmo modo, o método **parseFloat()** analisa uma string e retorna um número. Espaços são permitidos, mas somente o primeiro número é convertido.

```
parseFloat("10");          10
parseFloat("10.33");       10.33
parseFloat("10 20 30");    10
parseFloat("10 years");    10
parseFloat("years 10");    NaN
```

CONVERTER EM STRING

O método global **String()** pode converter números em strings. Ele pode ser usado em qualquer tipo de número, literais, variáveis ou expressões.

```
String(x)          123
String(123)        123
String(100 + 23)   123
```

O método **toString()** retorna um número como uma string. Todos os métodos numéricos podem ser usados em qualquer tipo de número (literais, variáveis ou expressões):

```
let x = 123;
x.toString();
(123).toString();
(100 + 23).toString();
```

```
123
123
123
```

Outros métodos de conversões numéricas:

Method	Description
toExponential()	Returns a string, with a number rounded and written using exponential notation.
toFixed()	Returns a string, with a number rounded and written with a specified number of decimals.
toPrecision()	Returns a string, with a number written with a specified length

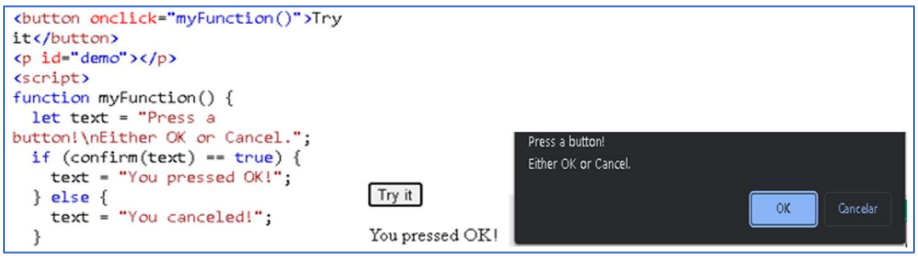
ALERT(), CONFIRM() E PROMPT()

Window alert()

O método **alert()** exibe uma caixa de alerta com um botão de “OK”. É usado quando se deseja que certo tipo de informação chegue ao usuário. A caixa de alerta tira o foco da janela atual do navegador e, força o usuário a ler a mensagem. É importante não abusar desse método, pois ele Impede que o usuário acesse outras partes da página até que a caixa de alerta seja fechada.

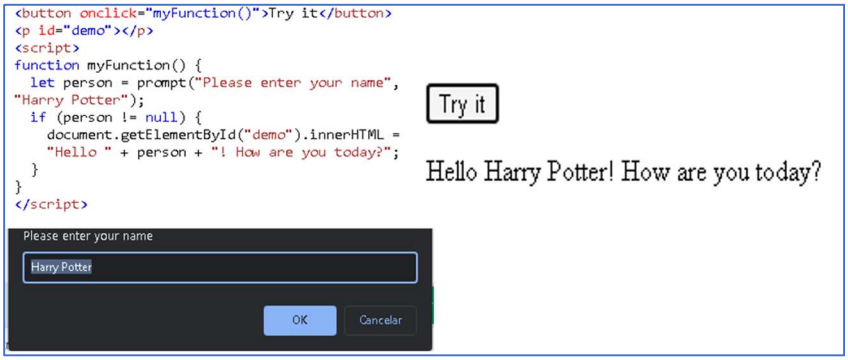
Window confirm()

O método **confirm()** exibe uma caixa de diálogo com uma mensagem, um botão de “OK”, e um botão de “Cancel”. Ele pode ser utilizado para receber dvalores de tipo booleano do usuário, true se o usuário clicou em “OK”, e false caso clique em “Cancel”.



Window prompt()

O método **prompt()** exibe uma caixa de diálogo que solicita um input do usuário. O método prompt() retorna o valor de entrada se o usuário clicar em "OK", caso contrário, retorna null.



INSTRUÇÕES DE CONDIÇÕES IF, ELSE E ELSE IF

Instruções condicionais são usadas para realizar diferentes ações baseadas em diferentes condições.

