

Linguagens Script

<JavaScript>

Licenciatura em Engenharia Informática

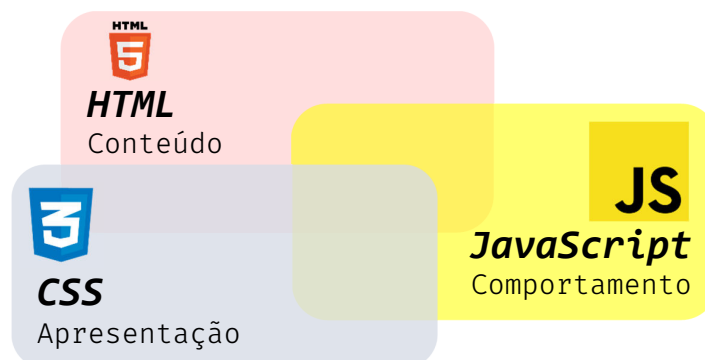
Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2022/2023

> Introdução ao JavaScript

- Linguagem de programação de alto nível, interpretada, com tipagem dinâmica fraca e multi-paradigma.
- Uma das tecnologias mais utilizadas no *front-end development*.
 - *Onde incidirão as aulas práticas Linguagens Script!*



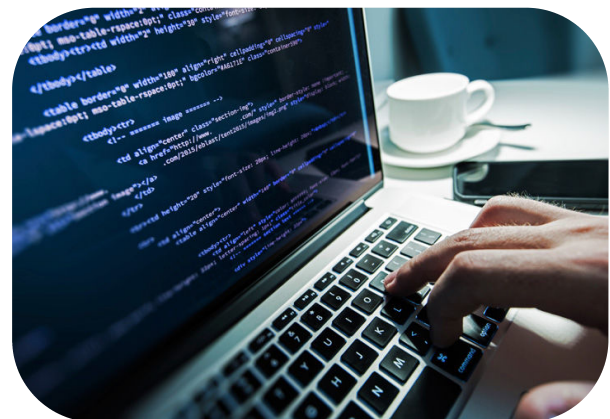
> Características

- Linguagem de Script
- Linguagem Dinâmica (*Dinamic Typing*)
- Linguagem Interpretada*
- Multi-Paradigma
 - JavaScript possibilita o uso de um conjunto de técnicas da linguagem de programação funcional (declarativa) e procedimental (imperativo);
 - O JavaScript permite a programação orientada a objetos (POO), embora **não siga os paradigmas mais puros** associados à POO;
 - Tal como o resto do JavaScript o objetivo é o resultado final e as funcionalidades oferecidas, em contraste com o respeito por paradigmas ou os padrões de programação mais rígidos;
 - *Prototype-based Programming*
- Independente de Plataforma
- Permite processamento assíncrono



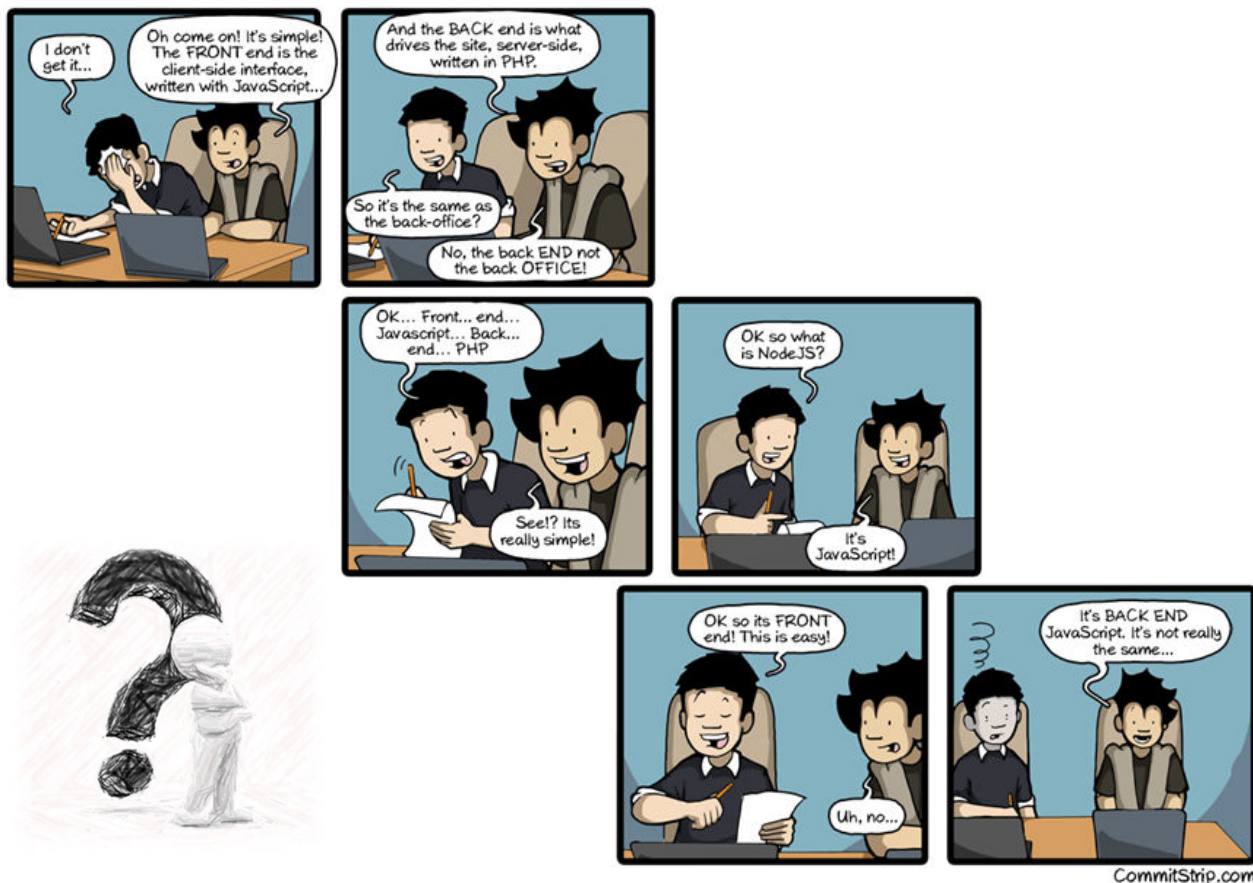
> Algumas Desvantagens...

