



Desenvolvimento Web&Mobile

Desenvolvimento Web para Server Side

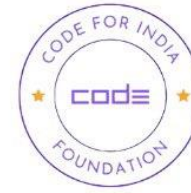
Sara Monteiro

sara.monteiro.prt@msft.cesae.pt

O que é o JS e para que serve

- É uma linguagem de script com base na linguagem de programação ECMAScript.
- É o padrão em linguagem “*client side*” para qualquer *browser*.
- Dá movimento ao nosso site.

SIMPLE STRUCTURE OF A **WEBSITE**

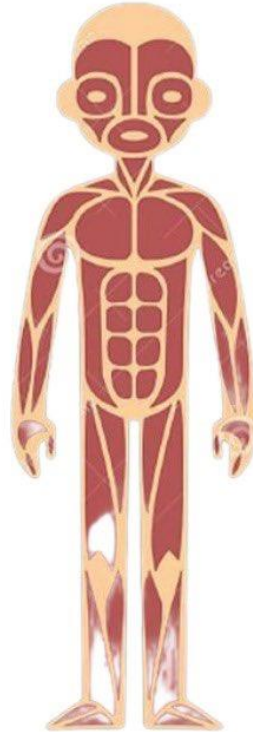


cesae
digital

Centro para o Desenvolvimento
de Competências Digitais



HTML
(Structural)



JAVASCRIPT
(Behavioral)



CSS
(Presentation)

O que é o JS e para que serve

- Validação de formulários
- Galerias de fotos
- Janelas de aviso
- *Banners*
- Janelas de publicidade
- Menus pop-up
- Animações
- ...

127.0.0.1:5500 says

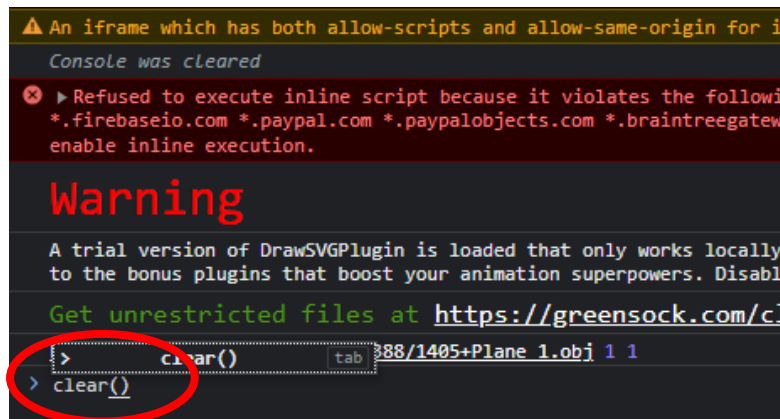
Hello World

OK

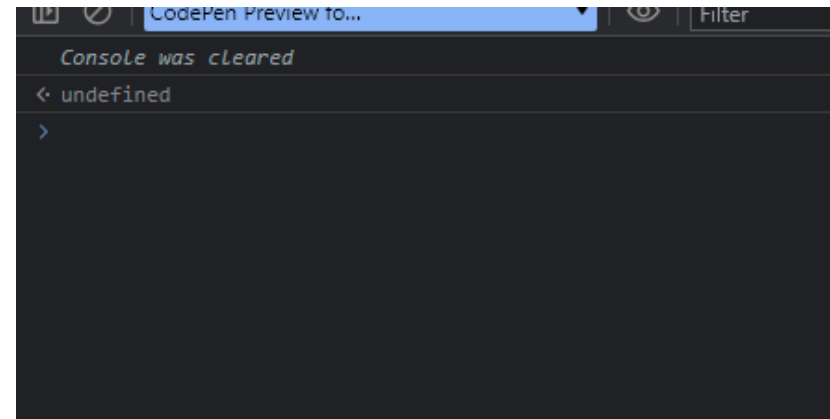
[Exemplo de Animação JS](#)

Usar JS

- Num qualquer site, clicar com o botão direito do rato e seleccionar "Inspecionar"
- Seleccionar o separador "Console"
- Caso haja algum erro, correr o comando `clear()`, seguido de enter.

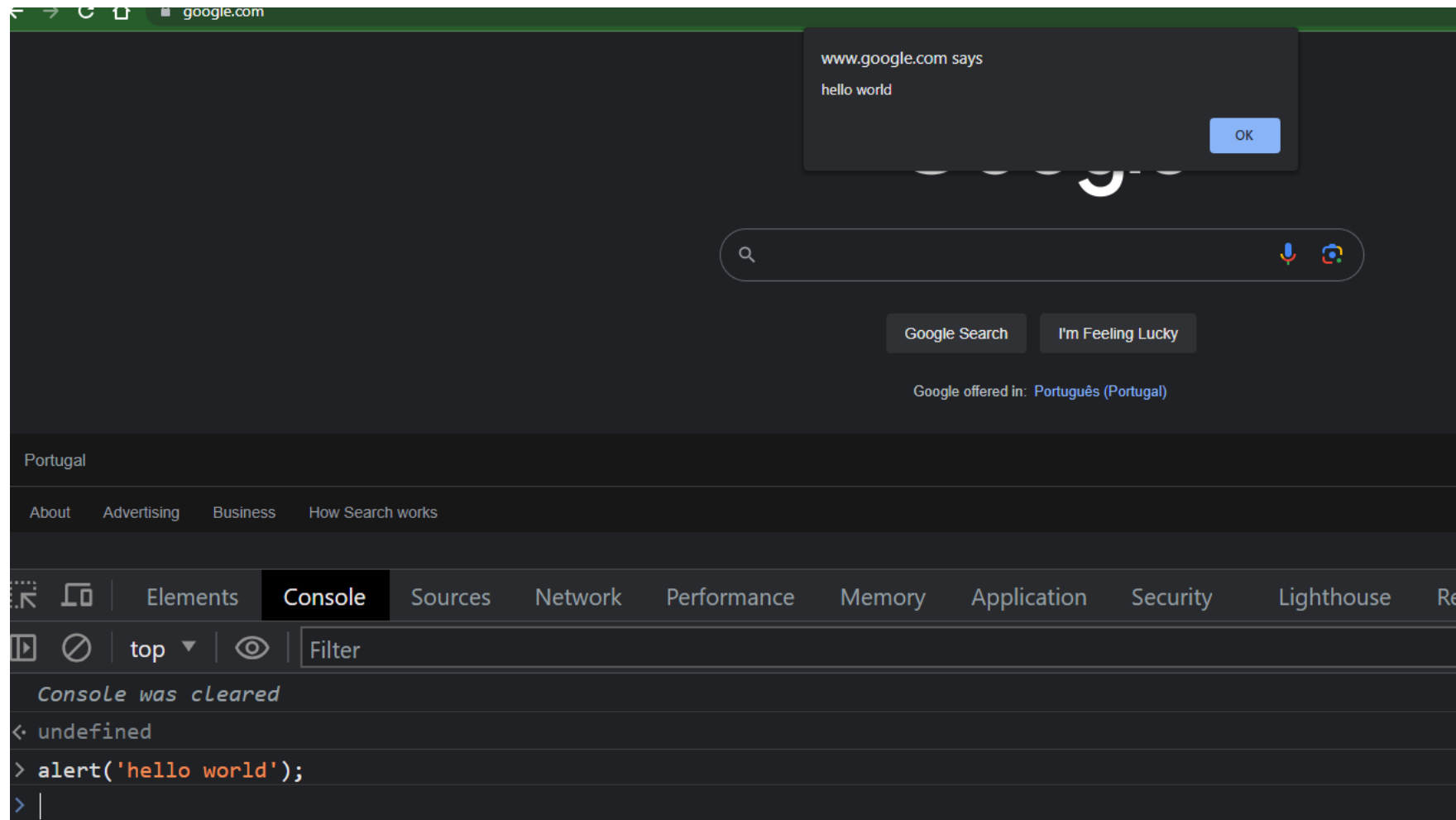


A screenshot of a browser's developer console. At the top, a yellow warning bar says "An iframe which has both allow-scripts and allow-same-origin for i". Below it, a message says "Console was cleared". Then, a red error message appears: "Refused to execute inline script because it violates the following Content Security Policy (CSP) directive: 'script-src *.firebaseio.com *.paypal.com *.paypalobjects.com *.braintreegateway.com'. Note that 'script-src-elem' is ignored by the browser. To see more information, open the console in the context of the site." Below the error, a large red "Warning" text is displayed. Underneath, a message states: "A trial version of DrawSVGPlugin is loaded that only works locally to the bonus plugins that boost your animation superpowers. Disable to get the full version." At the bottom, a green link says "Get unrestricted files at https://greensock.com/club". In the console input area, the command `clear()` has been typed and is circled in red.



