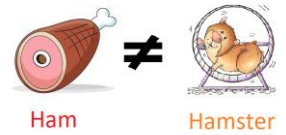
UPT UNIVERSIDADE PORTUCALENSE

CONTEÚDO

1. Contexto
2. Sintaxe
3. Tipos de dados
    1. Primitivos
    2. Não primitivos
    3. Conversão entre tipos de dados
4. Operadores
    1. Aritméticos
    2. Atribuição
    3. Comparação
    4. Lógicos
5. Estruturas
    1. Condicionais
    2. Cíclicas
6. Funções
    1. Sintaxe
    2. Retorno
    3. Âmbito de variáveis
    4. Funções de interação
7. Arrays
    1. Sintaxe
    2. Adicionar elementos
    3. Eliminar elementos
    4. Iteração
    5. Ordenação
    6. Métodos de pesquisa
    7. Métodos de transformação
8. Objetos:
    1. Criação e propriedades
    2. Aceder e alterar propriedades
    3. Métodos
    4. Cópia de objetos
9. Classes
    1. Sintaxe
    2. Exemplo

Contexto



- **Java ≠ JavaScript**
- **JavaScript** criado em 1995 por Brendan Eich para o Netscape Navigator 2
- Começou por chamar-se LiveScript mas como o Java era muito popular na altura, mudou-se o nome
- Em 1997 tornou-se uma especificação da *European Computer Manufacturers Association* (ECMA)
    - **ECMAScript:** https://www.ecma-international.org/publications/standards/Ecma-262.htm
        - última versão: ECMA-262, 10ª edição, junho 2019
- **JavaScript** é uma linguagem de *scripting* que está em conformidade com a especificação **ECMAScript**
- Popularizou-se como uma linguagem para navegadores web, mas hoje já é utilizada noutras circunstâncias (ex: Node.js)

- Consola de JavaScript: em qualquer *browser*, **Ctr + Shift + I**

Sintaxe

```
// comentário
```

```
/* isto também
é um comentário */
```

```
let username = "Maria" // o valor da variável username é "Maria"

{
let x = 1 // x é inicializado com o valor 1
x = 2 // x passa a ter o valor 2
let x = 3 // erro: o identificador x já foi utilizado
}

let x = 1 // x tem o valor 1
{
let x = 2 //já não dá erro. x tem o valor 1
}
// x tem o valor 1
```

As variáveis decalaradas com `let` têm âmbito local (bloco)

Nomes de variáveis

- Podem conter: letras, dígitos, símbolos ($, _).
- 1º carater não pode ser um número
- São *case-sensitive*
- Não podem ser palavras reservadas

A variável x só tem valor 2 dentro do bloco

```
const anoNacimento = 2000
anoNascimento = 2001 // erro: não é permitido alterar o valor de constantes
```

Tipos de dados primitivos

O JavaScript é uma linguagem dinamicamente tipada: uma variável pode conter qualquer tipo de dados. Exemplo:

```
let message = "olá"
message = 123 // é válido mudar o tipo da variável
```

Tipos de dados primitivos: `typeof x` `typeof(x)`

- **number**: números inteiros e decimais
  - Operações: +, -, *, /
  - Valores especiais: `Infinity (1/0), -Infinity (-1/0), NaN ("olá"*3)`

```
"2" * 3 // 6
```

```
isNaN("ola"*3) // true
```

- **string**: valores alfanuméricos
  - Backticks

```
const name = "Maria"
`Olá ${name}!` //Resultado: Olá Maria!
```

  - Concatenação (+)

```
"1" + 2 // 12
```

- **boolean**: true / false
- **null** (desconhecido)
- **undefined** (não atribuído)

Tipos de dados não primitivos

- **object**: estruturas de dados mais complexas

```
let person = {
     firstName: "Maria",
     lastName: "Sousa",
     age: 37,
     favouriteColor: "purple"
}
```

- **symbol**: identificadores exclusivos
    - Pode ser usado para identificar propriedades de objetos

Conversão entre tipos de dados

- Para String

```
let age = String(12)      // "12"
```

- Para Número

```
let age = Number(12)      // 12
Number("   123   ")       // 123
Number("casa")            // NaN
Number("")                // 0
Number("true")            // 1
Number("false")           // 0
```