- Podem existir questões de segurança ao nível do *client-side*, quando existe uma programação descuidada;
- Suporte dos *browsers*;
- Herança única;
- Dificuldade de *debug*;
- Erros de código podem provocar falha na renderização da página impedindo de a visualizar no *browser*, apesar da grande tolerância destes aos erros...



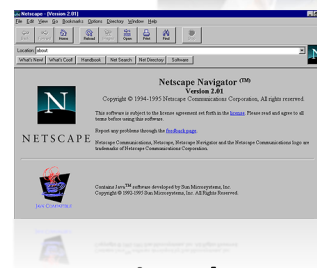
"To every disadvantage, there is a corresponding advantage", W. Clement Stone

> front-end vs back-end



> Um pouco de história...

- Criado por *Brendan Eich*, em **1995**, enquanto Engenheiro da **Netscape**;
- Primeiro lançamento junto com o *browser Netscape 2.0*, no início de **1996**;
- Microsoft lançou **JScript** com **Internet Explorer 3**;
- Mais tarde, Netscape submeteu JavaScript ao **ECMA International**
 - **European Computer Manufacture's Association**
 - Criada a primeira edição do standard ECMAScript
 - Standard para linguagens script
 - Tem sofrido várias alterações ao longo dos tempos, com uma alteração significativa em 2015

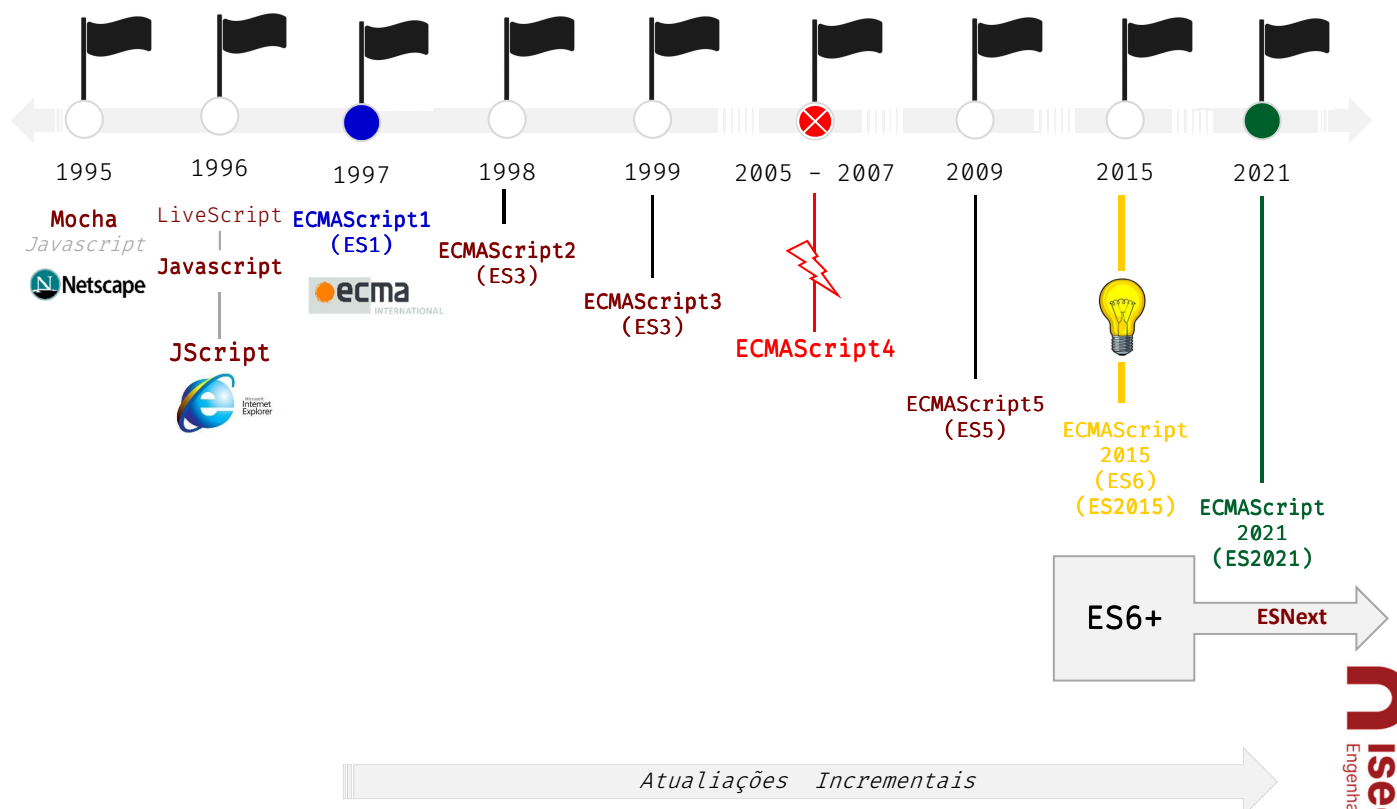


> ECScript (ES)

- ECMAScript é um standard para linguagens script, e o documento ECMA-262 é a especificação desta linguagem.
 - Desenvolvida pela *Technical Committee 39* (TC-39) - ECMA internacional
 - A primeira edição do ECMA-262 foi adotada pela *ECMA General Assembly* em Junho 1997.
 - <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>
 - JavaScript é a implementação mais popular deste standard, sendo por isso comum designar o “EcmaScript” como “JavaScript”.
- As características base do JavaScript são baseadas neste standard internacional, embora existem outras características que não estão nesta especificação.
- Não confundir a linguagem **JavaScript** com a linguagem **Java!!!**



> ECScript *Timeline*



JavaScript é uma linguagem **interpretada** ou **compilada**?



