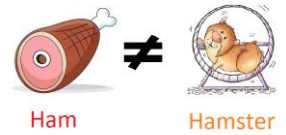
UNIVERSIDADE PORTUCALENSE

CONTENT

1. Context
2. Syntax
3. Data types
    1. Primitive
    2. Non primitive
    3. Type Conversion
4. Operators
    1. Arithmetic
    2. Assignment
    3. Comparison
    4. Logical
5. Structures
    1. Conditional
    2. Cycles
6. Functions
    1. Syntax
    2. Return
    3. Variable scope
    4. User interface
7. Arrays
    1. Syntax
    2. Adding elements
    3. Removing elements
    4. Iterating
    5. Ordering
    6. Search methods
    7. Transformation methods
8. Objects:
    1. Creation and properties
    2. Accessing and changing properties
    3. Methods
    4. Object copy
9. Classes
    1. Syntax
    2. Example

Context



- **Java ≠ JavaScript**
- **JavaScript** created in 1995 by Brendan Eich for the Netscape Navigator 2
- Its first name was LiveScript but because of Java's popularity at the time, the name was changed
- In 1997 it became a *European Computer Manufacturers Association* (ECMA) standard
    - **ECMAScript:** https://www.ecma-international.org/publications/standards/Ecma-262.htm
        - Last version: ECMA-262, 10th edition, June 2019
- **JavaScript** é is a *scripting* language that follows the **ECMAScript** standard
- It became popular as a programming language used for browsers, but it is now also used for other purposes (e.g: Node.js)

- To see JavaScript console in any browser: **Ctrl + Shift + I**

Syntax

```
// comment
```

```
/* another
comment */
```

```
let username = "Maria" // the value in variable username is "Maria"

{
let x = 1 // x is initialised with 1
x = 2 // x now has the value 2
let x = 3 // error: x identifier was already used
}
```

Variables declared with `let` have local scope (in the block)

Variable names

- May contain characters, numbers, symbols ($, _).
- The 1st character can not be a number
- Are *case-sensitive*
- Can not be reserved words (JavaScript keywords)

```
let x = 1 // x is initialised with 1
{
let x = 2 // no error. x now has the value 2
}
// x has the value 1 again
```

The variable x only has the value 2 inside the block

```
const yearOfBirth = 2000
yearOfBirth = 2001 // error: constant values can not be changed
```

Primitive data types

JavaScript is a dynamically typed language: a variable may contain data in any type. Example:

```
let message = "hello"
message = 123 // changing the data type is valid
```

Primitive data types: `typeof x` `typeof(x)`

- **number**: integer and decimal numbers
  - Operations: +, -, *, /
  - Special values: `Infinity (1/0)`, `-Infinity (-1/0)`, `NaN ("hello"*3)`

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

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

- **string**: sets of characters (words, text)
  - Backticks

```
const name = "Maria"
`Hello ${name}!` //Result: Hello Maria!
```

  - Concatenating (+)

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

- **boolean**: true / false
- **null** (unknown)
- **undefined** (not yet defined)

Non primitive data types

- **object**: more complex data structures

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

- **symbol**: exclusive identifiers
  - May be used to identify objects' properties

Type conversion

- To String

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

- To number

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

- To Boolean

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

Operators

In JavaScript there are several different operator types:

- Arithmetic: used for arithmetic operations
- Assignment: used to assing values to the variables
- Comparison: used to determine equalities between variables or values
- Logical: used to determine logical properties between variables or values

Arithmetic operators

Operator	Description	Example
`+, -, *, /`	Sum, subtraction, multiplication, division	`3+2 // 5`
`%`	Module	`12%10 // 2`
`**`	Power	`3**2 // 9`
`++, --`	Increment, decrement	`let y=3; y++; y // 4`

Assignment operators

Operator	Description	Example
`=`	Simple assignment	`let x = 3 // 3`
`+=, -=, *=, /=, %=`	Assignment with operations	`let x = 3; x+=2; x // 5`

Comparison operators

Operator	Description	Example

==	Equals (value)	2==2 // true
		2==3 // false

===	Equals (value and type)	2===2 // true
		2===“2” // false

!=	Different (value)	2!=2 // false 2!=3 // true
!==	Different (value or type)	2!==2 // false 2!==“2” // true 2!==3 // true
>, >= <, <=	Greater than, grater or equal, less than, less or equal	2>2 // false 2>=2 // true 2>1 // true “Porta” > “Porto” // false
?	Ternary operator	Math.PI > 4 ? ”S" : ”N"; // ”N”

Logical operator

Operator	Description	Example

`&&`	Logical AND	`false && false  // false`
		`false && true  // false`
		`true  && false  // false`
		`true  && true  // true`

`||`	Logical OR	`false || false  // false`
		`false || true  // true`
		`true  || false  // true`
		`true  || true  // true`