- Para Booleano

```
let eyes
Boolean(eyes)             // false
eyes = 2
Boolean(eyes)             // true
```

Operadores

Em JavaScript há vários tipos diferentes de operadores:

- Aritméticos: Lidar com cálculos
- Atribuição: Atribuir valores a variáveis
- Comparação: Determinar igualdade entre variáveis ou valores
- Lógicos: Determinar lógica entre variáveis ou valores

Operadores Aritméticos

Operador	Descrição	Exemplo
`+, -, *, /`	Soma, subtração, multiplicação, divisão	`3+2 // 5`
`%`	Módulo (resto da divisão)	`12%10 // 2`
`**`	Exponenciação	`3**2 // 9`
`++, --`	Incremento, decremento	`let y=3; y++; y // 4`

Operadores de Atribuição

Operador	Descrição	Exemplo
`=`	Atribuição simples	`let x = 3 // 3`
`+=, -=, *=, /=, %=`	Atribuição com operação	`let x = 3; x+=2; x // 5`

Operadores de Comparação

Operador	Descrição	Exemplo

==	Igual em valor	2==2 // true
		2==3 // false

===	Igual em valor e em tipo	2===2 // true
		2===“2” // false

!=	Diferente em valor	2!=2 // false 2!=3 // true
!==	Diferente em valor ou em tipo	2!==2 // false 2!==“2” // true 2!==3 // true
>, >= <, <=	Maior, maior ou igual, menor, menor ou igual	2>2 // false 2>=2 // true 2>1 // true “Porta” > “Porto” // false
?	Operador ternário	Math.PI > 4 ? ”S" : ”N"; // ”N”

Operadores Lógicos

Operador	Descrição	Exemplo

&&	E lógico	`false && false  // false`
		`false && true  // false`
		`true  && false  // false`
		`true  && true  // true`

||	OU lógico	`false || false  // false`
		`false || true  // true`
		`true  || false  // true`
		`true  || true  // true`

!	NÃO lógico	`!true  // false` `!false  // true`

Estruturas Condicionais

if

```
if(condição){
  instruções
}
```

```
if(hour <= 12){
  greeting = "Bom dia"
}
```

if / else

```
if(condição){
  instruções-if
} else {
  instruções-else
}
```

```
if(hour <= 12){
  greeting = "Bom dia"
} else {
  greeting = "Boa tarde"
}
```

if / else if / else

```
if(condição){
  instruções-if
} else if (condição2) {
  instruções-if2
} else {
  instruções-else
}
```

```
if(hour <= 12){
  greeting = "Bom dia"
} else if(hour<=20){
  greeting = "Boa tarde"
} else {
  greeting = "Boa noite"
}
```

switch

```
switch(valor){
  case …: instruções; break;
  ...
}
```

```
switch(new Date().getDay(){
  case 0: day = "Sunday"; break;
  case 1: day = "Monday"; break;
  case 2: day = "Tuesday"; break;
  case 3: day = "Wednesday"; break;
  case 4: day = "Thursday"; break;
  case 5: day = "Friday"; break;
  case 6: day = "Saturday"; break;
}
```

Estruturas cíclicas

for

```
for(inicialização; condição; atualização) {
  instruções
}
```

```
for(let i=0; i<3; i++){
  alert(i)
}
```

do-while

```
do {
  instruções
} while(condição)
```

```
let i=0
do {
  alert(i)
  i++
} while(i<3)
```

while

```
while(condição) {
  instruções
}
```

```
let i=0
while(i<3){
  alert(i)
  i++
}
```

Funções

- **Função JavaScript**: bloco de código para executar uma determinada tarefa e que pode ser reutilizado
- O código fica mais conciso, legível e de fácil manutenção

- Sintaxe:

```
function nome (param1, param2,  …) {
     código a ser executado
}
```

- Exemplo: declaração de função

```
function mostraMensagem () {
     let mensagem = "Olá!"
     console.log(mensagem)
}
```