< 9 >

> Compilada vs Interpretada

- JavaScript é considerada uma **linguagem interpretada**, mas...
 - Javascript *engines* modernos não interpretam apenas JavaScript, também efetuam uma compilação.
- Transformação com início em 2009, no compilador *SpiderMonkey JavaScript*, adicionado ao Firefox 3.5, tendo os outros browsers seguido essa mesma abordagem.








JavaScript é compilado internamente com o designado **just-in-time (JIT) compilation** permitindo melhorar o tempo de execução do código JavaScript.

> JavaScript Engine

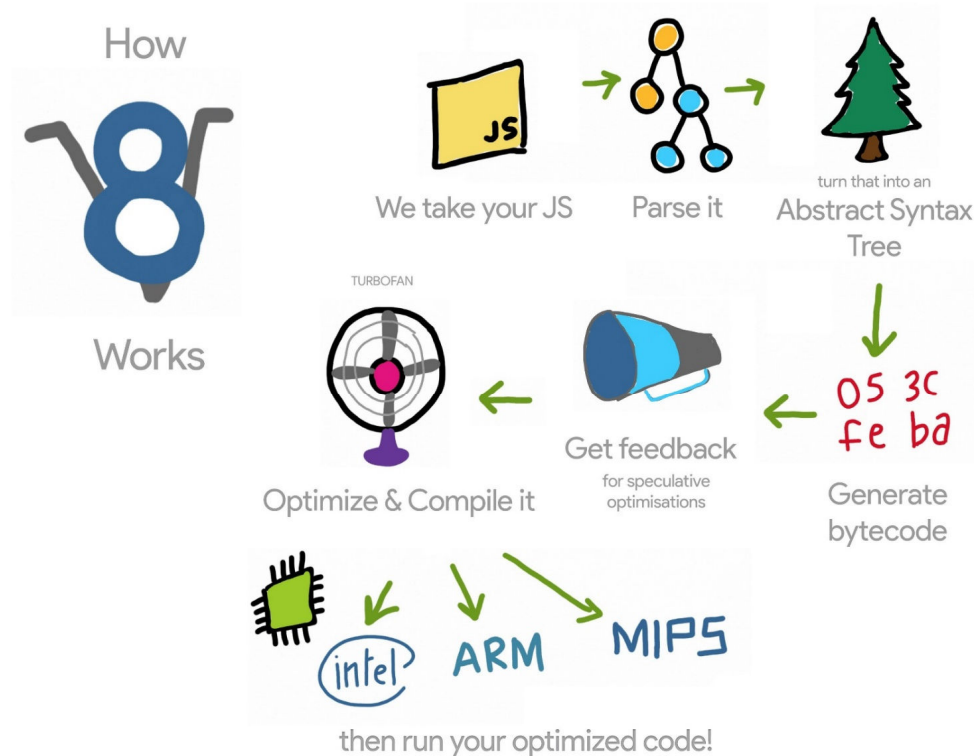
- *JS Engines* são essencialmente programas que convertem código JavaScript em código de baixo nível ou código máquina, de forma a ser perceptível pelo computador. Assim, **JavaScript engine** é um componente de software que permite a execução de código **JavaScript**.
- Embebidos em *browsers* e servidores web (NodeJS) para permitir a execução e compilação *runtime-time*
- Os primeiros *JavaScript engines* eram apenas interpretadores mas todos os mecanismos modernos, recorrem a mecanismos mais complexos como o *just-in-time compilation* de forma a melhorar o seu desempenho.
- Segue o standard ECMAScript



> JavaScript Engines

- **V8** (o mais popular) 
▪ Google Chrome, Edge*, NodeJs... <https://v8.dev>
▪ Video sobre V8 - [*"the key engineering decisions behind, V8, the JavaScript virtual machine used in Google Chrome. the JavaScript virtual machine used in Google Chrome"*](#)
- **SpiderMonkey**  
▪ Firefox (Warp – WarpBuilder) <https://spidermonkey.dev/>
- **Chakra**  **Microsoft**
▪ Internet Explorer (*browser* antigo da MS, o atual usa V8)
- **JavaScriptCore / Nitro**  **Apple**
▪ Safari

> V8 Engine



By @addyosmani

<https://twitter.com/addyosmani/status/829728691798188034/photo/1>

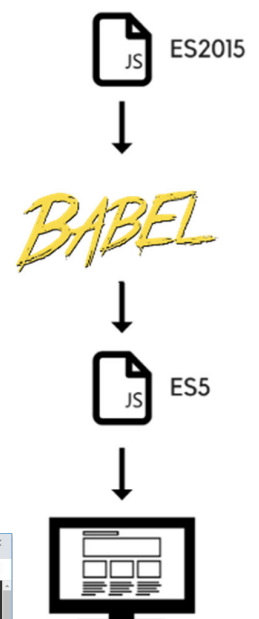
> Compatibilidade

- **Babel**
 - Compilador / **Transpilador**
 - Converte código JavaScript atual (ECMAScript 2015+) para uma versão em que o browser, de versão mais antiga, o possa executar.
- Importante para *front-end developers*.