A screenshot of a browser's developer console. At the top, a message says "Console was cleared". Below it, the output of the `clear()` command is shown as `< undefined`. The console input area is empty.

Usar JS



Adicionar JS ao nosso Projecto

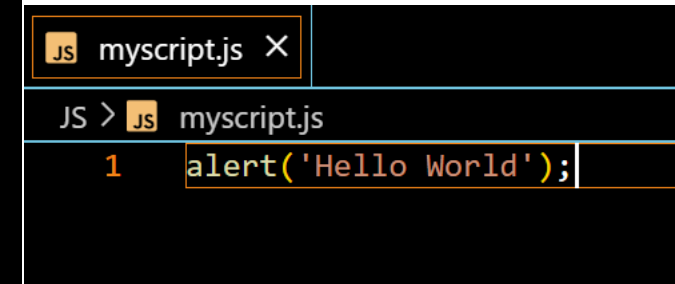
- JS Interno:

```
<script>  
    // JavaScript goes here  
</script>
```

```
<body></body>  
<script>  
    alert("Hello World");  
</script>
```

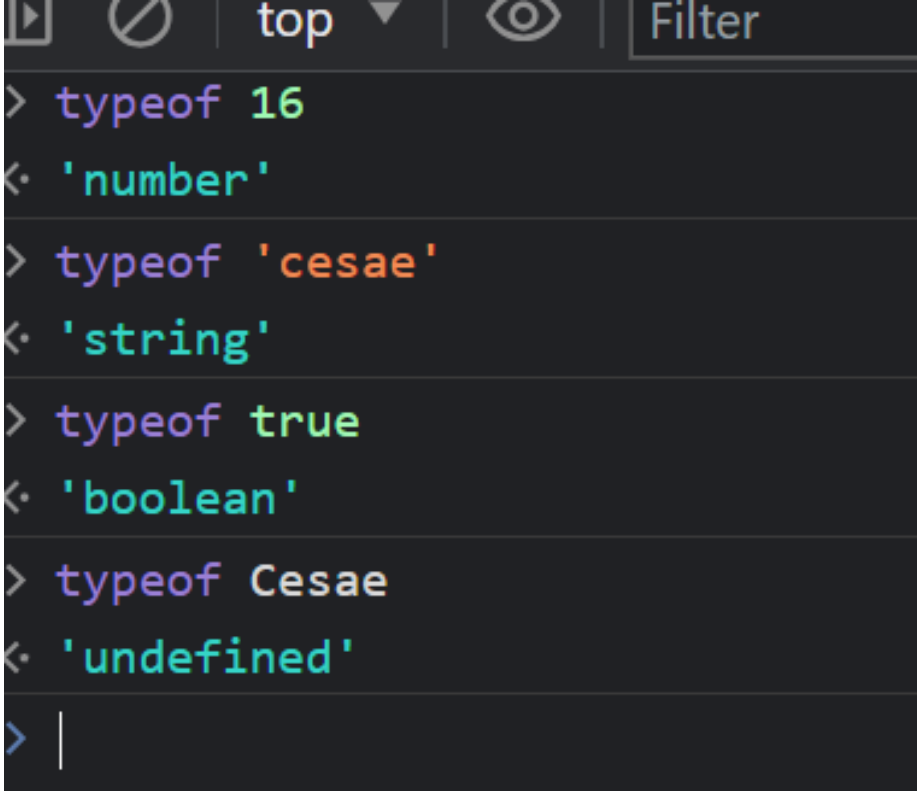
- JS Externo:

```
<html lang="en">  
  <head>  
    <meta charset="UTF-8" />  
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <title>Document</title>  
    <script src="JS/myscript.js" defer></script>  
  </head>  
<body></body>
```



Tipos de Dados Primitivos

- Number: 1; 5,6
- String: "Aula JS"
- Boolean: true, false
- Null
- Undefined

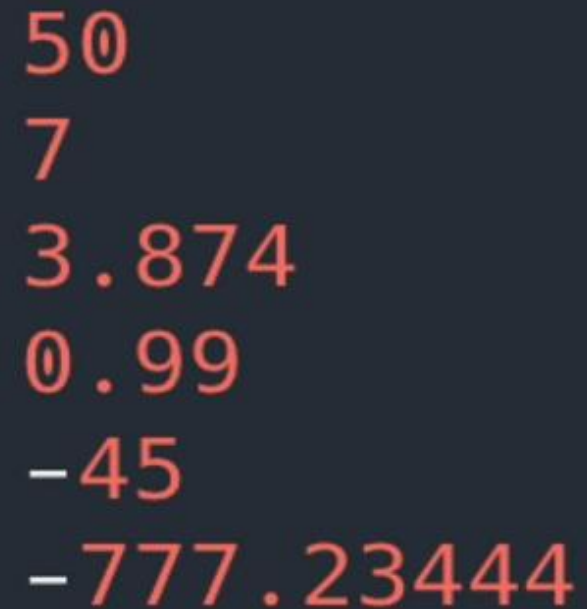


```
top Filter
> typeof 16
< 'number'
> typeof 'cesae'
< 'string'
> typeof true
< 'boolean'
> typeof Cesae
< 'undefined'
> |
```


Numbers

Em JS tem um tipo número, que representa:

- Números Positivos
- Números Negativos
- Números inteiros (integers)
- Números decimais



50
7
3.874
0.99
-45
-777.23444

Numbers: Operações Aritméticas

Arithmetic Operators

[Documentação](#)

```
> 1+2
< 3
> 5/2
< 2.5
> 5%2
< 1
```

Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get remainder in integer division	5%2	1
++	Increment	A = 10; A++	11
--	Decrement	A = 10; A--	9

Numbers:

Operações Comparação

```
> 5>0  
↳ true  
> 10<2  
↳ false  
> 10 == 10  
↳ true  
> |
```

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
==	Equal to
!=	Not equal to
===	Equal value and type
!==	Not equal value or equal type

Variáveis

“Variável é o nome utilizado para definir um ou mais valores que são manipulados pelos programas durante a sua operação. O nome “variável” é utilizado por ser um tipo de conteúdo que pode apresentar diferentes valores enquanto o sistema está em execução. Tudo dependerá do comando do usuário e o tipo de operação que é realizado.”

- in fonte

Variáveis

Em JS, as variáveis são declaradas da seguinte forma:

let someName = value;

```
> let myLuckyNumber = 16;  
< undefined  
  
> let myName = 'Sara';  
< undefined  
  
> myLuckyNumber  
< 16  
  
> myName  
< 'Sara'  
  
> |
```

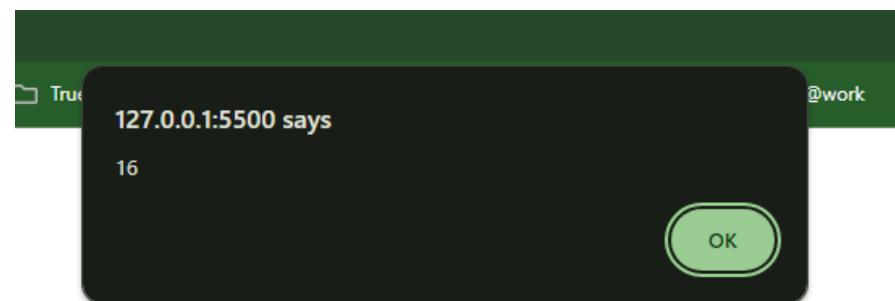
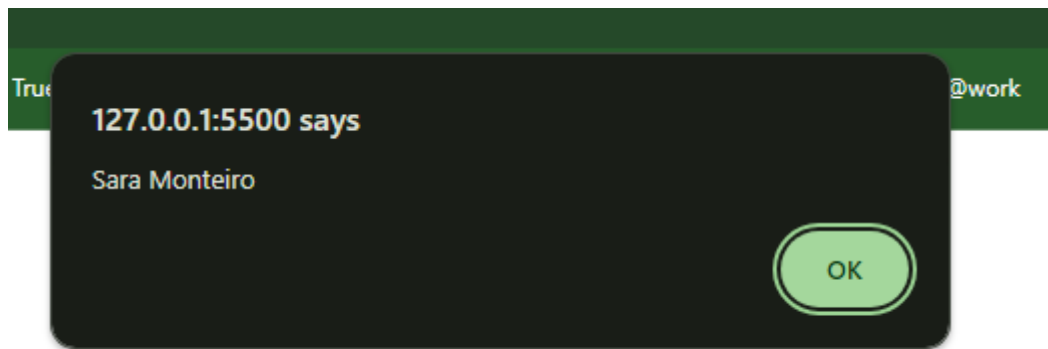
Variáveis

No ficheiro JS, declaramos da mesma forma mas quando as queremos chamar temos que indicar a forma como elas vão aparecer visualmente no browser.

