

Linguagens Script

<JavaScript>

Licenciatura em Engenharia Informática

Departamento de Engenharia Informática e de Sistemas

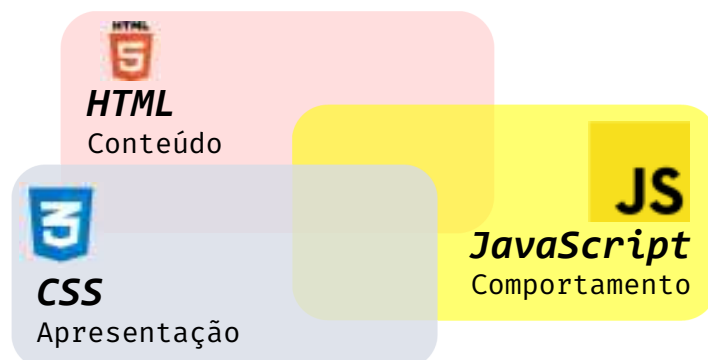
Cristiana Areias < cris@isec.pt >

2023/2024

> Introdução ao JavaScript

Introdução

- 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 (*Dynamic 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



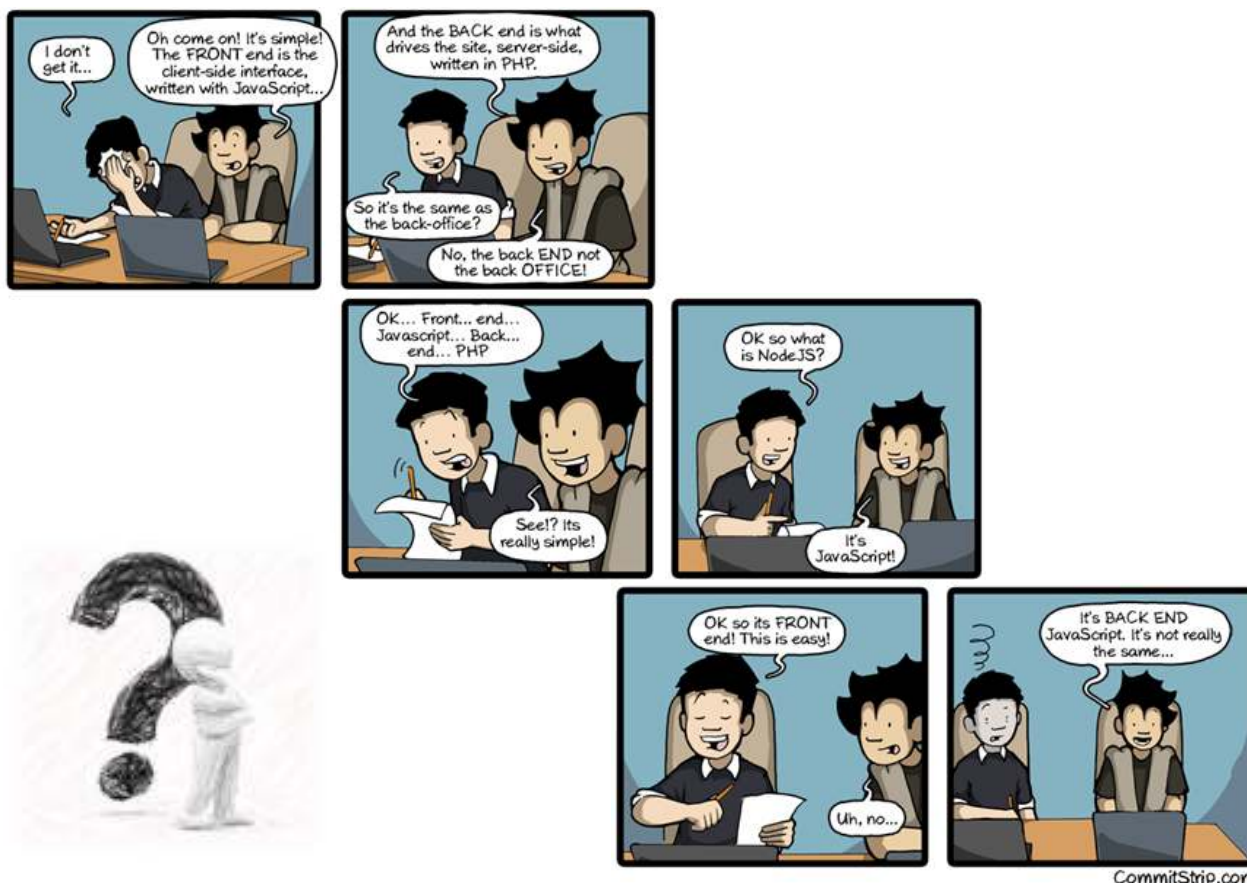
> Algumas Desvantagens...