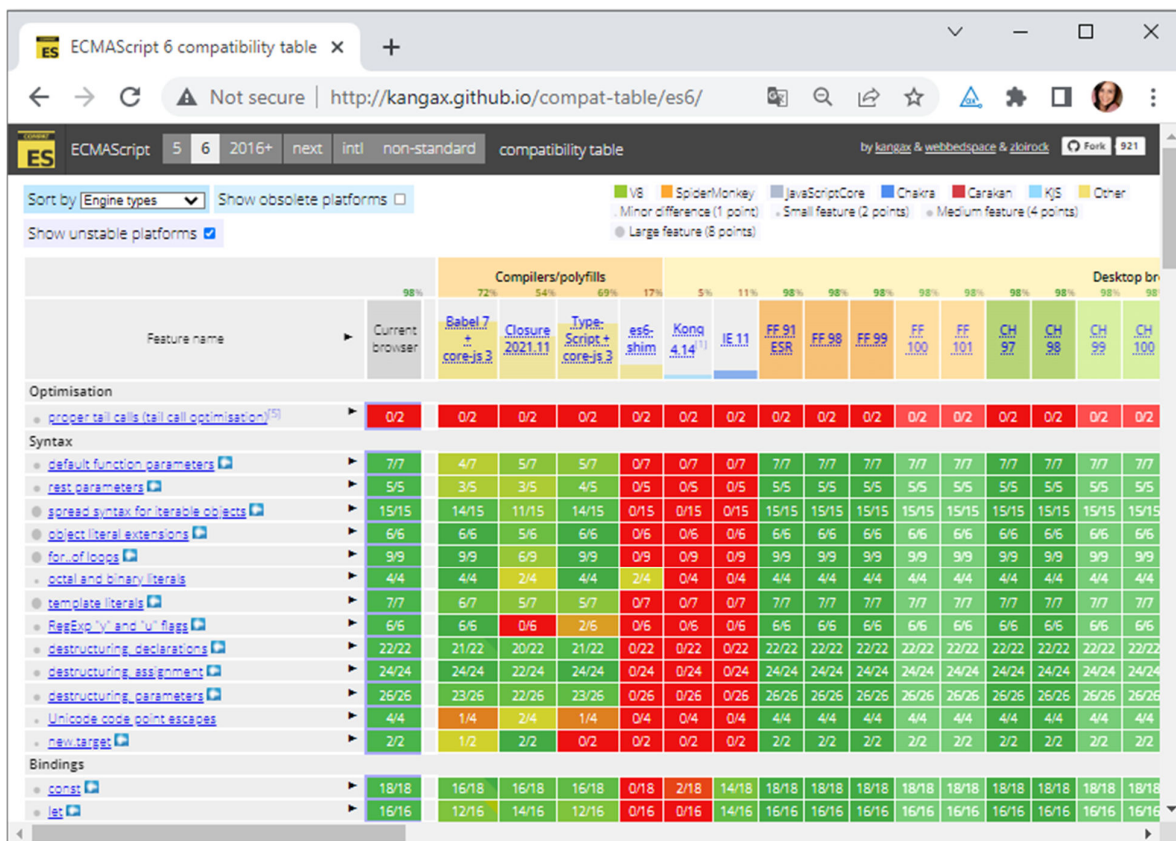
```
// Babel Input: ES2015 arrow function
[1, 2, 3].map(n => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function (n) {
  return n + 1;
});
```

<https://babeljs.io/docs/>



> Compatibilidades



The screenshot shows the 'ECMAScript 6 compatibility table' website. It features a navigation bar with tabs for 'ES 5', 'ES 6', '2016+', 'next', 'intl', 'non-standard', and 'compatibility table'. Below the navigation bar, there are filters for 'Sort by' (Engine types), 'Show obsolete platforms' (checkbox), and 'Show unstable platforms' (checkbox). A legend indicates feature types: Minor difference (1 point), Small feature (2 points), Medium feature (4 points), and Large feature (8 points). The table lists various features under categories like 'Optimisation', 'Syntax', and 'Bindings', with columns for different browsers and their compatibility status (e.g., 0/2, 7/7, 15/15).

| Feature name | Current browser | Babel 7 | Closure 2021.11 | TypeScript 3.9.7 | es6-shim | Kong 4.14 | IE 11 | FF 91 ESR | FF 98 | FF 99 | FF 100 | FF 101 | CH 97 | CH 98 | CH 99 | CH 100 |
|--|-----------------|---------|-----------------|------------------|----------|-----------|-------|-----------|-------|-------|--------|--------|-------|-------|-------|--------|
| Optimisation | | | | | | | | | | | | | | | | |
| • proper tail calls (tail call optimisation) | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 | 0/2 |
| Syntax | | | | | | | | | | | | | | | | |
| • default function parameters | 7/7 | 4/7 | 5/7 | 5/7 | 0/7 | 0/7 | 0/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 |
| • rest parameters | 5/5 | 3/5 | 3/5 | 4/5 | 0/5 | 0/5 | 0/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 | 5/5 |
| • spread syntax for iterable objects | 15/15 | 14/15 | 11/15 | 14/15 | 0/15 | 0/15 | 0/15 | 15/15 | 15/15 | 15/15 | 15/15 | 15/15 | 15/15 | 15/15 | 15/15 | 15/15 |
| • object literal extensions | 6/6 | 6/6 | 5/6 | 6/6 | 0/6 | 0/6 | 0/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 |
| • for...of loops | 9/9 | 9/9 | 6/9 | 9/9 | 0/9 | 0/9 | 0/9 | 9/9 | 9/9 | 9/9 | 9/9 | 9/9 | 9/9 | 9/9 | 9/9 | 9/9 |
| • octal and binary literals | 4/4 | 4/4 | 2/4 | 4/4 | 2/4 | 0/4 | 0/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 |
| • template literals | 7/7 | 6/7 | 5/7 | 5/7 | 0/7 | 0/7 | 0/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 | 7/7 |
| • RegExp 'v' and 'u' flags | 6/6 | 6/6 | 0/6 | 2/6 | 0/6 | 0/6 | 0/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 | 6/6 |
| • destructuring declarations | 22/22 | 21/22 | 20/22 | 21/22 | 0/22 | 0/22 | 0/22 | 22/22 | 22/22 | 22/22 | 22/22 | 22/22 | 22/22 | 22/22 | 22/22 | 22/22 |
| • destructuring assignment | 24/24 | 24/24 | 22/24 | 24/24 | 0/24 | 0/24 | 0/24 | 24/24 | 24/24 | 24/24 | 24/24 | 24/24 | 24/24 | 24/24 | 24/24 | 24/24 |
| • destructuring parameters | 26/26 | 23/26 | 22/26 | 23/26 | 0/26 | 0/26 | 0/26 | 26/26 | 26/26 | 26/26 | 26/26 | 26/26 | 26/26 | 26/26 | 26/26 | 26/26 |
| • Unicode code point escapes | 4/4 | 1/4 | 2/4 | 1/4 | 0/4 | 0/4 | 0/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 | 4/4 |
| • new.target | 2/2 | 1/2 | 2/2 | 0/2 | 0/2 | 0/2 | 0/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 | 2/2 |
| Bindings | | | | | | | | | | | | | | | | |
| • const | 18/18 | 16/18 | 16/18 | 16/18 | 0/18 | 2/18 | 14/18 | 18/18 | 18/18 | 18/18 | 18/18 | 18/18 | 18/18 | 18/18 | 18/18 | 18/18 |
| • let | 16/16 | 12/16 | 14/16 | 12/16 | 0/16 | 0/16 | 14/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 | 16/16 |