Para já usaremos o alert();

```
let myFirstName = 'Sara';  
let mySecondName = 'Monteiro';  
  
let number1 = 6;  
let number2= 10;  
  
alert(myFirstName + ' ' + mySecondName);  
alert(number1 + number2);
```

Variáveis



Exercício



a) Criar uma pasta para os exercícios de JS conforme as indicações aprendidas.
(Index na raíz, pasta para JS, etc)

b) No ficheiro index.js:

- definir uma variável chamada myLuckyNumber e atribuir um valor numérico;
- definir uma variável chamada myName atribuir o nome;
- chamar a função JS alert(myName + ' ' + myLuckyNumer);

c) Associar o documento ao nosso index.html e abrir a página

Actualizar Variáveis

- Como nas outras linguagens de programação, em JS as variáveis podem ser actualizadas ao longo do código conforme vamos necessitando delas.
- Para números podemos também actualizar usando o incremento `var++` ou decremento `var--` que significa adicionar ou retirar uma unidade à variável.

```
> let myProgress = '50%';  
< undefined  
> myProgress  
< '50%'  
> myProgress = '75%';  
< '75%'  
> myProgress  
< '75%'  
>
```

```
> let myNumber = 16;  
< undefined  
> myNumber++;  
< 16  
> myNumber;  
< 17  
> myNumber--;  
< 17  
> myNumber;  
< 16
```

Variável / Constante

Constante: usa-se como uma variável, excepto que não lhe podemos mudar o valor.

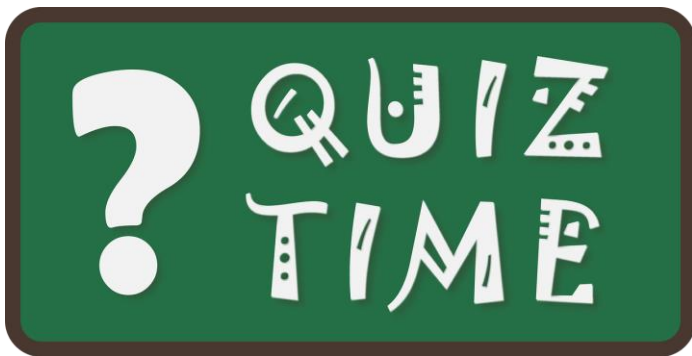
Declara-se da seguinte forma:

```
CONST MYNAME = 'Sara';
```

Boolean / mudança de tipos

- Um boolean é uma variável que assume o tipo true ou false
Ex: let jsIsCool = true;
- Em Js a mesma variável pode mudar de tipo conforme o que lhe vamos assignando.

```
> let myAge = 37;  
< undefined  
  
> typeof myAge  
< 'number'  
  
> myAge = 'Mudará no meu aniversário';  
< 'Mudará no meu aniversário'  
  
> typeof myAge  
< 'string'
```



```
let age = 37;  
let year = '2023';  
let sucess = true;  
let fail = 'false';  
Qual é o tipo de dados?
```

```
const age = 65;  
age++;  
Qual o valor da age?
```

```
let score = 5;  
score +4;  
Qual o valor do score?
```

```
let bankBalance = 100;  
bankBalance /= 2;  
bankBalance +=10;  
Qual o valor do bankBalance?
```

Boas práticas na criação de variáveis

- Não começar com números ou símbolos
- Usar camel case: myFirstName
- Usar nomes significativos.
Ex: let number = 1 em vez de let n = 1;

Strings

- Representam texto (um conjunto de caracteres)
- Usar sempre entre aspas
- As strings têm um index



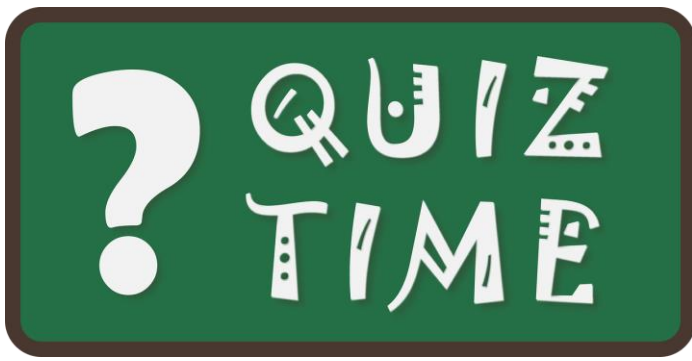
```
> let nameOfCourse = 'Front-end Developer';  
< undefined  
> nameOfCourse[3];  
< 'n'
```

Strings – Tamanho

- `length` = dá-nos o tamanho da nossa string, ou seja, o total de caracteres (incluindo espaços)
- Concatenar: podemos juntar duas variáveis através do `+`.

```
> nameOfCourse.length;  
< 19
```

```
> let firstName = 'Sara';  
< undefined  
  
> let secondName = 'Monteiro';  
< undefined  
  
> firstName + ' ' + secondName;  
< 'Sara Monteiro'
```



```
const animal = 'hippopotamus';  
animal[7]?
```

```
const city = "Kyoto";  
const country = 'Japan';  
const combo = city + country;  
combo?
```

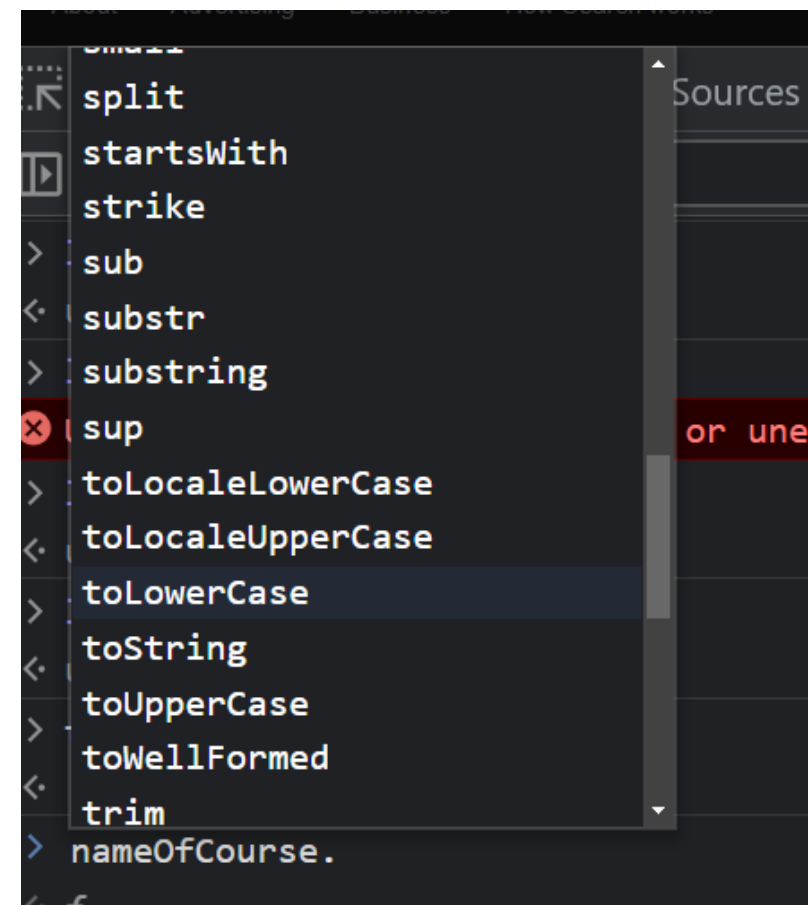
```
let year = '1998';  
year = year + 1;  
year?
```


Strings: Métodos

Métodos são acções incorporadas do JS que podemos utilizar nas strings.

Na consola, se escrevermos a nossa variável e um . aparecem todos os métodos disponíveis.