- 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

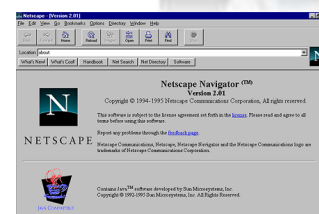


Cristiana Areias | Linguagens Script | 2023-2024

< 5 >

> 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



Cristiana Areias | Linguagens Script | 2023-2024

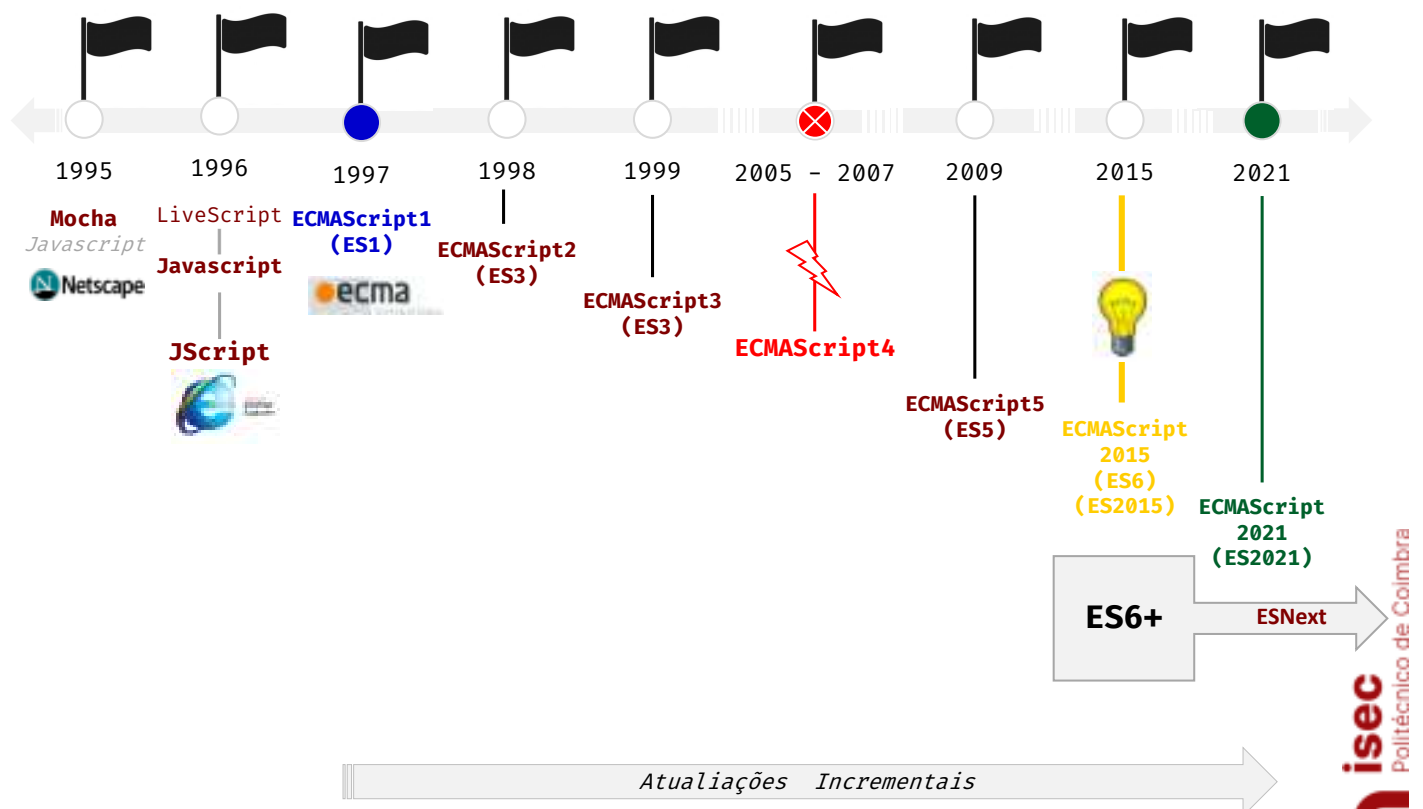
< 6 >

> 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 @ackyxmani

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

> Compatibilidade

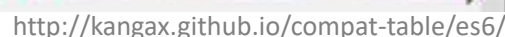
- **Babel**
 - Compilador / **Transpilador**
 - Converte código JavaScript atual (ECMAScript 2015+) para uma versão em que o browser, de versão mais antiga, o possa executar.
- Importante para *front-end developers*.

```
// 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/>



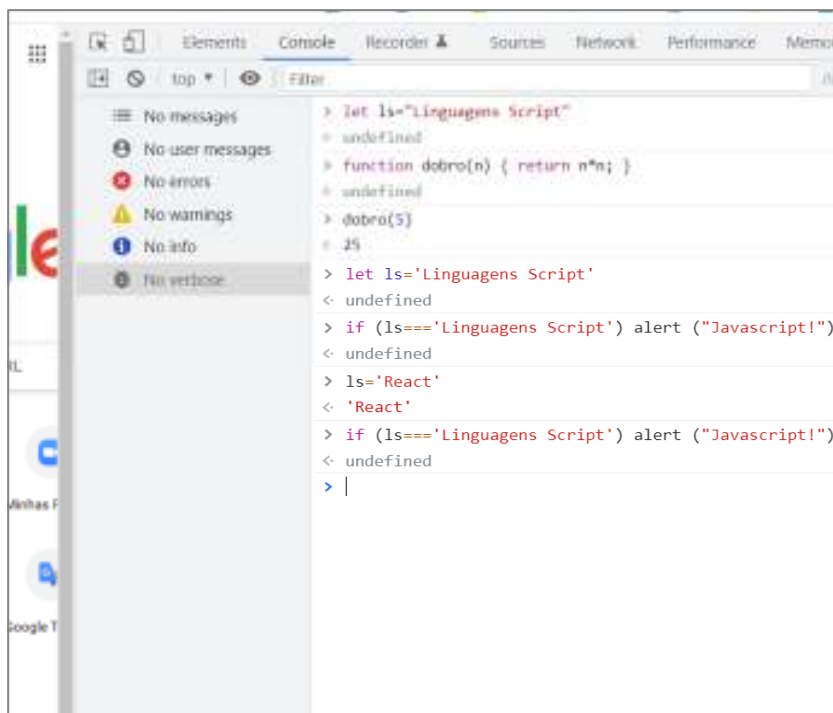


- 

< 16 >

> 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 linguagensScript **≠** Let linguagensscript
- Comentários
 - // símbolo do comentário
 - /* comentário para múltiplas linhas */
- Convenções de codificação em JavaScript
 - camelcase
 - linguagemScript
- Regras específicas