INSTRUÇÃO CONDICIONAL if

A instrução if é usada para especificar um bloco de código JavaScript que será executado se uma determinada condição for verdadeira. O exemplo vai gerar a frase de saudação “Good day” **se** as horas forem menores que 18:00.

```
<p id="demo"></p>
<script>
if (new Date().getHours() < 18) {
 document.getElementById("demo").inner
HTML = "Good day!";
}
</script>
```

INSTRUÇÃO CONDICIONAL else

A instrução else é usada para especificar um bloco de código JavaScript que será executado se uma determinada condição for falsa. O exemplo demonstra isso, **se** as horas forem menores que 18 (**condição verdadeira if**), a saudação “Good day” será atribuída como valor a uma variável, **senão** (**condição falsa else**), o valor atribuído será “Good Evening”, e após a execução desse código a variável será inserida em um determinado parágrafo.

```
<p id="demo"></p>
<script>
const hour = new Date().getHours();
let greeting;
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>
```

INSTRUÇÃO CONDICIONAL else if

É possível observar que as estruturas if e else são frequentemente utilizadas em conjunto, seu a condição for verdadeira e se a condição for falsa, ocasionando em somente duas possibilidades para a execução do bloco de código. Uma das maneiras de contornar isso é utilizando a instrução else if. Essa instrução será usada para especificar uma nova condição se a primeira condição for falsa.

No exemplo, se a hora for menor que 10:00, será criada uma saudação "Good morning", senão, mas se a hora for menor que 20:00, crie uma saudação "Good day", caso contrário, uma saudação "Good Evening":

```
<p id="demo"></p>
<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>
```

INSTRUÇÃO CONDICIONAL SWITCH

A instrução switch será usada para realizar uma ação diferente baseada em uma diferente condição. Pode ser utilizada para selecionar um dentre muitos códigos de blocos que serão executados.

No exemplo o método getDay() retorna o dia da semana como um número entre 0 e 6. Se hoje for sábado (6) ou domingo (0) uma mensagem será escrita, senão uma mensagem padrão será escrita:

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

SINTAXE:

É possível observar que: A expressão switch é avaliada uma vez; O valor da expressão é compara com os valores de cada um dos casos; Se houver uma correspondência, o bloco de código associado será executado; Se não houver correspondência, o bloco de código padrão (default) será executado.

```
switch(expressão) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

KEYWORDS BREAK E DEFAULT

Quando o código JavaScript switch atinge uma keyword **break**, ele sai do bloco switch. Isso vai interromper a execução dentro do bloco switch. Se omitir a instrução break, o próximo case será executado mesmo que a avaliação não corresponda ao case. Não é necessário quebrar o último caso em um bloco de comutação. O bloco quebra (termina) lá de qualquer maneira.

A keyword default (padrão) especifica o código a ser executado se não houver correspondência em nenhum dos cases(casos).

ESTRUTURAS DE REPETIÇÃO

Repetições (loops) podem executar um bloco de código um determinado número de vezes. O JavaScript suporta diferentes tipos de loops:

- **for** - percorre um bloco de código várias vezes
- **for/in** - percorre as propriedades de um objeto
- **for/of** - percorre os valores de um objeto iterável
- **while** - percorre um bloco de código enquanto uma condição especificada é verdadeira
- **do/while** - também percorre um bloco de código enquanto uma condição especificada for verdadeira

FOR LOOP

A estrutura de repetição for, é geralmente utilizado quando se sabe os limites da execução das repetições, ao contrário da repetição while. A repetição for tem a seguinte sintaxe:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Declaração 1 é executado (uma vez) antes da execução do bloco de código. **Declaração 2** define a condição para execução do bloco de código. **Declaração 3** é executado (todas as vezes) após o bloco de código ter sido executado.

No exemplo, a declaração 1 define a variável antes do loop começar (let i = 0. A declaração 2 define a condição para que o loop seja executado (i deve ser menor do que 5). A declaração 3 acrescenta um valor (i++) cada vez que o bloco de código no loop for executado.

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4

WHILE LOOP

O loop **while** percorre um bloco de código **enquanto** uma condição especificada for verdadeira. É preciso estar atento para utilizar essa repetição pois diferente da repetição for, ela não possui um ponto final para ela ser interrompida, ou seja, ela vai se repetir infinitamente e isso pode causar erros no navegador. A estrutura de repetição while tem a seguinte sintaxe:

```
while (condition) {
  // code block to be executed
}
```

No exemplo a seguir o código dentro do bloco vai se repetir, de novo e de novo, enquanto a variável (i) for menor do que 10:

```
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
```

The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
The number is 7
The number is 8
The number is 9

DO WHILE LOOP

O do while loop é uma variante do loop while. Este loop executará o bloco de código uma vez, antes de verificar se a condição seja verdadeira, então repetirá o loop enquanto a condição for verdadeira. O loop sempre será executado pelo menos uma vez, mesmo que a condição seja falsa, pois o bloco de código é executado antes que a condição seja testada:

```
do {
  // code block to be executed
}
while (condition);
```

FOR IN LOOP

Em JavaScript a estrutura de repetição **for in** é executada entre as propriedades de um objeto. A sua sintaxe deve ser:

```
for (key in object) {
  // code block to be executed
}
```

(for in, está relacionado ao index do objeto)

No exemplo abaixo, foi criado uma variável nomeada de “person”, foram atribuídas propriedades a essa variável como o fname(first name), lname(last name) e age, o loop for in, itera sobre esse objeto, cada iteração retorna uma chave (definida como x em “let x”), cada chave é usada para acessar o endereço chave de cada propriedade do objeto, e no exemplo, atribuir o valor correspondente ao index à variável “txt” .

```
const person = {fname:"John", lname:"Doe", age:25};
let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}
```

John Doe 25

FOR OF LOOP

A instrução JavaScript for of percorre os valores de um objeto iterável. Ela permite que você faça um loop sobre estruturas de dados iteráveis, como Arrays, Strings, Maps, NodeLists e muito mais:

```
for (variable of iterable) {
  // code block to be executed
}
```

(for of, está relacionado ao valor do objeto)

```
const cars = ["BMW", "Volvo", "Mini"];
let text = "";
for (let x of cars) {
  text += x + "<br>";
}
```

BMW
Volvo
Mini

ARRAYS (VETORES)

Um array é uma variável especial, que pode conter mais de um valor:

SINTAXE

Uma array é comummente declarada com a keyword const. Espaços e quebras de linha não são importantes. Uma declaração pode abranger várias linhas

```
const array_name = [item1, item2, ...];
```

```
const cars = ["Saab", "Volvo", "BMW"];
```

É possível criar um array utilizando a keyword “new Array()”.

```
const cars = new Array("Saab", "Volvo", "BMW");
```

Os dois exemplos fazem exatamente o mesmo. Não há necessidade de usar new Array(). Para simplicidade, legibilidade e velocidade de execução, use o método literal de matriz.

ELEMENTOS DO ARRAY

Os elementos de uma array são acessados e alterados referindo-se ao número do index dele. O index de uma array começa com 0.

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];
```

Saab

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
```

MÉTODOS E PROPRIEDADES DA ARRAY

METHOD	DESCRIPTION
concat()	Joins two or more arrays, and returns a copy of the joined arrays
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array pass a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that pass a test
find()	Returns the value of the first element in an array that pass a test
findIndex()	Returns the index of the first element in an array that pass a test
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Check if an array contains the specified element
indexOf()	Search the array for an element and returns its position
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Search the array for an element, starting at the end, and returns its position
map()	Creates a new array with the result of calling a function for each array element
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)
reduceRight()	Reduce the values of an array to a single value (going right-to-left)
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

PROPERTY	DESCRIPTION
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array
prototype	Allows you to add properties and methods to an Array object

FUNÇÕES

Uma função JavaScript é um bloco de código projetado para executar uma tarefa especifica. Uma função é executada quando ela é “invocada” (chamada).

SINTAXE

Para declarar uma função, primeiro ela deve ser definida com a keyword **function**, seguido pelo seu **nome** e seguido por parênteses **()**. O nome da função pode conter letras, dígitos underscores e cifrão (mesma regra para nomes de variáveis). **Entre os parênteses são incluídos os nomes dos parâmetros**, separados por vírgulas. **O código** que será executado pela função, é colocado **dentro de chaves { }**.

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

Os parâmetros são listados dentro dos parênteses () na definição da função. **Os argumentos são valores** recebidos pela função quando ela for chamada (invocada). Dentro da função os argumentos (parâmetros) se comportam como variáveis.