`!`	Logical NOT	`!true  // false` `!false  // true`

Conditional structures

if

```
if(condition){
  instructions
}
```

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

if / else

```
if(condition){
  instructions-if
} else {
  instructions-else
}
```

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

if / else if / else

```
if(condition){
  instructions-if
} else if (condição2) {
  instruções-if2
} else {
  instructions-else
}
```

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

switch

```
switch(value){
  case …: instructions; break;
  ...
}
```

```
switch(new Date().getDay(){
  case 0: day = "Domingo"; break;
  case 1: day = "Segunda"; break;
  case 2: day = "Terça"; break;
  case 3: day = "Quarta"; break;
  case 4: day = "Quinta"; break;
  case 5: day = "Sexta"; break;
  case 6: day = "Sábado"; break;
}
```

Cycle structures

for

```
for(initialization; condition; update) {
  instructions
}
```

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

do-while

```
do {
  instructions
} while(condition)
```

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

while

```
while(condition) {
  instructions
}
```

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

Functions

- **JavaScript function**: block of code designed to perform a specific task and can be reused
  - The code becomes smaller, more readable and easier to maintain

- Syntax:

```
function name (param1, param2,  …) {
     code to execute
}
```

- Example: declaring a function:

```
function showMessage () {
     let mensage = "Olá!"
     console.log(mensage)
}
```

- Example: calling a function:

```
showMessage() // Result: Olá!
```

Return

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

```
let result1 = sum(4,3)
result1 // 7
let result2 = sum(4)
result2 // NaN
```

```
function sum2(a,b = 0) {
    return a + b
}
```

```
let result3 = sum2(4)
result3 // 4
```

Variable scope

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

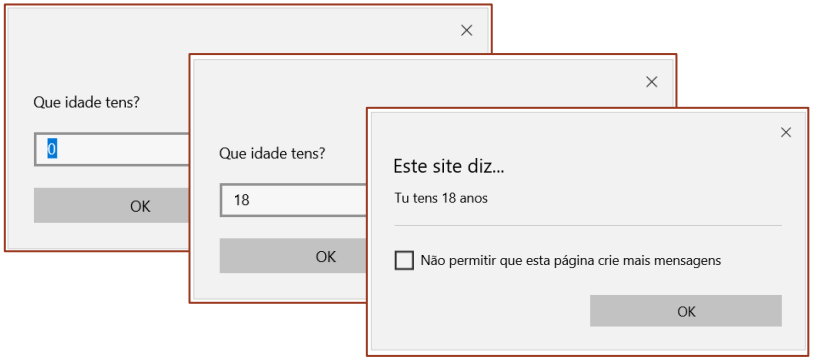
```
let nome = "Ana"
function mostraMensagem() {
  let nome = "Maria"
  let mensagem = `Olá ${nome}`
  console.log(mensagem)
}

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

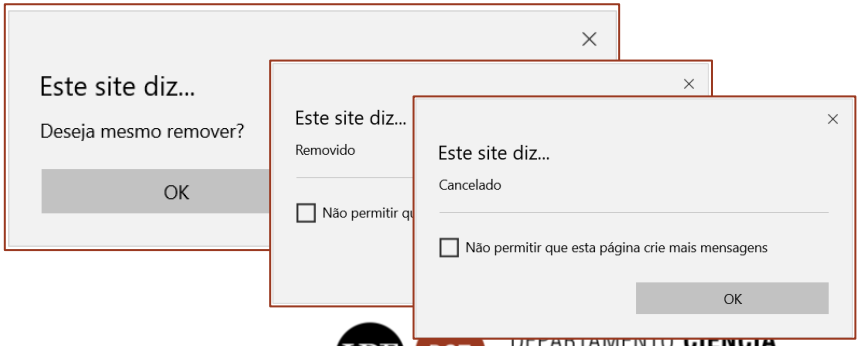
User interface functions

```
alert("Hello")
```


```
let idade = prompt("How old are you?",0)
if(idade!=null) {
  alert(`You are ${idade} years old`)
}
```



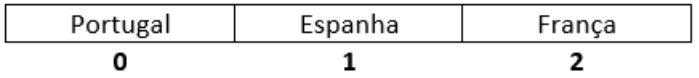

```
let removerRegisto = confirm("Remove?")
if(removerRegisto) {
  alert("Removed")
} else {
  alert("Cancelled")
}
```



Arrays

- Declaration

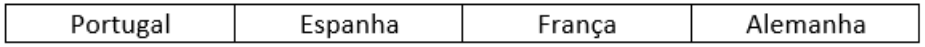
```
let countries = new Array()
let countries = []
let countries = [“Portugal”,”Espanha”,”França”]
```



- Operations

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

let outro = countries              // copy the arrays by reference
console.log(outro === countries) // true
outro[3] = “Alemanha”              // modifies the array
console.log(countries)             // [object Array]:["Portugal","Espanha","França“,”Alemanha”]
```



Array manipulation: adding elements

Portugal	Espanha	França
0	1	2

Example: add elements

`countries[3] = "Alemanha"`

Portugal	Espanha	França	Alemanha
0	1	2	3

`countries[5] = "Itália"`

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

Useful operations

Portugal	Espanha	França
0	1	2

- **Push**: adds elements to the end of the array. Returns its new size.