- Exemplo: invocação de função

```
mostraMensagem() // Resultado: Olá!
```

Retorno de funções

```
function soma(a,b) {
    return a + b
}
```

```
let resultado1 = soma(4,3)
resultado1 // 7
let resultado2 = soma(4)
resultado2 // NaN
```

```
function soma2(a,b = 0) {
    return a + b
}
```

```
let resultado3 = soma2(4)
resultado3 // 4
```

Âmbito de variáveis

Global

```
let nome = "Ana"
function mostraMensagem() {
  nome = "Maria"
  let mensagem = `Olá ${nome}`
  console.log(mensagem)
}

console.log(nome) // Ana
mostraMensagem()  // Olá Maria
console.log(nome) // Maria
```

Local

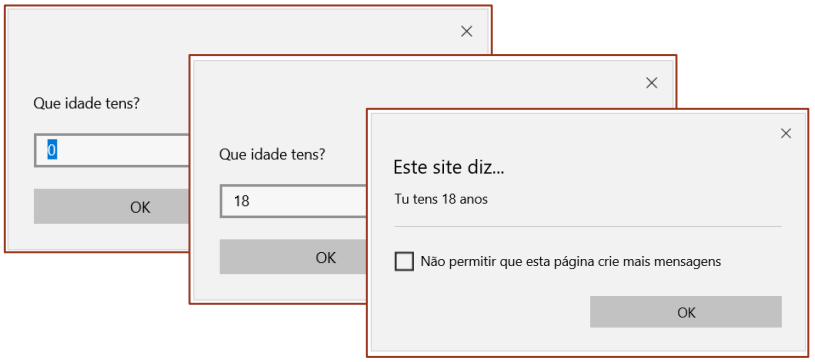
```
let nome = "Ana"
function mostraMensagem() {
  let nome = "Maria"
  let mensagem = `Olá ${nome}`
  console.log(mensagem)
}

console.log(nome) // Ana
mostraMensagem()  // Olá Maria
console.log(nome) // Ana
```

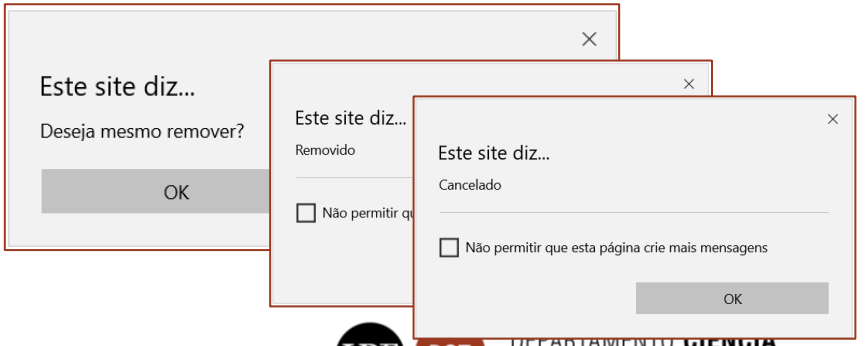
Funções de interação

```
alert("Olá")
```


```
let idade = prompt("Que idade tens?",0)
if(idade!=null) {
  alert(`Tu tens ${idade} anos`)
}
```



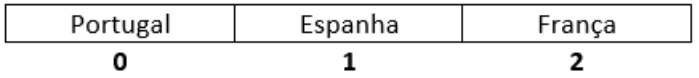

```
let removerRegisto = confirm("Deseja mesmo remover?")
if(removerRegisto) {
  alert("Removido")
} else {
  alert("Cancelado")
}
```



Arrays

- Declaração

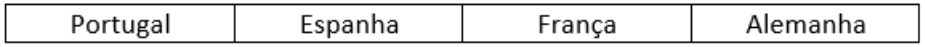
```
let paises = new Array()
let paises = []
let paises = [“Portugal”,”Espanha”,”Franca”]
```



- Operações

```
console.log(paises.length) // 3
console.log(países[1])     // Espanha
console.log(paises())      // [object Array]: ["Portugal", "Espanha", "França"]

let outro = paises             // copia os arrays por referência
console.log(outro === paises) // true
outro[3] = “Alemanha”          // modifica o array
console.log(paises)            // [object Array]:["Portugal","Espanha","França“,”Alemanha”]
```