 - Não podem iniciar com número

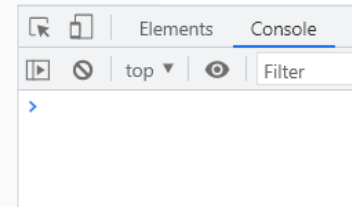
```
let strLS="Linguagens Script";  
let strJS="Javascript";  
  
let 2LS="LS";
```

> Strict Mode

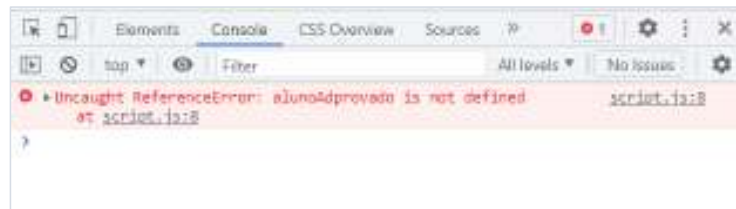
- Método mais rigoroso, evitando potenciais problemas de implementação;
- Facilita identificação de erros;
- Recomenda-se o uso do **strict mode** nas aulas práticas;

```
'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 >

2023/2024

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*, ...

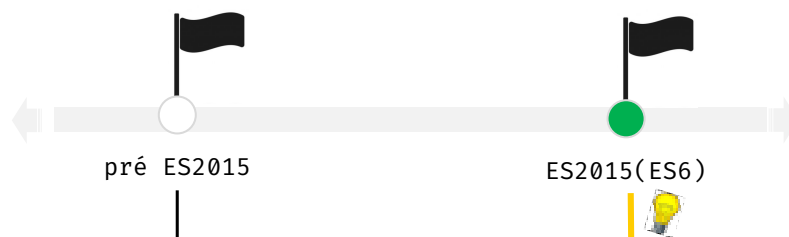
Scope
"The current context of execution. The context in which values and expressions are "visible" or can be referenced. If a variable or other expression is not "in the current scope," then it is unavailable for use.", [MDN](#)

▪ 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 undefined	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

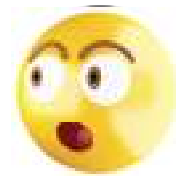
> Declaração variáveis > *hoisting*

```
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);
```



```
Olal
Olal
❌ ▶ 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);
```