<http://kangax.github.io/compat-table/es6/>

> JS Consola

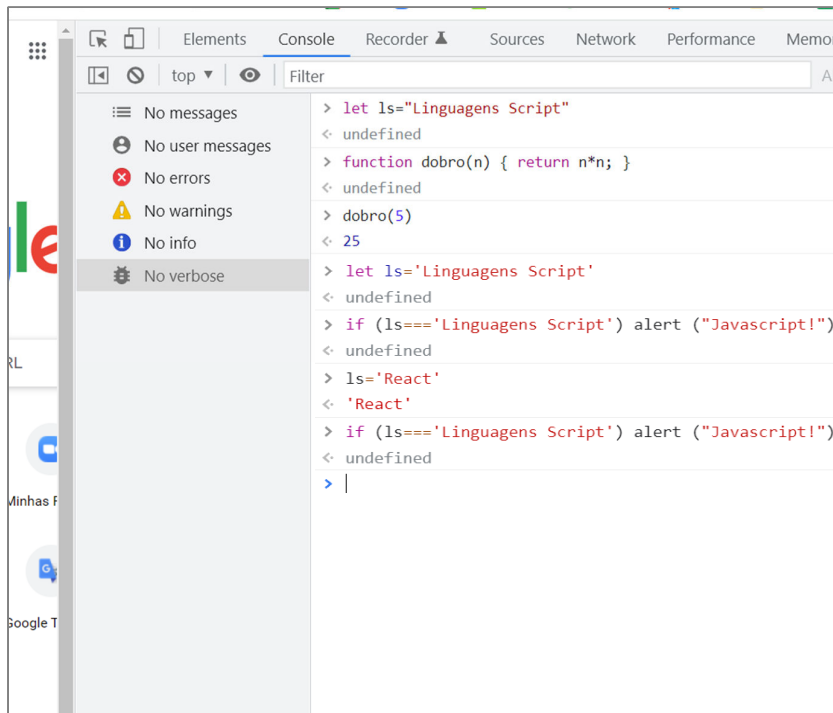
- Instalação do **nodejs.dev**



```
Command Prompt - node
C:\Users\Cristiana>node
Welcome to Node.js v16.14.0.
Type ".help" for more information.
> let palavra='Linguagens'
undefined
> let palavra2='Script'
undefined
> console.log(palavra+palavra2)
LinguagensScript
undefined
> 2*2
4
> let contador=10*5
undefined
> contador
50
>
```


> JS no browser

- Para criação de aplicações web (*front-end*)



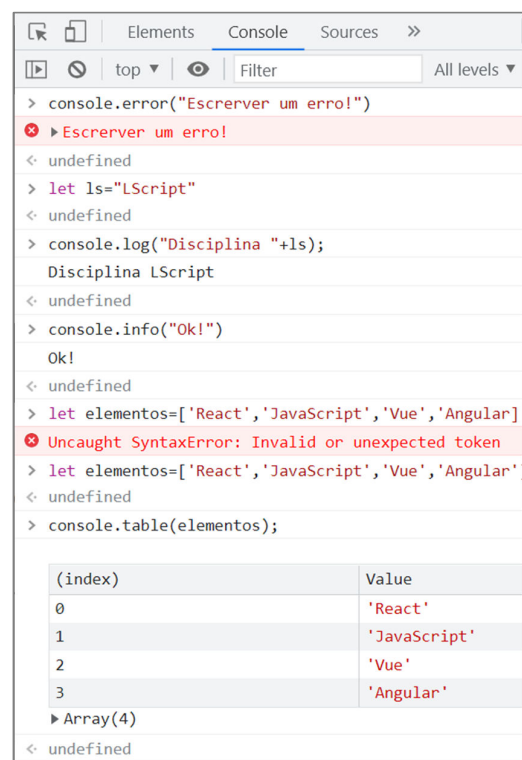
Ctrl + Shift + J

OK

> Comandos úteis na consola

- **console.log()**
- **console.info()**
- **console.warn()**
- **console.error()**
- **console.time()**
- **console.timeEnd();**
- **console.clear() ou Ctrl + L**
- Outros:

- <https://css-tricks.com/a-guide-to-console-commands/>



> Características do JS Atual

- Declaração de variáveis com **let** e **const**;
- Funções mais legíveis e reduzidas com **arrow functions**;
- Facilidade em interpolar variáveis e expressões em *strings* com **template literals**;
- Introdução de novas funções para manipulação de **arrays**;
- Possibilidade de inserir **parâmetros** por omissão;
- Existência de **atalhos** para criação de propriedades de objectos;
- **Classes e atributos privados**, ...
- ...