Manipulação de arrays: adicionar elementos

Portugal	Espanha	França
0	1	2

Exemplo: adicionar elementos

`paises[3] = "Alemanha"`

Portugal	Espanha	França	Alemanha
0	1	2	3

`paises[5] = "Itália"`

Portugal	Espanha	França	Alemanha	*undefined*	Itália
0	1	2	3	4	5

Operações úteis

Portugal	Espanha	França
0	1	2

- **Push**: adiciona elementos ao final do array. Retorna o novo tamanho.

```
paises.push("Alemanha") //4
```

Portugal	Espanha	França	Alemanha
0	1	2	3

- **Unshift**: adiciona elementos no início do array. Os outros elementos andam uma posição para a frente. Retorna o novo tamanho.

```
paises.unshift("Itália") // 5
```

Itália	Portugal	Espanha	França	Alemanha
0	1	2	3	4

- **Splice**: adiciona/remover elementos ao/do array sem criar "buracos"
  - sintaxe: `nomeDoArray.splice(índice, 0, elementosAdicionar1, elementosAdicionar2, …)`

```
paises.splice(1,0,"Polónia","Marrocos") // []
```

Itália	Polónia	Marrocos	Portugal	Espanha	França	Alemanha
0	1	2	3	4	5	6

Nota: é preferível fazer operações no final do array (mais rápidas)

Manipulação de arrays: eliminar elementos

Portugal	Espanha	França
0	1	2

Exemplo: eliminar elementos

```
delete paises[1]
```

Portugal	*undefined*	França
0	1	2

Operações úteis

Itália	Polónia	Marrocos	Espanha	França	Alemanha
0	1	2	3	4	5

- **Pop**: remove o último elemento do array. Retorna o elemento.

```
paises.pop() // Alemanha
```

Itália	Polónia	Marrocos	Espanha	França
0	1	2	3	4

- **Shift**: remove o primeiro elemento do array. Os outros elementos andam uma posição para trás. Retorna o elemento.

```
paises.shift() // Itália
```

Polónia	Marrocos	Espanha	França
0	1	2	3

- **Splice**: adiciona/remover elementos ao/do array sem criar “buracos”
  - sintaxe: `nomeDoArray.splice(índice, nrElementosRemover)`

```
paises.splice(1,1) // Marrocos
```

Polónia	Espanha	França
0	1	2

```
paises.splice(0,2) // Polónia, Espanha
```

França
0

Nota: é preferível fazer operações no final do array (mais rápidas)

Outros métodos de inserção / remoção de elementos em arrays

```
let paises = [“Portugal”,“Espanha”,“França”,“Itália”,“Alemanha”]
```

Método	Descrição	Exemplo

`slice(inicio, fim)`	Cria um novo array com os elementos entre as posições **inicio** e **fim**	`let menosPaises = paises.slice(1,3)`
		`console.log(menosPaises)`
		`// "Espanha", "França"`

`concat(…items…)`	Junta novos items ao array	`menosPaises.concat("Marrocos")` `// "Espanha", "França", "Marrocos"`

Iteração

```
let paises = [“Portugal”,“Espanha”,“França”]
```

```
let tamanhoPaises = paises.length
for(let i=0; i<tamanhoPaises; i++) {
  console.log(paises[i])
}
```

```
for(let indice in paises) {
  console.log(paises[indice])
}
```

```
for(const pais of paises) {
  console.log(pais)
}
```

```
paises.forEach((item, index, array) => {
  alert(`${item} está na posição ${index} do ${array}`)
});
```

Ordenação

```
let paises = ["Portugal","Espanha","França"]
paises.sort()  // "Espanha", "França", "Portugal‟ (ordem alfabética)
paises.reverse() // "Portugal", "França", "Espanha"


let numeros = [12,7,10,18,5]
numeros.sort() // 10, 12, 18, 5, 7 (ordem alfabética)


function ascOrder(a, b) {
  return a - b
}
numeros = [12,7,10,18,5]
numeros.sort(ascOrder) // 5, 7, 10, 12, 18 (ordem crescente)

numeros.sort(function(a,b){return a - b}) // 5, 7, 10, 12, 18 (compacta)
```