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

Portugal	Espanha	França	Alemanha
0	1	2	3

- **Unshift**: adds elements to the end of the array. The other elements are shifted forward. Returns its new size.

`countries.unshift("Itália") // 5`

Itália	Portugal	Espanha	França	Alemanha
0	1	2	3	4

- **Splice**: adds/removes elements without leaving empty spaces
  - syntax: `arrayName.splice(index, 0, elementToAdd1, elementToAdd2, …)`

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

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

Note: operations at the end of the array are better, because they are faster

Array manipulation: removing elements

Portugal	Espanha	França
0	1	2

Example: remove elements

```
delete countries[1]
```

Portugal	*undefined*	França
0	1	2

Useful operations

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

- **Pop**: removes the array's last element. Returns the element.

```
countries.pop() // Alemanha
```

Itália	Polónia	Marrocos	Espanha	França
0	1	2	3	4

- **Shift**: removes the array0s first element. Other elements shift back. Returns the element.

```
countries.shift() // Itália
```

Polónia	Marrocos	Espanha	França
0	1	2	3

- **Splice**: adds/removes elements without leaving empty spaces
  - syntax: `arrayName.splice(index, nrOfElementsToRemove)`

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

Polónia	Espanha	França
0	1	2

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

França
0

Note: operations at the end of the array are better, because they are faster

Other methods to add/remove array elements

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

Method	Description	Example

`slice(start, end)`	Creates a new array with the elements between the **start** and **end** indexes	`let menosPaises = paises.slice(1,3)`
		`console.log(menosPaises)`
		`// "Espanha", "França"`

`concat(…items…)`	Joins items to the array	`menosPaises.concat("Marrocos")` `// "Espanha", "França", "Marrocos"`

Iterating

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

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

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

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

```
paises.forEach((item, index, array) => {
  alert(`${item} is at index ${index} in ${array}`)
});
```

Ordering

```
let countries = ["Portugal","Espanha","França"]
countries.sort()  // "Espanha", "França", "Portugal‟ (alphabetical order)
countries.reverse() // "Portugal", "França", "Espanha"


let numbers = [12,7,10,18,5]
numbers.sort() // 10, 12, 18, 5, 7 (alphabetical order)


function ascOrder(a, b) {
  return a - b
}
numbers = [12,7,10,18,5]
numbers.sort(ascOrder) // 5, 7, 10, 12, 18 (ascending order)

numbers.sort(function(a,b){return a - b}) // 5, 7, 10, 12, 18 (same, but smaller)
```

Gives a different order

Search methods

```
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 isAdult(idade){
  return idade>=18
}
```

Method	Description	Example

`indexOf(element,pos)`	Searches for the **element** on the array starting on index **pos** and returns the index or -1	`paises.indexOf("Espanha") // 1`
		`paises.indexOf("Inglaterra") // -1`
		`numeros.indexOf(2) // 1`
		`numeros.indexOf(2,3) // 7`

`lastIndexOf(element)`	Searches for the **element** on the array and returns the index of the last occurrence or -1	`numeros.lastIndexOf(2) // 9` `numeros.lastIndexOf(7) // -1`
`includes(element)`	Determines if **element** belongs to the array	`numeros.includes(3) // true` `numeros.includes(7) // false`
`find(function)`	Searches elements through **function** and returns the 1st found	`idades.find(isAdult) // 20`
`filter(function)`	Searches elements through **function** and returns all found	`idades.filter(isAdult) // [20, 40, 31]`

Array transformation methods

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

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

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

Method	Description	Example
`map(function)`	Applies **function** to all the elements in the array and returns the result	`numeros.map(mult2)  // [2,4,6,8,10]`
`fill(value, start, end)`	Fills the arrays positions from **start** to **end** with **value**	`numeros.fill(9,1,3)  // [1,9,9,4,5]` `numeros.fill(9)  // [9,9,9,9,9]`
`join(separator)`	Joins the array's elements separated by **separator**, into a string	`numeros.join("*")  // 1*2*3*4*5`
`reduce(func, initValue)`	Applies **func**[(1)] from an **initValue** to reduce the array to a single value	`numeros.reduce(soma)  // 15`

[(1)]Function func must be like:

```
function(acumulator, current, index, array){…}
```