```
<p id="demo"></p>
<script>
function myFunction(p1, p2) {
  return p1 * p2;
}
document.getElementById("demo").innerHTML = myFunction(4, 3);
</script>
```

12

RETURN

Quando o JavaScript atinge uma declaração return, a execução da função vai ser parada. As funções geralmente calculam um valor de retorno. O valor de retorno é "retornado" de volta ao "chamador".

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(77);
```

25

OPERADOR ()

Usando o exemplo acima, “toCelsius” refere-se ao objeto da função e “toCelsius()” refere-se ao resultado da função. Acessar uma função sem () retornará o objeto da função em vez do resultado da função.

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius;
```

```
function toCelsius(f) { return (5/9) * (f-32); }
```

ARROW FUNCTIONS

As arrow functions foram introduzidas no ES6, elas permitem uma sintaxe curta para escrever expressões de função. Não é preciso da keyword function, da keyword return e das chaves, porém, só é permitido omitir as keywords return e as chaves se a função for uma única instrução.

```
// ES5
var x = function(x, y) {
  return x * y;
}

// ES6
const x = (x, y) => x * y;
```

OBJECTS

Em JavaScript tudo é um objeto. Bolleans podem ser objetos (se forem definidos com a keyword new). Numbers podem ser objeto (se for definido com a keyword new). Strings podem ser objetos (se forem definidas com a keyword new). Dates são sempre objetos. Maths são sempre objetos. Regexp são sempre objetos. Arrays são sempre objetos. Functions são sempre objetos. Objetos são smpre objetos. Todos os valores, exceto os primitivos, são objetos.

Em JavaScript variáveis podem conter um único valor.

```
let person = "John Doe";
```

Objetos são variáveis também, mas objetos podem conter muitos valores, eles são escritos em pares como **name : value** (nomes e valores separados por dois pontos). É uma prática comum declarar objetos com a keyword const.

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

CRIAÇÃO DE OBJETOS

Com JavaScript, é possível definir e criar objetos de diferentes maneiras:

- Criar um único objeto usando um **object literal.**
- Criar um único objeto com a **keyword new.**
- Definir um **object constructor**, e então criar objetos do tipo construído.
- Criar um objeto usando o método Object.create()

USANDO OBJECT LITERAL

Essa é a maneira mais fácil de criar um objeto em JavaScript. Usando um object literal, é possível definir e criar um objeto em uma única declaração. Um object literal é uma lista de pares de nomes : valores (como age : 50), dentro de chaves { }.

```
const person = {firstName:"John", lastName:"Doe", age:50,
eyeColor:"blue"};
```

USANDO KEYWORD NEW

O seguinte exemplo primeiramente cria um novo objeto vazio usando new object(), e após isso adiciona as 4 propriedades:

```
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 50;
person.eyeColor = "blue";
```

OBJETOS SÃO MUTÁVEIS

Objetos são mutáveis, eles são endereçados por referencia não por valores. O seguinte exemplo explica isso. Se uma pessoa é um objeto, a seguinte declaração deveria criar uma cópia de uma pessoa.

```
const x = person;
```

Porém, o objeto x não é uma cópia de person. Ele é o mesmo objeto que person, se alterações forem feitas no object person, as mesmas alterações ocorreram em x pois eles são o mesmo objeto.