[Documentação](#)

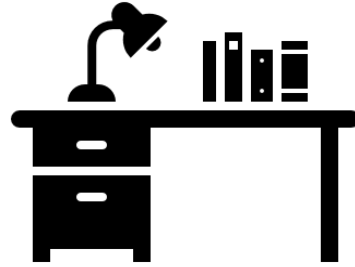


Strings: Métodos

Os métodos utilizam-se da seguinte forma:
nomeDaVar.método();

```
< 'JS Front-end Developer'
> let nameOfCourse = 'JS Front-end Developer';
< undefined
> nameOfCourse.toUpperCase()
< 'JS FRONT-END DEVELOPER'
> nameOfCourse = '    JS Pró    ';
< '    JS Pró    '
> nameOfCourse.trim()
< 'JS Pró'
> nameOfCourse = '    JS Pró    ';
< '    JS Pró    '
> nameOfCourse.trim().toUpperCase();
< 'JS PRÓ'
```

Exercício



1. No teu projecto, no ficheiro JS, cria a seguinte constante
`const message = " TASTE THE RAINBOW! ";`
2. Sem alterar a variável, define uma variável chamada `whisper` que é a versão em letra pequena e sem espaços no final e no início da `message`.
3. Faz `console.log(whisper)` para verificar se está correcto.

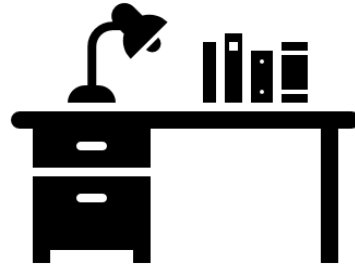
Strings: Métodos c/argumentos

Alguns métodos aceitam argumentos que modificam o seu comportamento. Estes argumentos são passados dentro dos ().

```
myName.replace('Sara', 'Rita');  
< 'Rita Monteiro'  
> |
```

```
> let myName = 'Sara Monteiro';  
< undefined  
  
> myName.indexOf('S');  
< 0  
  
> myName.indexOf('M');  
< 5  
  
> myName.indexOf('l');  
< -1  
  
> myName.slice(5,10);  
< 'Monte'
```

Exercício



1. No teu projecto, no ficheiro JS, cria a seguinte constante `const word = "skateboard"`;
2. Sem alterar a variável, usa um método que retire a parte 'board' da const.
3. Usando o método `replace`, substitua o 'o' por e.
4. Guarde o resultado numa variável chamada `facialHair`.
5. Faz `console.log(facialHair)` para verificar se está correcto.

String Template Literals

- É uma forma de, dentro de uma string, calcularmos e apresentarmos algo automaticamente.
- As strings template têm que ser usadas entre `` ``.
- São úteis para simplificar e não termos que concatenar constantemente variáveis.

```
> myStringTemplate = `A soma de 4 com 6 é ${4+6}`;  
< 'A soma de 4 com 6 é 10'
```

Null e Undefined

Undefined: variável desconhecida.
“não sei o que é isso”.

Null: não existe, está vazio.
“essa variável não tem nada”.

```
> 'Bytes'[11];  
← undefined  
  
> let x;  
← undefined  
  
> |
```

```
> let x = null;  
← undefined  
  
> x  
← null  
  
> |
```

Math Object

Conjunto de propriedades e métodos matemáticos. Usa-se evocando o Math.método().

```
> Math.PI;  
< 3.141592653589793  
  
> Math.round(3.566);  
< 4  
  
> Math.abs(-16);  
< 16
```

[Documentação](#)

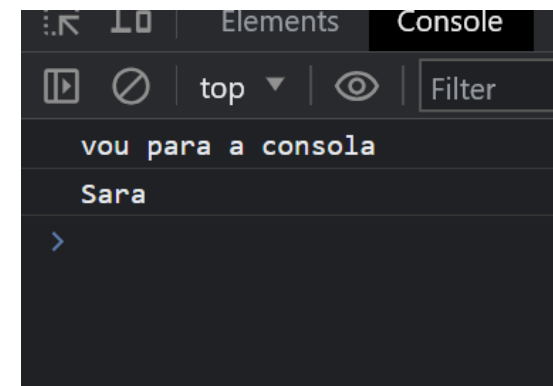
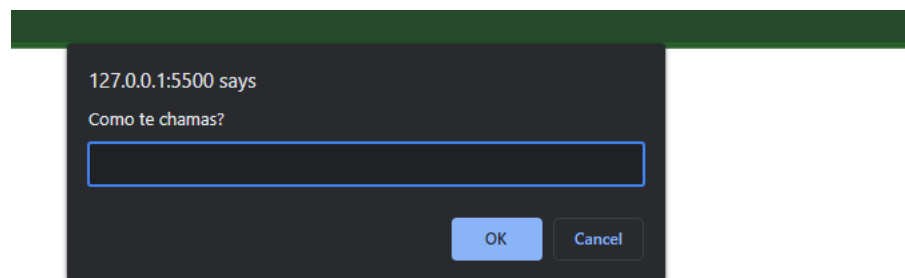
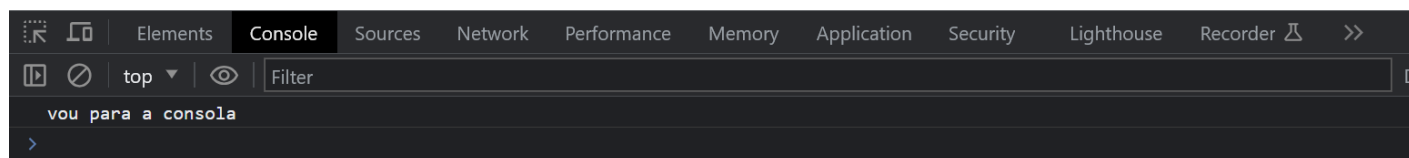
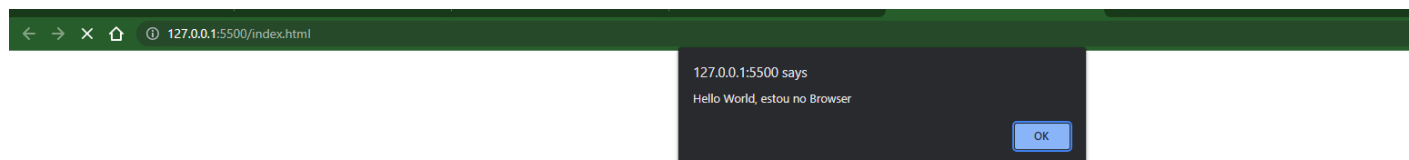
Console.log, Alert e Prompt

A partir de agora iremos passar da consola para o nosso script no projecto, e para isso será útil saber formas de interagir com o browser.

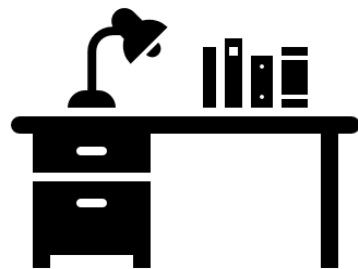
- `console.log()` -> imprime os argumentos na Consola. Podemos também usar o `console.error()`, `console.warn`, etc.
- `alert()`-> imprime um popup.
- `prompt()`-> abre numa caixa que possibilita o user colocar dados. Podemos pegar nesses dados e assignar a uma variável.
Estes dados são recebidos sempre como string.

```
console.log('vou para a consola');  
  
alert('Hello World, estou no Browser');  
  
let myName = prompt('Como te chamas?');  
  
console.log(myName);
```

Console.log, Alert e Prompt



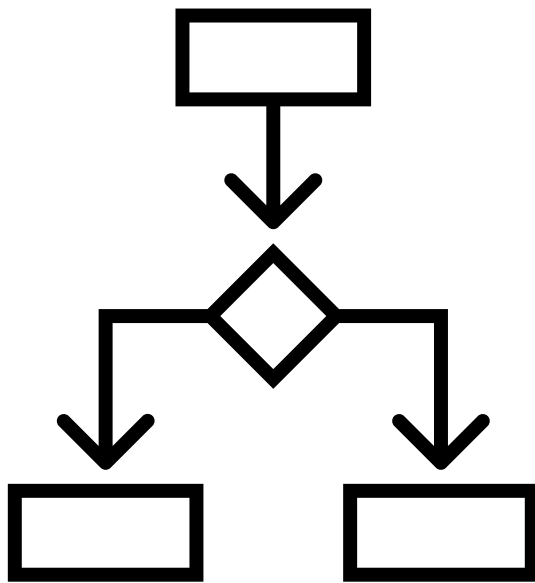
Exercício



Pergunta ao utilizador através do Prompt o primeiro e o último nome (em duas caixas diferentes) e coloque os mesmos em variáveis.

Apresenta através de um alert('Olá', nomeDaPessoa);

Decision Making



Processo de tomar decisões de acordo com os inputs do User, interactividade Browser /utilizador.