Fornece uma ordem alternativa

Métodos de pesquisa em arrays

```
let paises = ["Portugal","Espanha","França","Itália","Alemanha"]
let numeros = [1,2,3,4,5,4,3,2,1,2,3,4,5]
let idades = [12,20,40,15,17,31]
```

```
function eAdulto(idade){
  return idade>=18
}
```

Método	Descrição	Exemplo

`indexOf(elemento,pos)`	Procura pelo **elemento** no array a partir de **pos** e retorna o índice ou -1	`paises.indexOf("Espanha")  // 1`
		`paises.indexOf("Inglaterra")  // -1`
		`numeros.indexOf(2)  // 1`
		`numeros.indexOf(2,3)  // 7`

`lastIndexOf(elemento)`	Procura pelo **elemento** no array e retorna o índice da última ocorrência ou -1	`numeros.lastIndexOf(2)  // 9` `numeros.lastIndexOf(7)  // -1`
`includes(elemento)`	Determina se o **elemento** se encontra no array	`numeros.includes(3)  // true` `numeros.includes(7)  // false`
`find(função)`	Procura elementos através da **função** e retorna o 1º elemento	`idades.find(eAdulto)  // 20`
`filter(função)`	Procura elementos através da **função** e retorna todos os elementos	`idades.filter(eAdulto) // [20, 40, 31]`

Métodos de transformação de arrays

```
let numeros = [1,2,3,4,5]
```

```
function mult2(nr){
  return nr*2
}

function soma (acumulado,atual){
  return acumulado + atual
}
```

Método	Descrição	Exemplo
`map(função)`	Aplica a **função** a todos os elementos do array e retorna o resultado	`numeros.map(mult2)  // [2,4,6,8,10]`
`fill(valor, inicio, fim)`	Preenche as posições do array de **início** a **fim** com **valor**	`numeros.fill(9,1,3)  // [1,9,9,4,5]` `numeros.fill(9)  // [9,9,9,9,9]`
`join(separador)`	Une os elementos do array numa string, separados pelo **separador**	`numeros.join("*")  // 1*2*3*4*5`
`reduce(função, valorIni)`	Aplica a **função**[(1)] a partir de um **valorIni** para reduzir o array a um único elemento	`numeros.reduce(soma) // 15`

[(1)]A função é do tipo:

```
function(acumulador, corrente, índice, array){…}
```

Objetos: criação e propriedades

Criação de objetos

```
let utilizador1 = {}

let utilizador2 = new Object()
```

Objetos vazios

```
let utilizador3 = {
  nome: "Maria",
  idade: 39
}
```

Propriedades

pares `chave: valor`

```
let nome = "Ana"
let idade = 41
let utilizador4 = {nome,idade}
```

Criação de propriedade

```
let novaPropriedade = "genero"
let utilizador5 = {
  [novaPropriedade]: "masculino"
}
```

Eliminação de propriedade

```
delete utilizador3.idade
console.log(utilizador3)
          // {nome: "Maria Matos"}
```

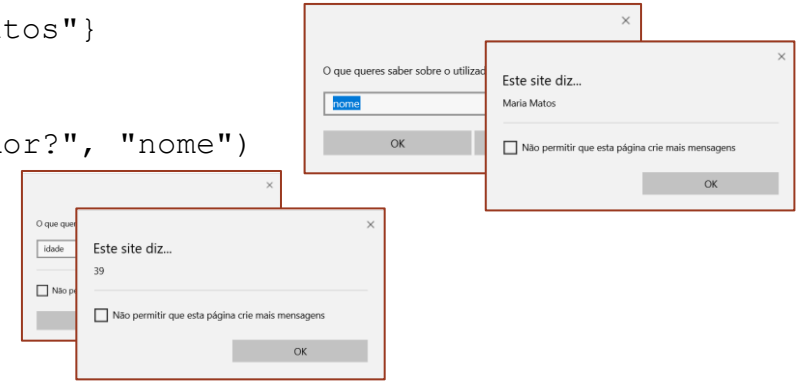
Objetos: aceder e alterar propriedades

Aceder / alterar propriedades de um objeto

```
console.log(utilizador3.nome) // “Maria”

utilizador3.nome = “Maria Matos”
console.log(utilizador3) // {idade: 39, nome: "Maria Matos"}

let chave = prompt("O que queres saber sobre o utilizador?", "nome")
alert(utilizador3[chave])
```