```
const person = {
  firstName:"John",
  lastName:"Doe",
  age:50, eyeColor:"blue"
}

const x = person;
x.age = 10;      // Will change both x.age and person.age
```

OBJECT PROPERTIES

Propriedades (properties) são a parte mais importante nos objetos. Propriedades são os valores associados com os objetos. Um objeto é uma coleção de propriedades não ordenadas. Geralmente, propriedades podem ser alteradas, adicionadas ou deletadas, mas algumas são read only.

As seguintes sintaxes podem ser usadas para acessar uma propriedade de um objeto.

```
objectName.property       // person.age
objectName["property"]    // person["age"]
objectName[expression]    // x = "age"; person[x]
```

É possível adicionar uma propriedade a um objeto existente, simplesmente nomeando ela e lhe dando um valor, conforme o exemplo.

```
person.nationality = "English";
```

É possível deletar uma propriedade simplesmente usando a keyword delete, conforme o exemplo.

```
delete person.age;
```

Os valores em um objeto podem ser outro objeto.

```
myObj = {
  name:"John",
  age:30,
  cars: {
    car1:"Ford",
    car2:"BMW",
    car3:"Fiat"
  }
}
```

Os valores em um objeto também podem ser arrays e valores em arrays podem ser objetos.

```
const myObj = {
  name: "John",
  age: 30,
  cars: [
    {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
    {name:"BMW", models:["320", "X3", "X5"]},
    {name:"Fiat", models:["500", "Panda"]}
  ]
}
```

OBJECT METHODS

Métodos (methods) são ações que podem ser realizadas em objetos. Um método é uma propriedade contendo uma definição de uma função. A seguinte sintaxe é usada para acessar um object methods:

```
objectName.methodName()
```

Se a propriedade for acessada sem parentes, o resultado irá retornar a definição da função.

É possível adicionar um método a um objeto existente, simples mente adicionando uma função como valor de uma propriedade.

```
person.name = function () {
  return this.firstName + " " + this.lastName;
};
```

Existem ainda os **métodos built-in**, são métodos já embutidos no JavaScript que permitem realizar muitas ações diferentes em objetos. O seguinte exemplo usa o método toUpperCase() para converter uma objeto string em um texto maiúsculo.

```
let message = "Hello world!";
let x = message.toUpperCase();
```

HELLO WORLD!

OBJECT CONSTRUCTORS

Criar objetos únicos podem trazer algumas limitações. É possível criar vários objetos do mesmo “tipo” usando uma **object constructor function**. Uma função que cria um “tipo de objeto”. No exemplo a seguir, function Person() é uma object constructor function. Objetos do mesmo tipo podem ser criados “chamando” a função com a keyword new. É considerado uma boa prática nomear funções de construtor com a primeira letra maiúscula.

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

THIS

Em JavaScript, a keyword **this** se refere a um objeto. Ela pode se referir a diferentes objetos dependendo de como ela é usada.

- Em um object methods, **this** refere-se ao objeto.
- Sozinha, **this** refere-se ao global object.
- Em uma função, **this** refere-se ao global object.
- Em uma função, em strict mode, **this** is undefined.
- Em um evento, **this** refere-se ao elemento que está recebendo o evento.
- Métodos como call(), apply() e bind() podem fazer this se referir a qualquer objeto.

THIS EM UM MÉTODO

Quando usado em um object methods, this refere-se ao objeto. No exemplo a seguir, this refere-se ao objeto “person”, já que, fullName(), é um método do objeto **person**.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

Caso seja solicitado a exibição do método fullName, as propriedades em que this faz referencia serão buscadas no objeto “person”.

```
document.getElementById("demo").innerHTML =
person.fullName();
```

John Doe

ALTERANDO CONTEXTO DO THIS

Resumidamente, existem funções especiais como call(), apply() e bind(), que são métodos predefinidos no JavaScript que podem mudar o contexto do this, logo, alterar como ele está fazendo a referência.

No exemplo a seguir o método fullName(), pertence ao objeto person1, porém com o uso dos métodos call() ou apply() esse método pode ser utilizado em outro objeto.