Exemplos:

- Tratar pelo nome, caso o utilizador tenha colocado um.
- input que pede o nome e idade do utilizador. Se idade > 30, trata por Senhor/ Senhora, se menor por menino/menina.
- Mostrar comentários no Instragram caso haja

....

Condicionais

```
const ageOfConsent = 18;
let userAge = parseInt(prompt('Que idade tens?'));

if(userAge >17){
  alert('ok');
}else{
  alert('Não tem idade para aceder a este conteúdo');
}
```

[Documentação](#)

```
let userColor = prompt('Qual é a tua cor preferida?');
switch (userColor) {
  case 'Verde':
    alert('Verde');
    break;
  case 'Vermelho':
    alert('Vermelho');
    break;
  case 'Preto':
    alert('Preto');
    break;
  default:
    alert('Ups, não temos essa cor');
}
```

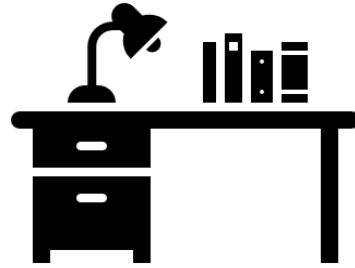
Decision Making : Switch Case

```
let dayOfWeek = prompt('Que dia da semana é hoje');

switch (dayOfWeek) {
  case 'Segunda':
    document.write('<h2>Ehrr, mais uma semana.</h2>');
    break;
  case 'Quarta':
    document.write('<h2>Já vamos a meio, yay.</h2>');
    break;
  case 'Sexta':
    document.write('<h2>Sextooooou!</h2>');
    break;
  case 'Sábado':
    document.write('<h2>yupppi, dia de festa.</h2>');
    break;
  default:
    document.write('<h2>Desculpa, desconheço esse dia.</h2>');
}
```

[Documentação](#)

Exercício



1. Pergunta ao utilizador através do Prompt que dia da semana é e coloca numa variável. Se a resposta for Sexta, apresenta o texto 'yay, sobrevivemos a mais uma semana'. Caso contrário, coloca uma mensagem à tua escolha.
2. Através do prompt, peça ao utilizador uma password com pelo menos 6 caracteres.. Caso a mesma não cumpra as condições, informe o mesmo, dizendo que a pass é inválida.
3. Repita a alínea 1 usando um Switch / Case e personalizando com várias opções.

Arrays

- Colecção de dados ordenada.
Ex: array de dias da semana, de músicas numa playlist.
- Declaração: `let daysOfWeek = [];`
- Tal como as strings, também têm um index e um `length`.

```
> let daysOfWeek = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta'];  
← undefined  
> daysOfWeek[2];  
← 'Quarta'  
> daysOfWeek.length;  
← 5  
> |
```

[Documentação](#)

Modificar arrays

Através do index podemos reassignar um valor, adicionar, etc..

Nas strings não podemos fazer estas operações!

```
> daysOfWeek[2] = 'Domingo';  
↵ 'Domingo'  
  
> daysOfWeek[6] = 'Sábado';  
↵ 'Sábado'  
  
> daysOfWeek;  
↵ ▶ (7) ['Segunda', 'Terça', 'Domingo', 'Quinta', 'Sexta', empty, 'Sábado']  
  
> |
```

Arrays: Métodos

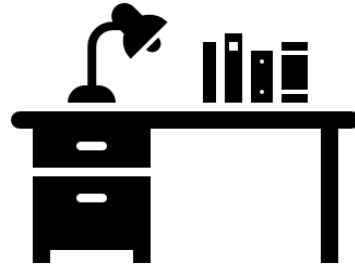
Da mesma forma que ocorria com as strings, temos uma interminável lista de métodos de Arrays em [documentação](#).

Para modificar arrays:

- Push: adiciona ao fim
- Pop: remove do fim
- Shift: remove do início
- Unshift: adiciona ao Início

```
> let daysOfWeek = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta'];  
< undefined  
  
> daysOfWeek.push('Sábado');  
< 6  
  
> daysOfWeek;  
< ▶ (6) ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado']  
  
> daysOfWeek.pop();  
< 'Sábado'  
  
> daysOfWeek;  
< ▶ (5) ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta']  
  
>
```

Exercício



1. Cria um array de planetas: Mercúrio, Vénus, Terra, Mart, Jupiter, Saturno, Urano, Neptuno, Plutão.
2. Mart foi escrito mal. Substitui pelo nome correcto, Marte.
3. Ups, o planeta Plutão foi removido do Sistema Solar.. Utilizando os métodos dos arrays, remove o planeta da lista.
4. Descobriste um novo planeta, yaaay. Inventa um nome e adiciona ao início do array.
5. Faz `console.log()` do array e verifica se está certo.

Arrays: Outros Métodos

- Concat: junta arrays;
- Includes: procura um valor;
- IndexOf (igual ao das strings);
- Join: transforma um array numa string;
- Reverse: muda a ordem para a inversa.
- Slice: copia / retira uma parte.
- Splice: remove ou substitui elementos.
- Sort: ordena um array

Arrays: Outros Métodos

```
> const array1 = ['a', 'b', 'c'];  
< undefined  
  
> const array2 = ['f', 'g', 'h'];  
< undefined  
  
> const array3 = array1.concat(array2);  
< undefined  
  
> array3  
< ► (6) ['a', 'b', 'c', 'f', 'g', 'h']  
> |
```

```
> array3.includes('c');  
< true  
  
> array3.includes('z');  
< false  
> |
```

```
> array3.indexOf('c');  
< 2  
> |
```

```
> array3.reverse();  
< ► (6) ['h', 'g', 'f', 'c', 'b', 'a']  
> |
```

```
> array3.slice(3);  
< ► (3) ['f', 'g', 'h']  
> |
```

```
> array3.splice(3,0,'d');  
< ► []  
  
> array3;  
< ► (7) ['a', 'b', 'c', 'd', 'f', 'g', 'h']  
> |
```

```
> array3.splice(6,1,'i');  
< ► ['h']  
  
> array3;  
< ► (7) ['a', 'b', 'c', 'd', 'f', 'g', 'i']  
> |
```

Nested Arrays

```
const colors = [
  ['red', 'crimson'],
  ['orange', 'dark orange'],
  ['yellow', 'golden rod'],
  ['green', 'olive'],
  ['blue', 'navy blue'],
  ['purple', 'orchid']
]
```

Em JS podemos guardar arrays dentro de arrays.

Para aceder a um item dentro de um array multidimensional temos que fazer uma combinação de indexes.

Por exemplo para aceder a olive:
`colors[3][1]`

```
> const daysOfWeek = [ ['Quarta Manhã', 'Quarta Tarde'], ['Quinta Manhã', 'Quinta Tarde'], ['Sexta Manhã', 'Sexta Tarde'] ];
```

```
< undefined
```

```
> daysOfWeek[1][0];
```

```
< 'Quinta Manhã'
```

```
>
```

Exercício



1. No vosso ficheiro JS criem o seguinte array multidimensional com os lugares de um avião: (podem encontrar o código na pasta exs do projecto do git).

```
const airplaneSeats = [  
  ['Ruth', 'Anthony', 'Stevie'],  
  ['Amelia', 'Pedro', 'Maya'],  
  ['Xavier', 'Ananya', 'Luis'],  
  ['Luke', null, 'Deniz'],  
  ['Rin', 'Sakura', 'Francisco']  
];
```

2. No sítio onde existe um lugar vago, assignem o vosso nome. Façam `console.log()` do novo array para confirmar se vos foi atribuído o lugar.

Object Literals

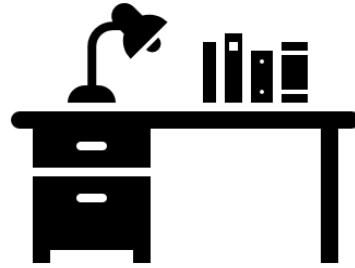
- Estrutura de Dados;
- Coleções de propriedades com valores;
- É como um array, mas damos etiquetas aos valores: key->value pair.
- Para aceder a dados usamos a key customizada em vez do index.

```
> let userData = { name:'Sara', age:36, address:'São João da Madeira', job:'Web Developer'};
< undefined
> userData.age;
< 36
> userData.job;
< 'Web Developer'
```


Object Literals

```
> const course = {name: 'FE developer', hours: 900, modules: ['mySql', 'EWeb', 'DC']}  
< undefined  
  
> course.name  
< 'FE developer'  
  
> course.modules[1];  
< 'EWeb'  
  
> course.hours = 950;  
< 950  
  
> course  
< ▶ {name: 'FE developer', hours: 950, modules: Array(3)}  
  
> |
```

Exercício



1. Cria um objecto chamado product com as seguintes propriedades:
 - nome e assigna-lhe um nome à tua escolha.
 - inStock, como verdadeiro.
 - Price com o valor 1.99
 - Colors, com um array de vermelho, azul e verde;
2. Imprime na consola os seguintes valores:
 - Price
 - Cor Verde
3. Ups, a inflacção chegou à nossa loja. Muda o valor do produto para 2.55.