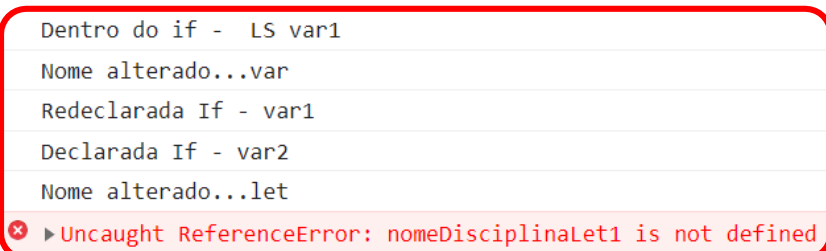
```
❌ ▶ Uncaught ReferenceError: a is not defined
```

> Variáveis > var vs let

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



```
var nomeDisciplinaVar1 = "Nome alterado...var";  
var nomeDisciplinaVar2 = "Redeclarada If - var1";  
let nomeDisciplinaLet1 = "Declarada If - var2";  
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);
```



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); 4  
x = x + 2;  
console.log(x); 6  
x = 2;  
console.log(x++); 2  
console.log(--x); 2  
console.log(x); 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**;

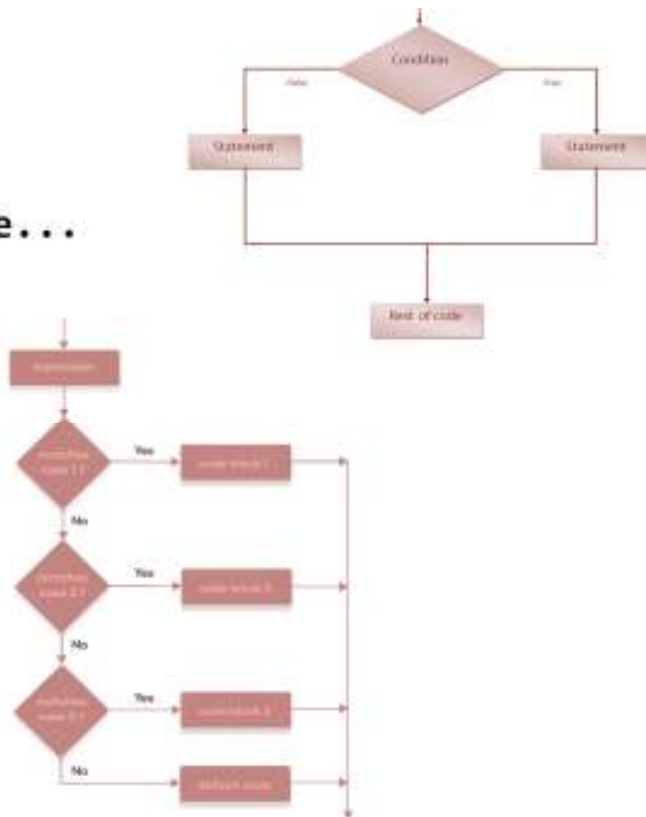
▪ Destructuring Assignment

- Permite atribuição via desestruturação;

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

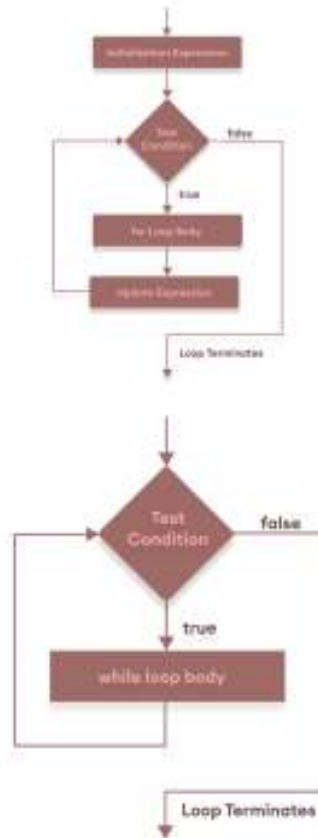
> Estruturas Condicionais

- if...
- if...else...
- if...else if...else...
- switch...case...



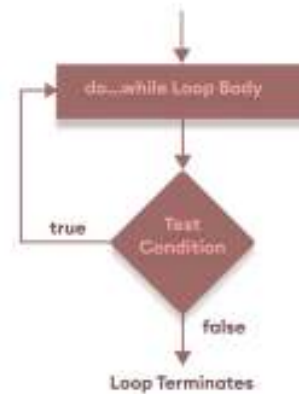
> Estruturas de Repetição

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



Vários exemplos da aplicação durante as aulas...