A explorar nas aulas...

> Sintaxe Básica > Considerações

- *case sensitive*
 - let linguagens**S**cript **≠** Let linguagens**ss**cript
- Comentários
 - // símbolo do comentário
 - /* comentário para múltiplas linhas */
- Convenções de codificação em JavaScript
 - camelcase
 - linguagem**S**cript
- Regras específicas
 - Não podem iniciar com número

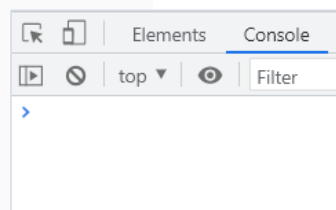
```
let strLS="Linguagens Script";  
let strJS="Javascript";  
  
let 2LS="LS";
```

> Strict Mode

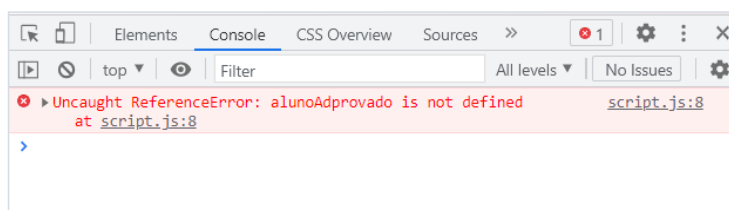
- Método mais rigoroso, evitando potenciais problemas de implementação;
- Facilita identificação de erros;
- Recomenda-se o uso do **strict mode** nas aulas práticas;

`'use strict';`

```
//'use strict';  
  
let disciplina = "Linguagens Script";  
let nota =10;  
let alunoAprovado=false;  
if(nota>=10) alunoAdprovado=true;  
if (alunoAprovado) console.log("Aluno Aprovado a "+disciplina+"!");
```



`'use strict';`



JavaScript

<Variáveis>

Linguagens Script @ LEI / LEI-PL / LEI-CE

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2022/2023

Declaração de Variáveis

- › Declaração de Variáveis
 - › *var vs let vs const*
- › Visibilidade das Variáveis (*Scope*)
 - › Global
 - › Function
 - › Block

< 23 >

> Declaração de Variáveis

- Declaração de variáveis:
 - **var** 🖐️
 - **let** (ES6) 👍
 - **const** (ES6) 👍
- Objetos e funções são também variáveis.
- **Hoisting** é um comportamento padrão em JavaScript, movendo as declarações para o topo do *scope* presente (seja topo do *script* ou topo da função).
- A forma **como** se declara e **onde** se declara uma variável, um objeto, função, **influencia a sua visibilidade e acessibilidade.**



> Scope das Variáveis

▪ Scope

- Visibilidade e acessibilidade das variáveis em diferentes zonas do programa, isto é, que dados podem ser acedidos em determinada zona do programa;
- Permite criação de variáveis públicas ou privadas;
- Permite evitar a colisão de nomes de variáveis;
- *Garbage Collection*, ...

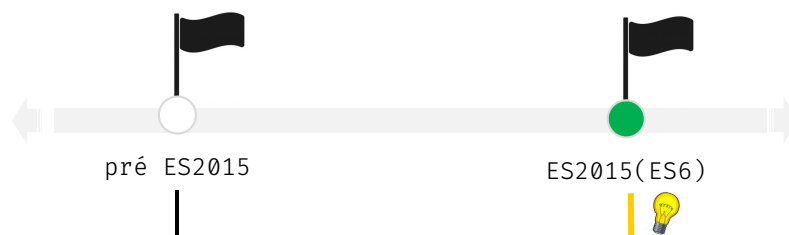
Scope
"The current context of execution. The context in which values and expressions are "visible" or can be referenced. If a variable or other expression is not "in the current scope," then it is unavailable for use.", [MDN](#)

▪ JavaScript atual permite três tipos de *scope*

- **Global** – Quando declarados fora de qualquer função ou bloco de código;
- **Function** – Quando declarados em funções;
- **Block** (surgiu com o ES6) – Quando declarados em estruturas de controlo (if, for, while,...), num bloco { }

- Um *scope* tem acesso a todas as variáveis de todos os seus escopos externos - *scope chain*;

> Variáveis > var vs let vs const



| | var | let | e | const |
|---|--|---|--------------|-----------------|
| scope | global ou function | global ou block | | global ou block |
| hoisting | Sim, para o topo da sua execução (global ou function scope) e é inicializado como <i>undefined</i> | Sim, para o topo da sua execução (global ou block scope) e não é inicializada | igual ao let | |
| Permite nova declaração dentro do scope? | Sim | Não | Não | |
| Permite nova atribuição no scope? | Sim | Sim | Não | |

> Declaração variáveis > var

```
var mensagem = "Nova Mensagem!";  
console.log(mensagem);
```

```
mensagem = "Mensagem final...."  
console.log(mensagem);
```

Nova Mensagem!
Mensagem final....

```
var mensagem = "Bem Vindo!";  
function mensagemBoasVindas() {  
    var ola = "Ola";  
}
```

```
console.log(ola);  
console.log(mensagem);  
console.log(ola + ' ' + mensagem);
```

✖ ▶ Uncaught ReferenceError: ola is not defined

Bem Vindo!