Combinações de Arrays e Objectos

```
const student = {  
  firstName: 'David',  
  lastName: 'Jones',  
  strengths: ['Music', 'Art'],  
  exams: {  
    midterm: 92,  
    final: 88  
  }  
}
```

Loops

Ciclos de Repetições. Ex: escrever 10x 'Hello World', escrever os números de 1 a 10; Comentários numa publicação.

- For loop
- While loop
- For .. of loop
- For .. in loop

[Documentação](#)

Loops - For Loop

```
for ([expressaoInicial]; [condicao]; [incremento])  
  declaracao
```

For Loop que escreve os
números de 1 a 10:

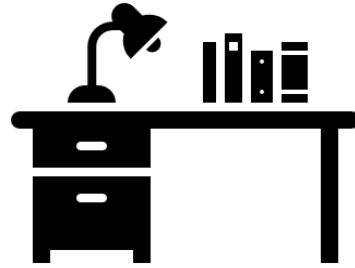
```
for(let i = 1; i<11; i++){  
  console.log(i);  
}
```

Loops - Exemplos

- Imprimir os números pares até 20.

```
for (let i = 0; i <= 20; i += 2) {  
  console.log(i)  
}
```

Exercício



1. Lembra-se da música I'm blue dos Eiffel 65? Vamos usar o ciclo for para escrever 6 vezes o "**Da ba dee da ba daa**";
2. Faça um ciclo for que imprima estes números, por esta ordem:
 - 25
 - 20
 - 15
 - 10
 - 5

Loops - A problemática Dos Loops Infinitos

- Ter em atenção sempre que os loops têm que ter um final, caso contrário vão ocupar a memória toda e bloquear o nosso programa.

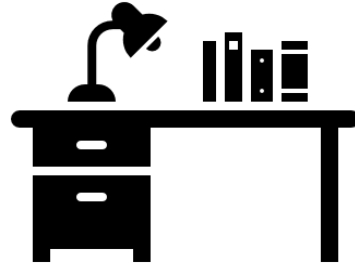
```
//DO NOT RUN THIS CODE!  
for (let i = 20; i >= 0; i++) {  
    console.log(i);  
} //BADD!!!
```


Loops - Iterar arrays

Para fazer um loop a um array começa no index 0 e continua até ao último index (length -1)

```
> let daysOfWeek = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta'];  
< undefined  
> for(let i = 0; i < daysOfWeek.length; i++){  
    console.log(i, daysOfWeek[i]);  
}  
0 'Segunda'  
1 'Terça'  
2 'Quarta'  
3 'Quinta'  
4 'Sexta'
```

Exercício



1. Declare o seguinte array:
`const people = ["Scooby", "Velma", "Daphne", "Shaggy", "Fred"];`
2. Imprima na página os nomes em letra maiúscula.

Loops: o While

```
while (condicao)
    declaracao
```

Corre enquanto a condição se verifica.

Atenção aos loops infinitos!!

```
let mySecretCode = 'jsrocks';
let userCode = prompt('Digite o código secreto');

while(userCode != mySecretCode){
    userCode = prompt('Digite o código secreto');
}

alert('Yay, código ok');
```

```
> while(num < 10){
  console.log(num);
  num++;
}
```

1

2

3

4

5

6

7

8

9

Loops: for..in e for..of

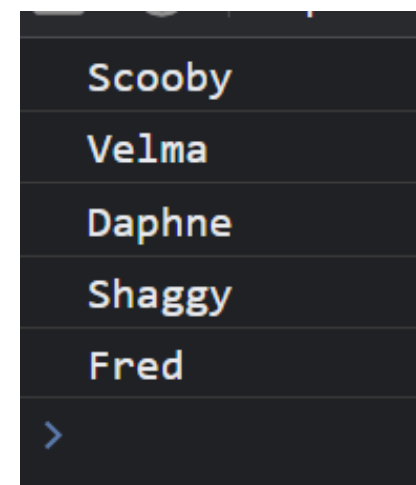
- Não suportados no IE, novos no JS.
- Enquanto o for...in interage com o nome das propriedades, o for...of interage com o valor das propriedades.

```
for (variavel of objeto) {  
    declaracoes  
}
```

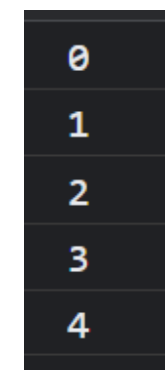
```
for (variavel in objeto) {  
    declaracoes  
}
```

Loops: for..in e for..of

```
const people = ["Scooby", "Velma", "Daphne", "Shaggy", "Fred"];  
  
for(let element of people){  
  console.log(element);  
}
```



```
for(let element in people){  
  console.log(element);  
}
```



Exercício



1. Utilizando os loops e as condições que já aprendemos, construa uma lista de compras.

Dicas:

- A lista é um array.
- Fazer prompt para o utilizador adicionar items na lista, e adicionar esses items ao array.
- Criar uma variável "código" e se o utilizador digitar isso, o ciclo acaba e a lista aparece. Por exemplo, se digitar 'fim', deixam de aparecer prompts.

Funções

- Blocos de código que contêm procedimentos que podemos reutilizar.
- Estes blocos podem ser associados a um elemento HTML e só ser activados mediante certa acção do user. (exemplo: botões)

O meu nome é Sara!

Adivinhe o código secreto Digite os números pares até 20 Hello World

```
<button class="btn btn-info" onclick="secretCode()">Adivinhe o código secreto</button>
<button class="btn btn-warning" onclick="numbers()">Digite os números pares até 20</button>
<button class="btn btn-success" onclick="hello()">Hello World</button>
```

```
function secretCode(){
  let secretCode = 'bytes';
  let userCode = prompt('Digite o código');

  while(userCode != secretCode){
    userCode = prompt('Digite o código');
  }
}

function numbers(){
  for( let i = 0 ; i<21; i+=2){
    document.write(i + '<br>');
  }
}

function helloWorld(){
}
```

Funções

É preciso dois passos para escrever uma função:

1. Definir a função
2. Executar a função

```
function helloWorld(){  
  let hello = 'Hello World';  
  alert(hello);  
}  
  
helloWorld();
```

Documentação

Exercício



1. Escreva uma função chamada `printHeart` que envie um alerta com um coração (<3);
2. Crie um botão bootstrap danger e execute a função ao clicar.

Funções - argumentos

- Neste momento as nossas funções aceitam zero argumentos, ou seja, elas funcionam sempre da mesma forma.
- As funções aceitam inputs, chamados argumentos.

Exemplo de método sem argumentos

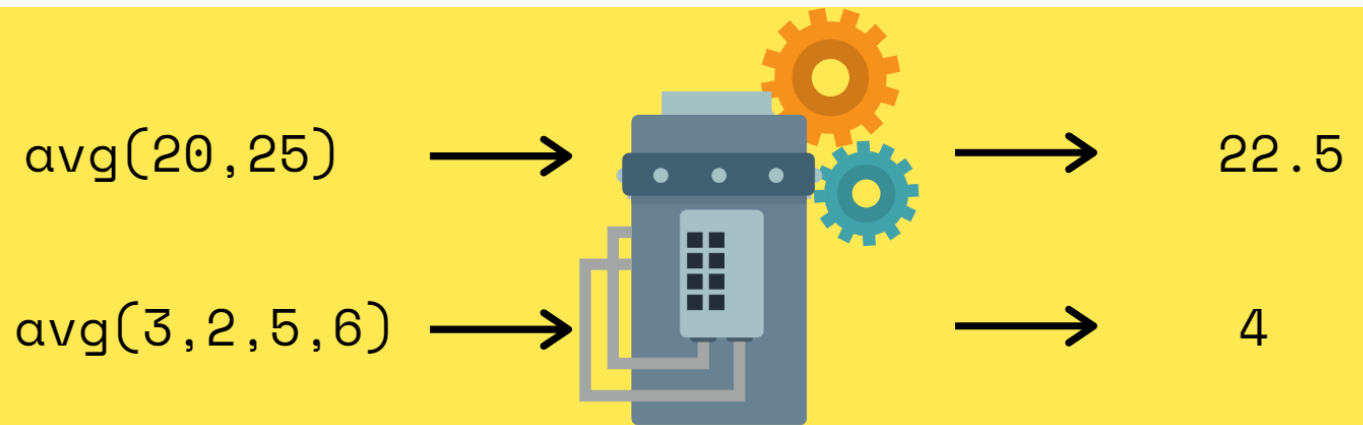


```
//No input  
"hello".toUpperCase( );
```

Funções - argumentos

Exemplo de método com argumentos:
dependendo do que lhe enviamos nos
argumento, o output é distinto.

```
//Different inputs...  
"hello".indexOf('h'); //0  
//Different outputs...  
"hello".indexOf('o'); //4
```



Funções - argumentos

```
function helloWorld(hello){  
  alert(hello);  
}
```

```
helloWorld('hello World');  
helloWorld('cucu');
```

Exercício



1. Escreva uma função chamada rant que aceite um argumento chamado message.
2. A função deverá ser associada a um botão que, quando clicado, deverá imprimir o que o utilizador quiser definir como mensagem 3 vezes em letra maiúscula.