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



JavaScript

<Tipos de Dados>

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

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2023/2024

Tipos de Dados

- › **Primitivos** – *Value Types*
 - › *Boolean, String, Number,...*
- › **Objetos** – *Reference Types*
 - › *Objetos, Arrays, Funções*

< 48 >

> Tipos de Dados

Primitivos
Value Types

Objetos
Reference Types

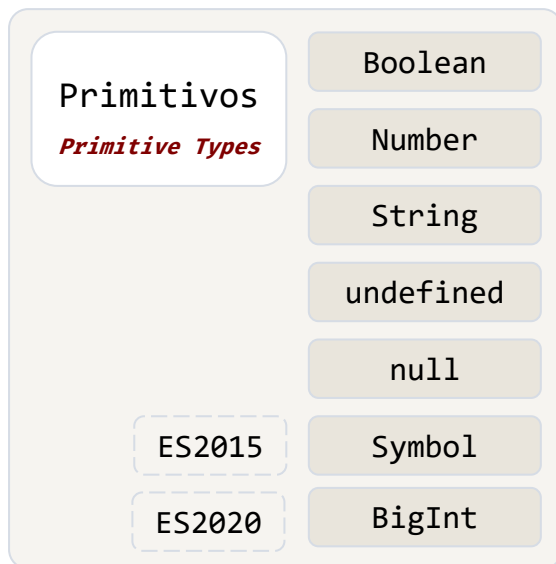
- Todo o valor é um valor **primitivo** ou um **objeto**.
 - Um **primitivo** é um dado que não é representado através de um objeto, por consequência, não possui métodos.
- Uma característica do JavaScript é **tipagem dinâmica**, logo:
 - não é necessário declarar o tipo da variável antes de ser atribuída;
 - Os tipos de dados são determinados de forma automática e podem alterar em tempo de execução;

```
let alunoAprovado = true;  
console.log(typeof alunoAprovado);  
  
alunoAprovado="José Meira";  
console.log(typeof alunoAprovado);
```

boolean
string

> Primitivos > Value Types

- A variável contém diretamente o **valor atribuído**;
- Todos os tipos primitivos são **imutáveis**;
- O mais recente padrão *ECMAScript* define sete tipos de dados primitivos;



```
let alunoAprovado = true;
let nota = 15;
let disciplina = "Linguagens Script";
let nomeAluno;
let alunoSeleccionado=null;
```

<https://developer.mozilla.org>

> Primitivos > Value Types

```
> let disciplina = "Linguagens Script"; // String literal
typeof disciplina;
< 'string'

> let nota = 15; // Number literal
typeof nota;
< 'number'

> let alunoAprovado = true;
typeof alunoAprovado;
< 'boolean'

> let nome;
typeof nome;
< 'undefined'

> let nome=undefined;
typeof nome;
< 'undefined'

> let nomeSeleccionado=null;
typeof nomeSeleccionado;
< 'object'
```

undefined – Indica variavel não inicializada
É um tipo mas também um valor

Indica um não valor, deliberadamente

objecto!

> Strings > Template Literals

- Para a declaração de *strings* recorre-se às aspas simples `' '` ou duplas `" "`;
- O ES6 introduziu um novo tipo de “template literals” que recorre ao uso dos acentos graves `` ``
 - Permitem que expressões básicas de interpolação de *strings* sejam incorporadas, depois analisadas e avaliadas automaticamente;

Pré ES6

```
var ls = "Linguagens Script";  
var info = "UC: " + ls + "!";  
console.log(info);  
console.log(typeof info);
```

ES6

```
var ls = "Linguagens Script";  
var info = `UC: ${ls}!`;  
console.log(info);  
console.log(typeof info);
```

```
UC: Linguagens Script!  
string
```

> Strings > Template Literals

- Permitirem especificar *strings* **multilinhas** sem necessidade de adicionar quebras de linha ou caracteres de escape como o `'\n'`

```
var ls = "Linguagens Script";  
var sem = 2;  
var info =  
`UC: ${ls}!
```

```
  Disciplina do 1º Ano - ${sem} Semestre
```

```
  Licenciatura em Engenharia Informática e de Sistemas
```

```
ISEC`;
```

```
console.log(info)
```

```
UC: Linguagens Script!  
Disciplina do 1º Ano - 2 Semestre  
Licenciatura em Engenharia Informática e de Sistemas  
ISEC
```

- Permitirem embutir **expressões**

> Primitivos - *Value Types*

```
let media = 15;
let mediaAntiga = media;
console.log(media);
console.log(mediaAntiga);
media = 12;
console.log(media);
console.log(mediaAntiga);
```