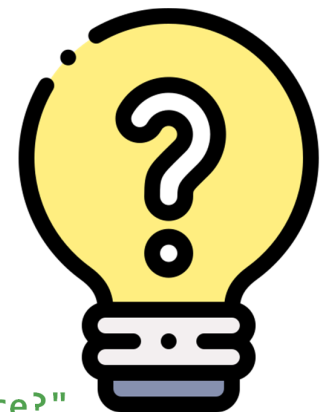
✖ ▶ Uncaught ReferenceError: ola is not defined

> Variáveis > var > hoisting

```
console.log(hoistingMessage);  
var hoistingMessage = "O que acontece?"
```

var hoistingMessage;
console.log(hoistingMessage);
hoistingMessage = "O que acontece?"

undefined



> Declaração variáveis > *hoisting*

```
var mensagem = "Ola";
var contador = 4;
if (contador > 3) {
  console.log(mensagem);
  var mensagem = "Dizer antes: Olá Malta!";
}
console.log(mensagem);
```



Ola
dizer antes: Olá Malta!

```
let mensagem = "Ola";
let contador = 4;
if (contador > 3) {
  console.log(mensagem);
  let mensagem = "dizer antes: Olá Malta!";
}
console.log(mensagem);
```



✖ ▶ Uncaught ReferenceError: Cannot access 'mensagem' before initialization

> Declaração variáveis > let

```
let mensagem = "Nova Mensagem!"
console.log(mensagem);
mensagem = "Mensagem final...."
console.log(mensagem);
```



Nova Mensagem!
Mensagem final....

```
let mensagem = "Nova Mensagem!"
console.log(mensagem);
let mensagem = "Mensagem final...."
console.log(mensagem);
```



✖ Uncaught SyntaxError: Identifier 'mensagem' has already been declared

> Declaração variáveis > let

```
let mensagem = "Ola!";
if (true) {
  console.log(mensagem);
  let mensagem2 = "Olá Malta";
}
console.log(mensagem);
console.log(mensagem2);
```



```
Ola!
Ola!
✖ ▶ Uncaught ReferenceError: mensagem2 is
not defined
```

```
let mensagem = "Ola!";
if (true) {
  let mensagem = "Olá Malta";
  console.log(mensagem);
}
console.log(mensagem);
```



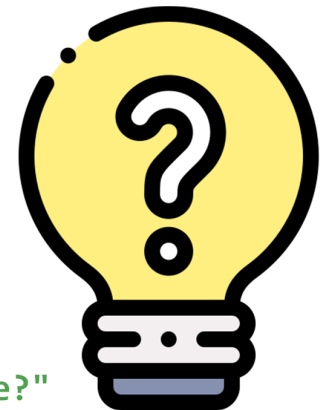
```
Olá Malta
Ola!
```

> Variáveis > let > hoisting

```
console.log(hoistingMessage);
let hoistingMessage = "O que acontece?"
```



```
let hoistingMessage;
console.log(hoistingMessage);
hoistingMessage = "O que acontece?"
```



```
✖ ▶ Uncaught ReferenceError: Cannot access
'hoistingMessage' before initialization
```

> Variáveis > let > Exercício

```
let varA = 2;
```

```
if (varA > 1) {  
  let varB = varA * 3;  
  console.log(varB);
```



```
  for (let i = varA; i <= varB; i++) {  
    let j = i + 10;  
    console.log(j);  
  }
```



```
  let c = varA + varB;  
  console.log(c);  
}
```



| |
|----|
| 6 |
| 12 |
| 13 |
| 14 |
| 15 |
| 16 |
| 8 |

> Declaração variáveis > const

```
const mensagem = "Bem Vindo!";  
console.log(mensagem);
```



```
mensagem = "Olá Malta!";  
console.log(mensagem);
```

Bem Vindo!

✖ ▶ Uncaught TypeError: Assignment to constant variable.

```
const mensagem = "Bem Vindo!";  
console.log(mensagem);
```



```
const mensagem = "Olá Malta!";  
console.log(mensagem);
```

✖ Uncaught SyntaxError: Identifier 'mensagem' has already been declared

```
const mensagem;
```



✖ Uncaught SyntaxError: Missing initializer in const declaration

> Declaração variáveis > const

- Toda a declaração const, deve ser inicializada no momento da declaração, no entanto, o comportamento é diferente, quando se declara objectos com const.

```
const aluno = {  
  nome: "Manuel Ruivo",  
  numero:123  
}  
aluno = {  
  nome: "Jose Ruivo",  
  numero: 123  
}  
aluno.nome = "Jose Ruivo";
```



A ver mais tarde...

> Variáveis > var vs let vs const

```
'use strict';  
{  
  let disciplina = 'Linguagens Script';  
  const CODIGO = 'LS';  
  var ano = 1;  
}
```

```
console.log(disciplina);  
console.log(CODIGO);  
console.log(ano);
```

✖ ▶ Uncaught ReferenceError: disciplina is not defined
✖ ▶ Uncaught ReferenceError: CODIGO is not defined

1

```
function exemplo1() {  
  var a = "Linguagens Script";  
}  
console.log(a);
```

✖ ▶ Uncaught ReferenceError: a is not defined

> Variáveis > var vs let

JavaScript

```
'use strict';  
for (let i = 0; i < 3; i++) {  
    console.log(i);  
}  
console.log(i);
```

| |
|---|
| 0 |
| 1 |
| 2 |

✖ ▶ Uncaught ReferenceError: i is not defined

```
var nomeDisciplinaVar1 = "Nome alterado...var";  
var nomeDisciplinaVar2 = "Declarada If - var2";  
let nomeDisciplinaLet1 = "Declarada If - let1";  
if (true) { console.log("Dentro do if - LS var1");  
    var nomeDisciplinaVar1 = "Nome alterado...var";  
    var nomeDisciplinaVar2 = "Declarada If - var2";  
    let nomeDisciplinaLet1 = "Declarada If - let1";  
    nomeDisciplinaVar = "Nome alterado...var";  
    nomeDisciplinaLet = "Nome alterado...let";  
}  
console.log(nomeDisciplinaVar); console.log(nomeDisciplinaVar1);  
console.log(nomeDisciplinaVar2); console.log(nomeDisciplinaLet);  
console.log(nomeDisciplinaLet1);
```

| |
|------------------------|
| Dentro do if - LS var1 |
| Nome alterado...var |
| Redeclarada If - var1 |
| Declarada If - var2 |
| Nome alterado...let |