Ex:

```
<button class="btn btn-danger" onclick="rant('odeio CSS')">Função Rant.</button>
```

```
1 'ODEIO CSS'
2 'ODEIO CSS'
3 'ODEIO CSS'
>
```

Funções c/ vários Argumentos

As funções podem ser definidas para aceitar mais que um argumento.
Ex: primeiro e último nome.

```
function greet(firstName, lastName){  
  console.log(`Hey, ${firstName} ${lastName[0]}.`);  
}  
  
greet('Sara', 'Monteiro');
```

Hey, Sara M.

Funções c/ vários Argumentos

```
✓ function repeat(msg, numTimes ){  
✓   for(i=0; i<numTimes; i++){  
   console.log(i, msg);  
   }  
}
```

```
> repeat('A turma de FE é incrível', 5);  
0 'A turma de FE é incrível'  
1 'A turma de FE é incrível'  
2 'A turma de FE é incrível'  
3 'A turma de FE é incrível'  
4 'A turma de FE é incrível'
```

Exercício



1. Vamos jogar um jogo chamado 'olhos de cobra'. Escreva uma função chamada `isSnakeEyes` que aceite dois números como inputs.
2. Se ambos os números forem 1, escreva uma mensagem a dizer: 'Yay, snake eyes', se não 'Não são olhos de cobra':

Ex:

```
> isSnakeEyes(1,5)
Não são olhos de cobra
< undefined
> isSnakeEyes(1,1)
Yay, snake eyes
< undefined
```


Funções - return

- O return tira o valor executado dentro da função para fora dela.
- O return para a execução da função, nada do que está depois é executado.

```
function sum(num1, num2){  
  console.log(num1+num2);  
}
```



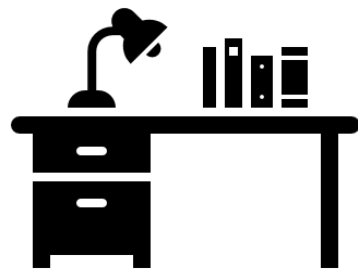
```
> sum(2,5);  
7  
undefined  
> sum(2,5) * 3;  
7  
NaN
```

```
function sum(num1, num2){  
  return num1+num2;  
}
```

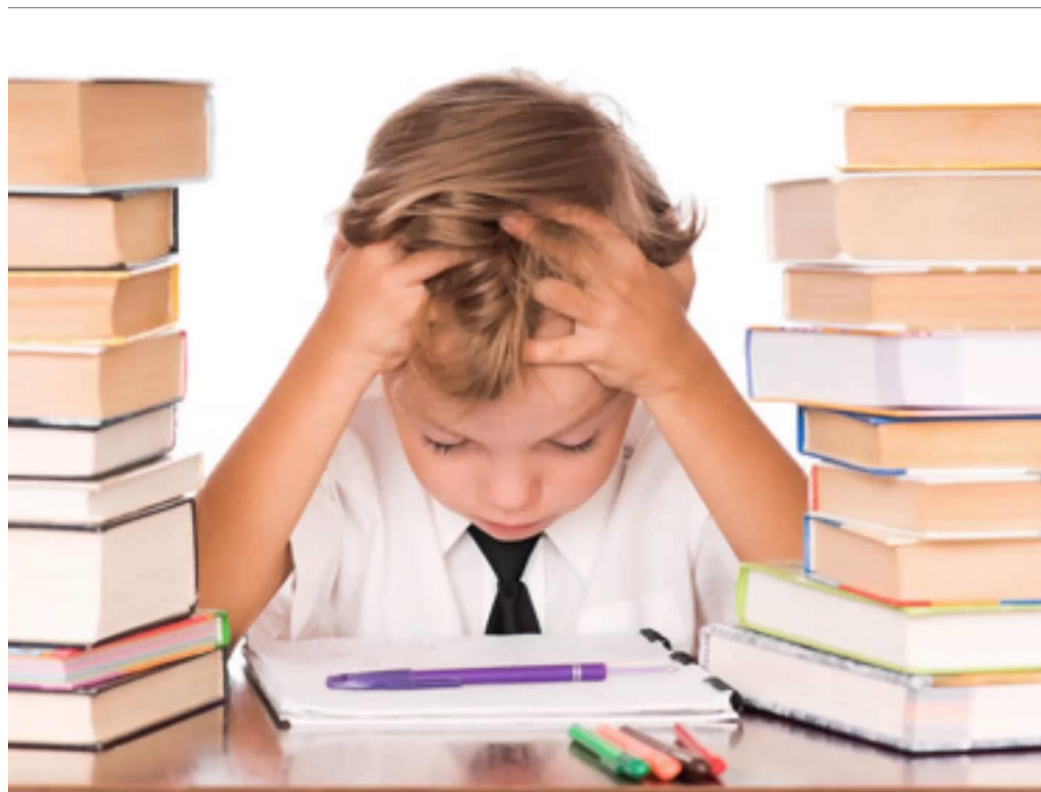


```
> sum(2,5);  
< 7  
> sum(2,5) * 3;  
< 21
```

Exercício



Ficha de Trabalho 1



Funções - scope das variáveis

- O scope é a visibilidade da variável.
- Define onde temos acesso à variável.
- Se definida dentro de uma função, a variável só pode ser usada dentro dela.

```
✓ function collectEggs(){  
  let totalEggs = 6;  
  console.log(totalEggs);  
}
```

```
LIVE Reload enabled.  
> collectEggs()  
6  
< undefined  
> console.log(totalEggs);  
✖ ▶ Uncaught ReferenceError: totalEggs is not defined  
   at <anonymous>:1:13  
>
```

Funções - scope das variáveis

```
totalEggs = 20;  
  
✓ function collectEggs(){  
  let totalEggs = 6;  
  console.log(totalEggs);  
}
```

```
collectEggs()
```

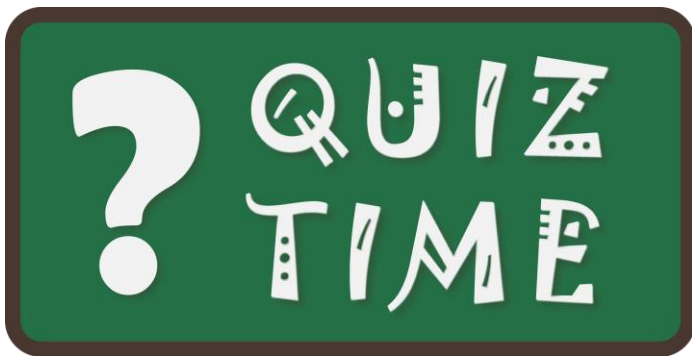
```
6
```

```
undefined
```

```
console.log(totalEggs);
```

```
20
```

```
undefined
```



```
let animal = "Giant Pacific Octopus";  
function observe(){  
  let animal = "Pajama Squid";  
  console.log(animal);  
}  
observe();
```

is the result of running the following code:

```
const creature = "Common Sea Dragon";  
  
function scubaDive(){  
  const creature = "Spanish Dancer"; //A type of sea slug  
  console.log(creature);  
}  
  
scubaDive();
```

two values are printed to the console:

```
let deadlyAnimal = "Blue-Ringed Octopus";  
  
function handleAnimal() {  
  let deadlyAnimal = "Scorpionfish";  
  console.log(deadlyAnimal);  
}  
  
handleAnimal();  
console.log(deadlyAnimal)
```

Funções - Outras formas de definir

```
const sum = function(num1, num2){  
  return num1+num2;  
}
```

```
> sum(6,6);  
◀ 12
```

Métodos

- Podemos adicionar funções como propriedades de um objecto.
- A essas funções chamamos métodos.

```
> myMath.sum(2,2)
```

```
< 4
```

```
> myMath.divide(2,2)
```

```
< 1
```

```
> myMath.multiply(6,2)
```

```
< 12
```

```
✓ const myMath = {  
  sum: function (x, y){  
    return x + y;  
  },  
  multiply: function (x, y){  
    return x * y;  
  },  
  divide: function (x, y){  
    return x / y;  
  }  
}
```

Métodos, a key “this”.