15
15
12
15

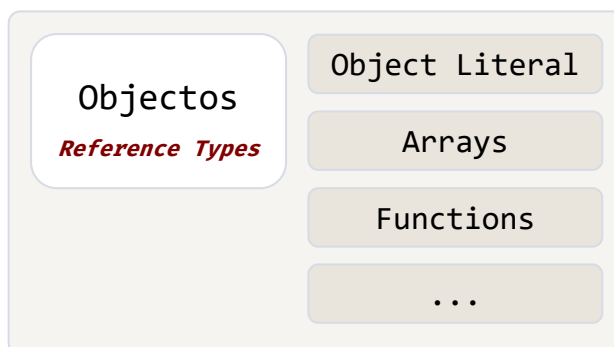
Localizações de
memória diferentes!



Frames

Global frame	
media	12
mediaAntiga	15

> Objetos - *Reference Types*

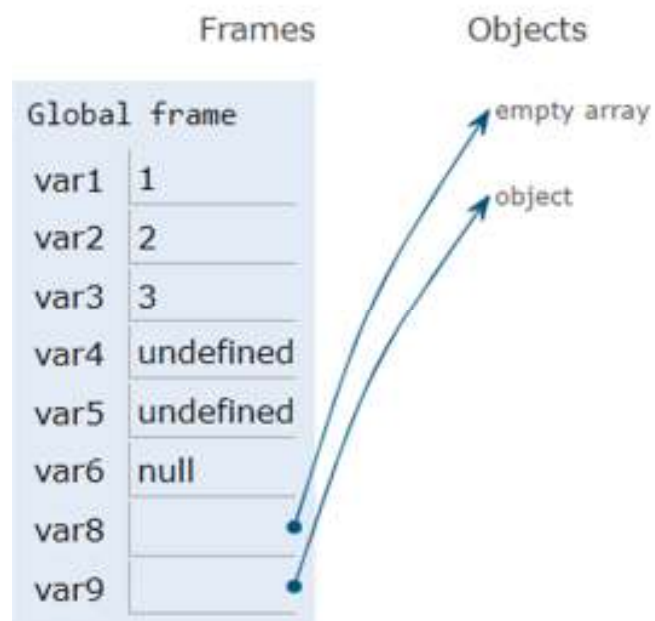


```
let varObj = {};
```

- Tudo o que não é primitivo, é **Objeto**!
- Objetos podem conter diferentes tipos de dados!
- Objetos, arrays, funções são tipos de dados **mutáveis**;
- Um objeto também pode ter funções (designados como métodos);
- Um objeto é composto por uma lista de pares entre propriedades e valores, associando uma chave com um valor;

> Dados > valor vs referência

```
var var1 = 1;  
let var2 = 2;  
const var3 = 3;  
let var4;  
let var5 = undefined;  
let var6 = null;  
let var8 = [];  
let var9 = {};
```



<https://pythontutor.com/javascript.html>

JavaScript <Objetos>

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

Departamento de Engenharia Informática e de Sistemas

Cristiana Areias < cris@isec.pt >

2023/2024

Objetos

- › O que são objetos?
- › Objetos em JavaScript
 - › Formas de Criação - Literal
 - › Acesso e alteração das propriedades

< 58 >

> O que é um Objeto?

- **Objetos** → pensar em situações do mundo real...
 - **Algo visível**: um autocarro, chave, saco, livro...
 - **Algo que não se pode tocar**: tempo, um evento, uma conta bancária...
- **Objetos** podem ser vistos como um conjunto de **Propriedades**
 - **Características do objeto**: cor, nº de páginas num livro,...
 - Descrevem o estado corrente de um objeto.
 - O **estado** de um objeto é independente de outro.
Exemplo: Um carro é amarelo e o outro é azul.
- O **Comportamento** refere-se ao que "permite fazer";
 - Numa conta bancária: um depósito, levantamento,...
- **Objetos são substantivos! Não são comportamentos nem propriedades!**



> Objetos em JavaScript

- Objetos em JavaScript podem ser vistos como simples *coleções* compostas por pares **chave e valor**

```
const aluno = {  
  nome: 'Manuel Afonso',  
  numero: 1232123  
}
```