✖ ▶ Uncaught ReferenceError: nomeDisciplinaLet1 is not defined



isec
Engenharia

JavaScript

Operadores e Estruturas

- › Operadores
 - › Básicos
 - › Condicional – Ternário
 - › Outros (ES6)
- › Estruturas de Controlo
 - › Condicionais
 - › Repetição

> Alguns Operadores JavaScript

- JavaScript suporta vários tipos de operadores:
 - Aritméticos:** Operadores matemáticos básicos;
 - Relacionais:** Operadores que comparam dois valores. Muitas vezes designados como operadores de comparação;
 - Lógicos:** Operadores lógicos de forma a combinar dois ou mais *statements* relacionais;
 - Bitwise:** Usado para desempenhar operações com bits ou valores binários que envolvam a manipulação individual de bits;
 - Atribuição (Assignment):** Utilizados para atribuir um novo valor a uma variável, propriedade, evento ;
 - Type :** É um operador unário (typeof) que permite retornar o tipo do operando;
 - Outros:** Concatenation (+)+, Negação (-), Operador Condicional (?)

> Operadores > Básicos

- Aritméticos:** + - * / e %
- Atribuição** de valores
 - Com operador =
 - Instruções de atribuição compostas: += e -=
- Incrementos** com ++ e **Decrementos** com --
 - Como prefixo ou sufixo
- Operador + também permite **concatenação** de *strings*

```
let x = 2;  
x += 2;  
console.log(x);  
x = x + 2;  
console.log(x);  
x = 2;  
console.log(x++);  
console.log(--x);  
console.log(x);
```

4
6
2
2
2

```
let ls = 'Linguagens Script';  
console.log('Ola!' + ls);  
let sem = 2;  
let ano = 1;  
console.log(sem + ano + ' - Disciplina!' + ls);  
console.log('Disciplina!' + ls + " - " + sem + ano);
```

Ola!Linguagens Script
3 - Disciplina!Linguagens Script
Disciplina!Linguagens Script - 21

> Operadores > Comparações

- Comparações em JavaScript podem ser efetuadas com recurso a <, >, <= e >=, seja para valores numéricos ou strings;
- Igualdade
 - '18' == 18 **true**
 - 1 == true **true**
 - Automaticamente efetua a conversão de tipos (*dynamic type coercion*)
 - '18' === 18 **false**
 - 1 === true **false**
- Diferença
 - '18' != 18 **false**
 - '1' !== 1 **true**

> Operador Condicional > Ternário

- O operador condicional **?** retorna um de dois valores, tendo em consideração o valor lógico da condição;

condição ? exprSeTrue : exprSeFalse

```
let idade = 15;  
let situacao = (idade >= 18) ? "adulto" : "menor de idade";  
console.log(situacao);
```

menor de idade

```
let temporizador = 0;  
function gameOver(tempoJogo) {  
    return (tempoJogo == 0) ? true : false;  
}
```

```
gameOver(temporizador)  
    ? console.log("Fim de Jogo")  
    : console.log("Pode continuar!");
```

Fim de Jogo

> Outros Operadores – ES6

▪ Spread/Rest Operator ...

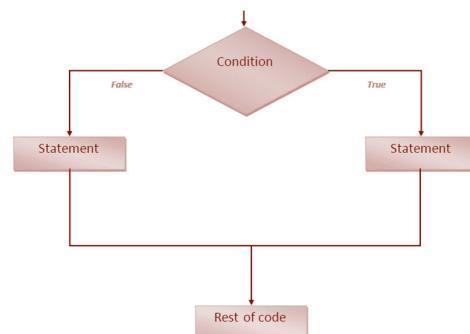
- ES6 introduziu o operador ... designado como *spread* ou *rest*, dependendo de como e onde os ... são utilizados;
- Exemplos:
 - Quando os ... são usados na frente de um array, o operador comporta-se como **spread operator** no seus valores individuais.
 - Quando os ... estão no fim dos parâmetros de uma função, é designado como **rest operator**;

- **Destructuring Assignment:** Recurso sintático chamado de desestruturação que permite atribuição via desestruturação;

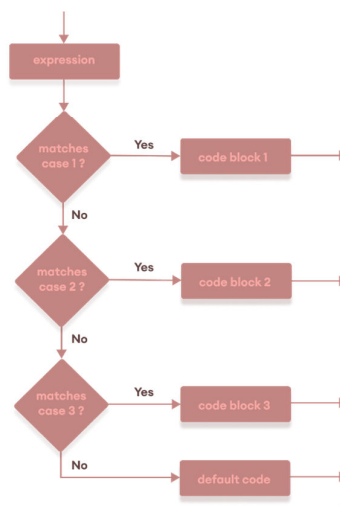
Vários exemplos da aplicação destes operadores nas secções de arrays, objetos, funções.

> Estruturas Condicionais

- **if...**
- **if...else...**
- **if...else if...else...**

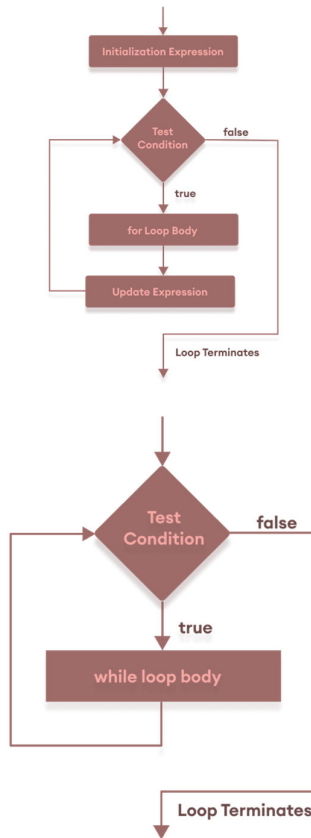


- **switch...case...**



> Estruturas de Repetição

- **for...**
- **for...in**
- ...



Vários exemplos da aplicação durante as aulas...

- **while...**
- **do...while**