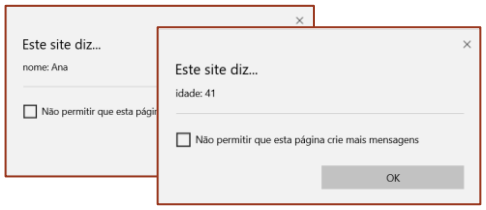


Verificar se propriedade existe

```
console.log(“idade” in utilizador4)    // true
console.log(“corOlhos” in utilizador4) // false
```

Percorrer todas as chaves do objeto

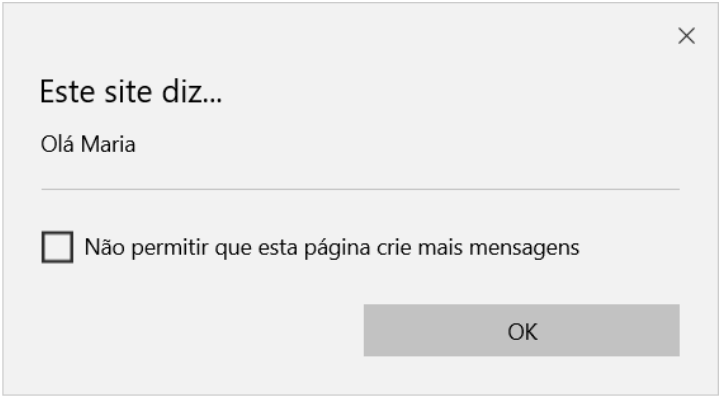
```
for(let chave in utilizador4){
  alert(chave + “: “ + utilizador4[chave])
}
```



Objetos: métodos

```
let utilizador = {
  nome: "Maria",
  idade: 39,
  dizerOla() {
    alert(`Olá ${this.nome}`)
  }
}

utilizador.dizerOla()
```



Cópia de objetos

```
let pessoa = {nome = "Ana"}
```

- Cópia é feita por referência

```
let novaPessoa = pessoa
novaPessoa.nome = "Maria"
console.log(novaPessoa.nome) // "Maria"
console.log(pessoa.nome) // "Maria" (altera também  em pessoa)
```

- Para copiar sem ser por referência

```
let novaPessoa ={}
// copiar todas as propriedades de pessoa
for(let chave in pessoa) {
  novaPessoa[chave]  = pessoa[chave]
}
novaPessoa.nome = "Maria"
console.log(novaPessoa.nome) // "Maria"
console.log(pessoa.nome) // "Ana" (mantém-se)
```

- Outra forma de copiar sem ser por referência

```
let novaPessoa = Object.assign({},pessoa)
novaPessoa.nome = "Maria"
console.log(novaPessoa.nome) // "Maria"
console.log(pessoa.nome) // "Ana" (mantém-se)
```

Classes

- Usadas quando necessitamos de criar vários objetos

- Sintaxe

```
class MinhaClasse {
  construtor(…){…}
  metodo1(…){…}
  metodo2(…){…}
  get propriedade(…){…}
  set propriedade(…){…}
  static metodoStatic(…){…}
  …
}
```

Primeira letra maiúscula

- Criação de um objeto da classe

```
let meuObjeto = new MinhaClasse(…)
```

Classes

- Exemplo

```
class Pessoa{
  constructor(nome, idade) {
    this.nome = nome
    this.idade = idade
  }
  get nome(){
    return this._nome
  }
  set nome(valor){
    this._nome = valor
  }
  dizerOla() {
    alert(`Olá ${this._nome}`)
  }
  static comparar(pessoaA,pessoaB) {
    return pessoaA.idade - pessoaB.idade
  }
}
```

_ : variável interna

```
let pessoa1 = new Pessoa("Maria",39)
pessoa1.dizerOla()
```

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.