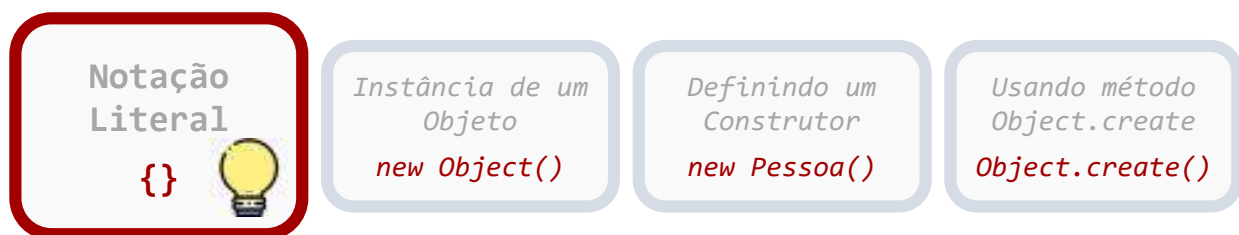
Chave Valor

```
const aluno = { nome: 'Manuel Afonso', numero: 1232123 }
```

- Comparativamente com outras linguagens, podem ser vistos como:
 - Hash tables* na linguagem C e C++
 - HashMap* na linguagem Java
 - Dicionários* na linguagem Python
 - ...

> Objetos > Criação

- Existem diferentes formas de se criar um **objeto**



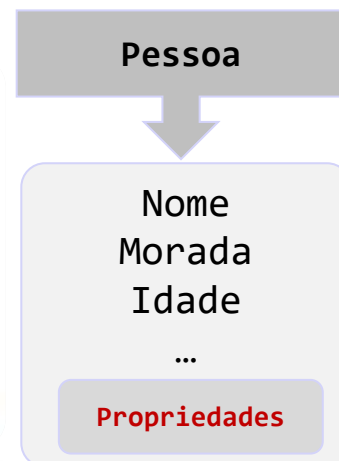
- É muito comum criar um **objeto** usando um literal de objeto quando se pretende transferir um conjunto de dados estruturados relacionados.
- A criação de um **objeto literal** é tipicamente utilizada **para criar um único objeto**, enquanto que com recurso a um construtor é útil para criar múltiplos objetos.



> Objetos > Criação

```
let nome;  
let morada;  
let idade;
```

Notação Literal



```
let pessoa = {  
  nome: 'Manuel Afonso',  
  morada: 'Rua Carlos Seixas',  
  idade: '45'  
}
```

propriedades

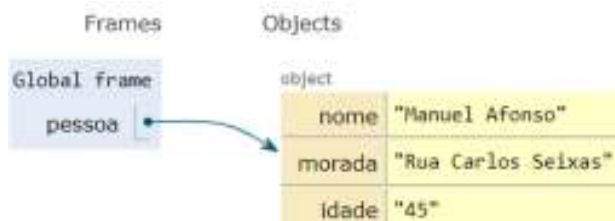
> Objetos > Criação

Notação Literal

```
let pessoa = {  
  nome: 'Manuel Afonso',  
  morada: 'Rua Carlos Seixas',  
  idade: '45'  
}
```



```
{nome: 'Manuel Afonso', morada: 'Rua Carlos Seixas',  
  idade: '45'}  
  idade: "45"  
  morada: "Rua Carlos Seixas"  
  nome: "Manuel Afonso"  
  ► [[Prototype]]: Object
```



> Objetos > Criação

```
const nuno = {  
  nome: 'Nuno Afonso',  
  numero: 2102124,  
  morada: 'Rua Nova, Coimbra',  
  disciplinasInscritas: ['AP', 'LS', 'TW', 'SO', 'AM', 'TAC']  
}
```

Vários tipos!

```
▼ {nome: 'Nuno Afonso', numero: 2102124, morada: 'Rua Nova, Coimbra', disciplinasInscritas: Array(6)}  
  ▼ disciplinasInscritas: Array(6)  
    0: "AP"  
    1: "LS"  
    2: "TW"  
    3: "SO"  
    4: "AM"  
    5: "TAC"  
    length: 6  
    ► [[Prototype]]: Array(0)  
  morada: "Rua Nova, Coimbra"  
  nome: "Nuno Afonso"  
  numero: 2102124  
  ► [[Prototype]]: Object
```

> Objetos > Propriedades

- As propriedades podem ser acedidas ou alteradas de duas formas:

Dot notation

Notação com .

`objectName.objectProperty`

```
console.log(nuno.nome)
```

```
nuno.nome = "Nuno Afonso";
```

Bracket notation

Notação com [

`objectName['objectProperty']`

```
console.log(nuno['morada'])
```

```
nuno['morada'] = "Rua Velha";
```


Qual método usar?