- A key this é usada quando queremos aceder a outras propriedades dentro do mesmo objecto

```
const person = {  
  first: 'Robert',  
  last: 'Herjavec',  
  fullName() {  
    return `${this.first} ${this.last}`  
  }  
}  
  
person.fullName(); //"Robert Herjavec"  
person.last = "Plant";  
person.fullName(); //"Robert Plant"
```


Callbacks e Métodos de Arrays

- Os arrays de JS têm uma série de métodos que permitem que lhes passemos directamente uma função.
 - ForEach
 - Map
 - Filter
 - Find
 - Some & Every
 - Reduce

- Arrow Functions

Métodos de Arrays: For Each

- Aceita uma função de callback.
- Chama a função uma vez por cada elemento do Array.
- A função é anónima porque só nos irá servir para este propósito

```
const nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];

nums.forEach(function (n) {
  console.log(n * n)
  //prints: 81, 64, 49, 36, 25, 16, 9, 4, 1
});

nums.forEach(function (el) {
  if (el % 2 === 0) {
    console.log(el)
    //prints: 8, 6, 4, 2
  }
})
```

```
const daysOfWeek = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta',
  'Domingo'];

daysOfWeek.forEach(function(n) {
  console.log(n);
});
```

Métodos de Arrays: For Each

```
const movies = [  
  {  
    title: 'Amadeus',  
    score: 99  
  },  
  {  
    title: 'Stand By Me',  
    score: 85  
  },  
  {  
    title: 'Parasite',  
    score: 95  
  },  
  {  
    title: 'Alien',  
    score: 90  
  }  
]
```

```
movies.forEach(function (movie) {  
  console.log(`${movie.title} - ${movie.score}/100`)  
})
```

Métodos de Arrays: Map

- Aceita uma função de Callback.
- Cria um novo array com os resultados da chamada a cada elemento do Array.
- Mapeia um array de um estado para outro.

Array original

Array do Mapeamento

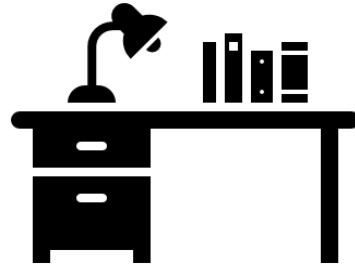
```
const texts = ['rofl', 'lol', 'omg', 'ttyl'];  
const caps = texts.map(function (t) {  
  return t.toUpperCase();  
})  
texts; //["rofl", "lol", "omg", "ttyl"]  
caps;  //["ROFL", "LOL", "OMG", "TTYL"]
```

Métodos de Arrays: Map

```
const movies = [  
  {  
    title: 'Amadeus',  
    score: 99  
  },  
  {  
    title: 'Stand By Me',  
    score: 85  
  },  
  {  
    title: 'Parasite',  
    score: 95  
  },  
  {  
    title: 'Alien',  
    score: 90  
  }  
]
```

```
const titles = movies.map(function (movie) {  
  return movie.title.toUpperCase();  
})
```

Exercício



1. Guarde o seguinte array no seu código:

```
const fullNames = [{first: 'Albus', last: 'Dumbledore'}, {first: 'Harry', last: 'Potter'},  
{first: 'Hermione', last: 'Granger'}, {first: 'Ron', last: 'Weasley'}, {first: 'Rubeus', last:  
'Hagrid'}, {first: 'Minerva', last: 'McGonagall'}, {first: 'Severus', last: 'Snape'}];
```
2. Crie um novo array chamado `firstNames` que nos retorne os primeiros Nomes do Array Original.

Arrow Functions

- Nova sintaxe para definir funções.
- Não têm suporte no IE.
- Alternativa mais compacta para definir uma função regular.

```
const square = (x) => {  
  return x * x;  
}
```

```
const sum = (x, y) => {  
  return x + y;  
}
```

Exercício



1. Usando as Arrow Functions, crie uma função chamada greet que receba um argumento que represente o nome de uma pessoa.

```
// crie uma função arrow greeting usando a  
| greet("Hagrid") //"Hey Hagrid!"  
| greet("Luna") //"Hey Luna!"
```


Arrow Functions: o return Implícito

A ideia das arrow functions é simplificar o nosso código.

Se substituirmos as {} por () ele automaticamente toma como implícito que é para retornar os valores.

Atenção: apenas funciona nas Arrow Functions.

```
const isEven = function (num) { //regular function expression
  return num % 2 === 0;
}
const isEven = (num) => { //arrow function with parens around param
  return num % 2 === 0;
}
const isEven = num => { //no parens around param
  return num % 2 === 0;
}
const isEven = num => ( //implicit return
  num % 2 === 0
);
const isEven = num => num % 2 === 0; //one-liner implicit return
```

Arrow Functions: o return Implícito

```
const firstNames = fullNames.map(function (name) {  
  return name.first;  
});  
  
const newFirstNames = fullNames.map(name => (  
  name.first  
))
```

Funções de Callback: setTimeout

A função `setTimeout` usa-se quando queremos que algo seja executado passado x tempo.

Recebe dois argumentos: uma função que fica em espera e o timer.

```
console.log('hello');  
setTimeout(() => {  
  alert("Ainda estás aí?!")  
}, 3000)
```

Funções de Callback: setInterval



Centro para o Desenvolvimento
de Competências Digitais

A função setInterval define que a cada x tempo uma função corre;

```
setInterval(() => {  
  console.log(Math.random())  
}, 2000)
```

Métodos de Arrays: Filter

- Cria um novo array com todos os elementos que passaram no teste implementado na função de filtro.

```
const nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];
const odds = nums.filter(n => {
  return n % 2 === 1; //our callback returns true or false
  //if it returns true, n is added to the filtered array
})
//[9, 7, 5, 3, 1]

const smallNums = nums.filter(n => n < 5);
//[4, 3, 2, 1]
```

Exercício



1. Escreva uma função chamada `validUserNames` que aceite um array de usernames (que serão strings).
2. A nossa função deverá retornar um novo array contendo apenas os usernames que têm menos de 10 caracteres.

```
validUserNames(['mark', 'staceysmom1978',  
                'q29832128238983', 'carrie98', 'MoanaFan']);  
// => ["mark", "carrie98", "MoanaFan"]
```

Métodos de Arrays: Every

- Testa se **todos** os elementos do array que passam na função de validação e retorna um Boolean;

```
const words = ["dog", 'dig', 'log', 'bag', 'wag'];

words.every(word => {
  return word.length === 3;
}) //true

words.every(word => word[0] === 'd'); //false

words.every(w => {
  let last_letter = w[w.length - 1];
  return last_letter === 'g'
}) //true
```

Métodos de Arrays: Some

- Semelhante ao Every mas testa se algum dos elementos do array passa na função de validação e retorna um Boolean;

```
const words = ['dog', 'jello', 'log', 'cupcake', 'bag', 'wag'];

//Are there any words longer than 4 characters?
words.some(word => {
  return word.length > 4;
}) //true

//Do any words start with 'Z'?
words.some(word => word[0] === 'Z'); //false

//Do any words contain 'cake'?
words.some(w => w.includes('cake')) //true
```


Exercício



1. Escreva uma função chamada `allEvens` que aceite um array de números.
2. A nossa função deverá retornar verdadeiro se todos os números forem pares.

```
allEvens([2,4,6,8]) //true  
allEvens([1,4,6,8]) //false  
allEvens([1,2,3]) //false
```

Métodos de Arrays: Reduce

- Executa uma função que reduz todos elemento do array a um valor, somando-os ou usando a operação pretendida.

```
[3, 5, 7, 9, 11].reduce((accumulator, currentValue) => {  
    return accumulator + currentValue;  
});
```

Recursos

- <https://developer.mozilla.org/en-US/docs/Web/JavaScript>