Objects: creation and properties

Object creation

```
let user1 = {}

let user2 = new Object()
```

Empty objects

```
let user3 = {
  name: "Maria",
  age: 39
}
```

Properties

pairs `key: value`

```
let name = "Ana"
let age = 41
let user4 = {name, age}
```

Property creation

```
let newProp = "gender"
let user5 = {
  [newProp]: "male"
}
```

Property deletion

```
delete user3.age
console.log(user3)
          // {name: "Maria"}
```

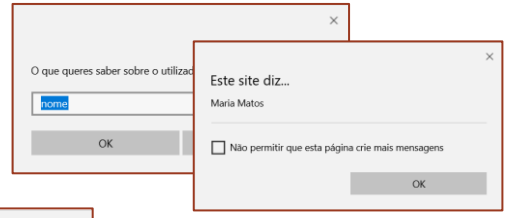
Objects: access and modify properties

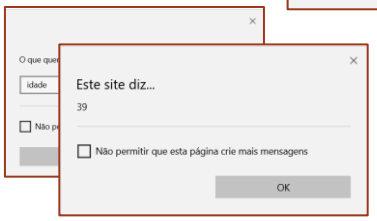
Access / modify object's properties

```
console.log(user3.name) // "Maria"

user3.name = "Maria Matos"
console.log(user3) // {age: 39, name: "Maria Matos"}

let key = prompt("What do you want to know about the user?", "name")
alert(user3[key])
```



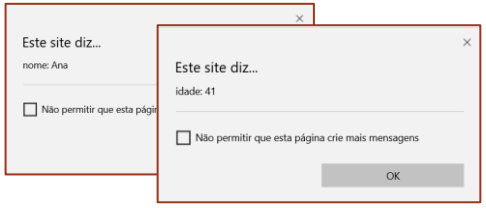


Determine if a property exists

```
console.log("age" in user4)      // true
console.log("eyeColor" in user4) // false
```

Iterate through all object's keys

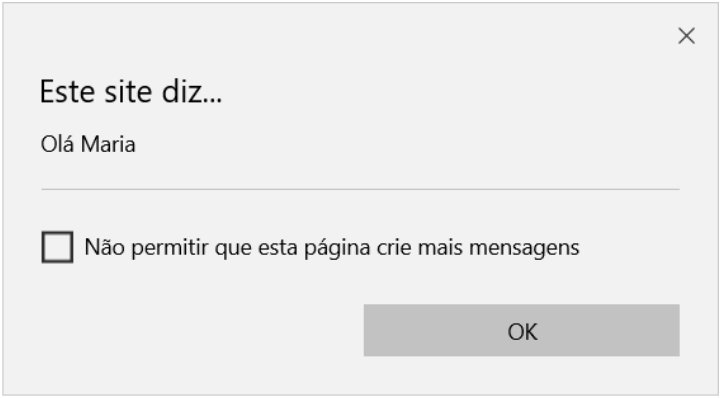
```
for(let key in user4){
  alert(key + ": " + user4[key])
}
```



Objects: methods

```
let user = {
  name: "Maria",
  age: 39,
  sayHello() {
    alert(`Hello ${this.name}`)
  }
}

user.sayHello()
```



Object copy

```
let person = {name = "Ana"}
```

- The copy is made by reference

```
let newPerson = person
newPerson.name = "Maria"
console.log(newPerson.name) // "Maria"
console.log(person.name) // "Maria" (the value is modified in both objects)
```

- How to not copy by reference

```
let newPerson ={}
// copy all properties
for(let key in person) {
  newPerson[key]  = person[key]
}
newPerson.name = "Maria"
console.log(newPerson.name) // "Maria"
console.log(person.name) // "Ana" (unchanged)
```

- Another way

```
let newPerson = Object.assign({},person)
newPerson.name = "Maria"
console.log(newPerson.name) // "Maria"
console.log(person.name) // "Ana" (unchanged)
```

Classes

- Used when several objects are needed

- Syntax

```
class MyClass {
  constructor(…){…}
  method1(…){…}
  method2(…){…}
  get property(…){…}
  set property(…){…}
  static staticMethod(…){…}
  …
}
```

1st character is CAPS

- Create a class object

```
let myObject = new MyClass(…)
```

Classes

- Example

```
class Person{
  constructor(name, age) {
    this.name = name
    this.age = age
  }
  get name(){
    return this._name
  }
  set name(value){
    this._name = valur
  }
  sayHello() {
    alert(`Olá ${this._name}`)
  }
  static compare(personA,personB) {
    return personA.age - personB.age
  }
}
```

_ : internal variable

```
let person1 = new Person("Maria",39)
person1.sayHello()
```

UNIVERSIDADE
PORTUCALENSE

Do conhecimento à prática.