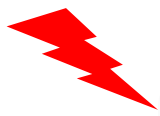
- › Dot Notation .
- › Bracket notation []



< 66 >

> Objetos > Acesso às Propriedades

JavaScript



```
const propriedade='morada';  
console.log(nuno.propriedade);  
console.log(nuno[propriedade]);
```

undefined

► ['Rua Velha, Coimbra']

```
console.log(`O aluno ${nuno.nome} com número ${nuno['numero']}  
está inscrito a ${nuno.disciplinasInscritas.length} disciplinas:  
${nuno.disciplinasInscritas}`);
```

```
O aluno Nuno Manuel Afonso com número 2102124  
está inscrito a 6 disciplinas:  
AP,LS,TW,SO,AM,TAC
```

> Objetos > Acesso às Propriedades

JavaScript

```
const aluno1 = {  
  nome: "Nuno Afonso",  
  media: 15,  
  cursoConcluido: false  
};
```

```
const aluno2=aluno1;  
aluno2.nome="Ricardo Afonso";
```

```
console.log(aluno1);  
console.log(aluno2);
```



```
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}  
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
```



> Objetos > Acesso às Propriedades

JavaScript

```
const aluno1 = {  
  nome: "Nuno Afonso",  
  media: 15,  
  cursoConcluido: false  
};
```

```
const aluno2=aluno1;  
aluno2.nome="Ricardo Afonso";
```

```
aluno2 = {};  
console.log(aluno2);
```

```
✖ ▶ Uncaught TypeError: Assignment to constant variable.
```



> Objetos > Cópia

```
const aluno1 = {  
  nome: "Nuno Afonso",  
  media: 15,  
  cursoConcluido: false  
};  
  
const aluno2 = Object.assign({},aluno1);  
console.log(aluno1);  
console.log(aluno2);  
  
aluno2.nome="Ricardo Afonso";  
console.log(aluno1);  
console.log(aluno2);
```



Tem limitações!

```
► {nome: 'Nuno Afonso', media: 15, cursoConcluido: false}  
► {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
```

> Objetos > Alteração Propriedades

```
const aluno = {  
  nome: "Nuno Afonso",  
  media: 15,  
  cursoConcluido: false  
};
```

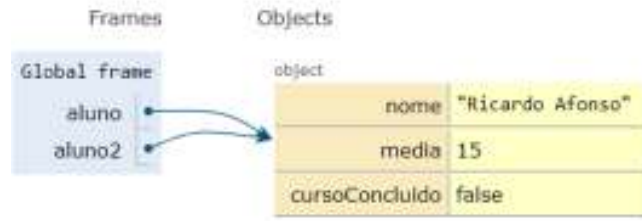


```
let aluno2 = aluno;  
console.log(aluno);  
console.log(aluno2);  
  
aluno2.nome = "Ricardo Afonso";  
  
console.log(aluno);  
console.log(aluno2);  
  
aluno2.novaProp = "Prop. Nova";  
console.log(aluno);  
console.log(aluno2);  
  
aluno2 = { nome: "Ana Afonso",  
  idade: 30 }  
console.log(aluno);  
console.log(aluno2);
```



> Objetos > Alteração Propriedades

```
let aluno2 = aluno;  
console.log(aluno);  
console.log(aluno2);
```



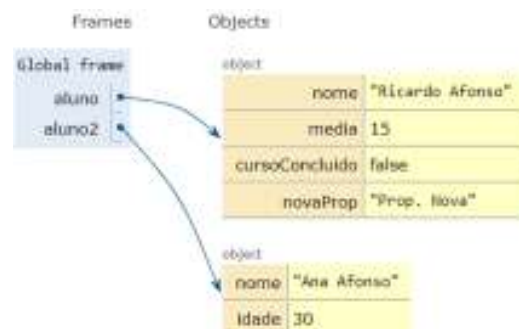
```
aluno2.nome = "Ricardo Afonso";  
console.log(aluno);  
console.log(aluno2);
```

```
▶ {nome: 'Nuno Afonso', media: 15, cursoConcluido: false}  
▶ {nome: 'Nuno Afonso', media: 15, cursoConcluido: false}  
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}  
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false}
```

> Objetos > Alteração Propriedades

```
aluno2.novaProp = "Prop. Nova";  
console.log(aluno);  
console.log(aluno2);
```

```
aluno2 = {  
  nome: "Ana Afonso",  
  idade: 30 }  
console.log(aluno);  
console.log(aluno2);
```



```
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false, novaProp: 'Prop. Nova'}  
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false, novaProp: 'Prop. Nova'}  
▶ {nome: 'Ricardo Afonso', media: 15, cursoConcluido: false, novaProp: 'Prop. Nova'}  
▶ {nome: 'Ana Afonso', idade: 30}
```

> Objetos - Reference Types

```
let x = 10;  
let obj = { valor: 20 };  
let y = obj;  
obj.valor = 30;
```

```
console.log(x);  
console.log(y);  
console.log(obj.valor);  
console.log(y.valor);
```