```
const person1 = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}

const person2 = {
  firstName:"John",
  lastName: "Doe",
}

let x = person1.fullName.call(person2);
```

John Doe

A diferença entre os métodos é que call() pega os argumentos separadamente, enquanto apply() pega os argumentos em um array.

THIS PRECEDÊNCIA

Para determinar a qual objeto thisse refere; Use a seguinte ordem de precedência.

Precedence	Object
1	bind()
2	apply() and call()
3	Object method
4	Global scope

E faça as seguintes perguntas:

- Is this in a function being called using bind()?
- Is this in a function is being called using apply()?
- Is this in a function is being called using call()?
- Is this in an object function (method)?
- Is this in a function in the global scope.

DATAS

Datas são objetos criados com o constructor new Date(). Existem 4 maneiras de criar uma data.

```
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
```

new Date()

new Date() cria um novo objeto com a exata hora e data atual.

```
const d = new Date();
document.getElementById("demo").innerHTML = d;
```

Sun Jun 05 2022 20:52:46 GMT-0300 (Horário Padrão de Brasília)

new Date(*year, month, ...*)

Cria um novo objeto data com hora e data especifica. Podem ser especificados 7 números: ano, mês, dia, hora minuto, segundo e milissegundo (nessa exata ordem). Importante estar atento que JavaScript conta meses de 0 a 11 ou seja, Janeiro = 0 e Dezembro = 11 .

```
const d = new Date(2018, 11, 24, 10, 33, 30, 0);
document.getElementById("demo").innerHTML = d;
```

Mon Dec 24 2018 10:33:30 GMT-0200 (Horário de Verão de Brasília)

new Date(*milliseconds*)

O JavaScript armazena valores de datas com milissegundos, a partir de **January 01, 1970, 00:00:00 UTC (Universal Time Coordinated)**. new Date(milliseconds) Cria uma nova dada a partir do "tempo zero" somado com os milissegundos especificados. Ou seja, agora a data é (consulta da data na criação do texto) **Sun Jun 05 2022 20:44:56 GMT-0300 (Horário Padrão de Brasília)** e **1654472696529** milissegundos passados desde 01 de janeiro de 1970.

new Date(*datestring*)

Cria objetos datas a partir de uma data em formato string.

```
const d = new Date("October 13, 2014 11:13:00");
document.getElementById("demo").innerHTML = d;
```

Mon Oct 13 2014 11:13:00 GMT-0300 (Horário Padrão de Brasília)

GET DATE METHODS

Esses são métodos que podem ser usados para obter (getting) informações de um date object:

Method	Description
getFullYear()	Get the **year** as a four digit number (yyyy)
getMonth()	Get the **month** as a number (0-11)
getDate()	Get the **day** as a number (1-31)
getHours()	Get the **hour** (0-23)
getMinutes()	Get the **minute** (0-59)
getSeconds()	Get the **second** (0-59)
getMilliseconds()	Get the **millisecond** (0-999)
getTime()	Get the time (milliseconds since January 1, 1970)
getDay()	Get the weekday as a number (0-6)
Date.now()	Get the time. ECMAScript 5.

```
const d = new Date();
document.getElementById("demo").innerHTML = d.getFullYear();
```

2022

SET DATE METHODS

Os métodos Set Date permitem definir valores de data (anos, meses, dias, horas, minutos, segundos, milissegundos) para um objeto de data. Esses são os métodos Set Date usados para definir uma parte de uma data:

Method	Description
setDate()	Set the day as a number (1-31)
setFullYear()	Set the year (optionally month and day)
setHours()	Set the hour (0-23)
setMilliseconds()	Set the milliseconds (0-999)
setMinutes()	Set the minutes (0-59)
setMonth()	Set the month (0-11)
setSeconds()	Set the seconds (0-59)
setTime()	Set the time (milliseconds since January 1, 1970)

```
const d = new Date();
d.setFullYear(2020);
document.getElementById("demo").innerHTML = d;
```

Fri Jun 05 2020 21:14:43 GMT-0300 (Horário Padrão de Brasília)