

# Redução do Acoplamento

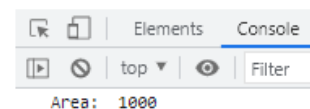
Keyword *this*

## *this*

- A keyword **this** pertence a um determinado *scope*, isto é:
  - Se usado num **método de um objeto** a keyword **this** refere-se a esse objeto
    - Se o objeto é criado com base numa class a utilização do **this** refere-se a cada instância em particular

```
const shape={  
  width:50,  
  height:20,  
  calculateArea:function(){  
    return this.width*this.height  
  }  
}  
console.log('Area: ' + shape.calculateArea())
```

width / height do objeto shape



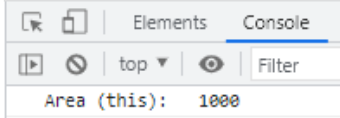
- Se aplicado numa função destinada a definir o método de um objeto, **this** refere-se ao objeto que contém o método

```
var width=100;
const shape={
  width:50,
  height:20
}

var calculateArea=function(){
  return this.width*this.height
}

shape.getArea=calculateArea
console.log('Area (this): ' + shape.getArea())
```

**this** refere-se ao objeto **shape**, uma vez que a função **calculateArea** é usada para definir o método **getArea** do objeto **shape**



- Se aplicado isolado ou numa função genérica, **this** refere-se ao global object (window object)
- Se aplicado num evento, **this** refere-se ao elemento que recebe o evento

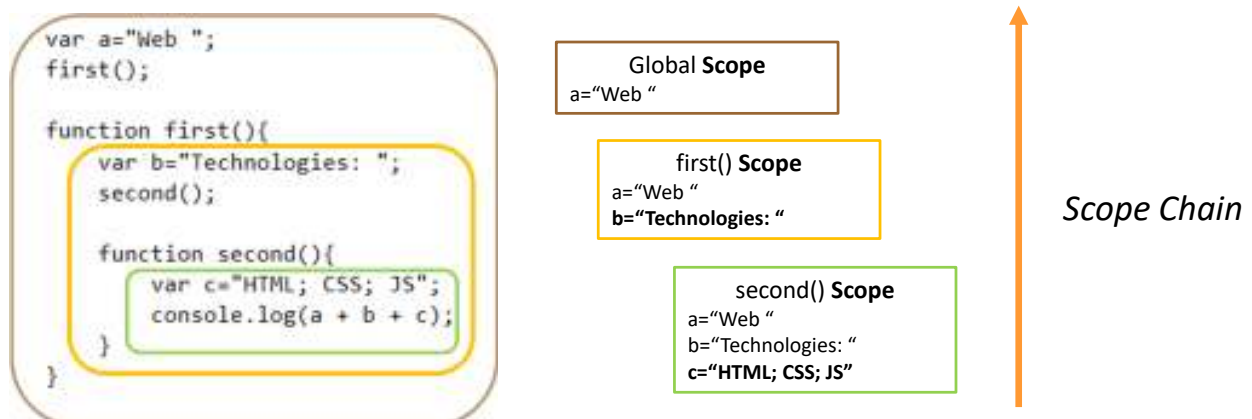
## Scope das Variáveis

# Scope

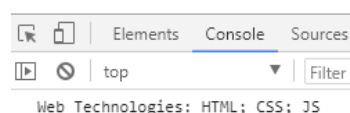
- Variável Global
  - Criada fora de qualquer função
  - É visível em todo o script (contexto global)
  - Armazenamento permanente (a variável existe enquanto a página estiver ativa no browser)
    - requer mais memória que as variáveis locais (eliminadas após a execução a função onde foram declaradas)
    - possíveis conflitos na nomenclatura das variáveis
- Variável Local
  - Criada no interior de uma função
  - Só é visível na função onde foi criada
  - Uma vez terminada a execução da função a variável é eliminada

# Scope

- Cada nova função cria um *scope*
  - Um espaço/ambiente onde as respetivas variáveis são acessíveis



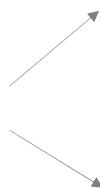
- Uma função que é **definida no interior** de uma outra função, tem **acesso ao scope da função que a envolve** (*scope chain*)



# Criação de Objetos JS

## Objetos

JavaScript



Objeto

Evento

### JavaScript Objects



In JavaScript, objects are king. If you understand objects, you understand JavaScript.

In JavaScript, almost "everything" is an object.

- Booleans can be objects (or primitive data treated as objects)
- Numbers can be objects (or primitive data treated as objects)
- Strings can be objects (or primitive data treated as objects)
- Dates are always objects
- Maths are always objects
- Regular expressions are always objects
- Arrays are always objects
- Functions are always objects
- Objects are objects

In JavaScript, all values, except primitive values, are objects.

Primitive values are: strings ("John Doe"), numbers (3.14), true, false, null, and undefined.

[http://www.w3schools.com/js/js\\_object\\_definition.asp](http://www.w3schools.com/js/js_object_definition.asp)

# Objetos

- Um objeto representa uma entidade física ou conceptual.
  - Tem estado (**propriedades**), comportamento (**métodos**) e identidade (**nome**).

- **Propriedades**

- Valores associados ao objeto

- **Métodos**

- Ações associadas ao objeto
  - Mesma sintaxe que propriedades
    - O valor é uma função

```
<script>
  var hotel = {
    name: 'Coimbra',
    rooms: 20,
    booked: 15,
    gym: true,
    roomTypes: ['single', 'double', 'suite'],

    checkAvailability: function () {
      return this.rooms - this.booked;
    }
  }
</script>
```

# Objetos

3 formas distintas  
de criar objetos

Forma Literal

new + Object()

class

# Objetos

## Forma Literal

```
const hotel={  
  
  name:'Coimbra',  
  rooms:20,  
  booked:15,  
  gym:true,  
  roomTypes:['single','double','suite'],  
  
  checkAvailability:function(){  
    return this.rooms - this.booked  
  }  
}
```

### Nome do objeto

**Propriedades** em que os valores podem ser:

string  
number  
boolean  
array

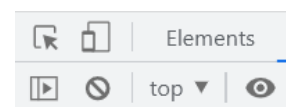
### Métodos

O scope do **this** é o objeto hotel

# Objetos

## new + Object()

```
const hotel= new Object();  
  
hotel.name='Coimbra';  
hotel.rooms=20;  
hotel.booked=15;  
hotel.gym=true;  
hotel.roomTypes=['single','double','suite'];  
  
hotel.checkAvailability=function(){  
  return this.rooms - this.booked  
}  
  
console.log(hotel.checkAvailability());
```



- cria um objeto ao qual **se adicionam propriedades e métodos**
- sintaxe totalmente diferente da forma literal.

## class

- Definição de uma *class*: “A class is a blueprint for objects”
  - Todas as classes possuem um **constructor** (método) onde se inicializam as propriedades

```
class Writer{  
  constructor(firstName,lastName){  
    this.fname=firstName,  
    this.lname=lastName  
  }  
  showName(){  
    console.log('Writer name: ' + this.fname + this.lname)  
  }  
}
```

nome da Classe (Por convenção o nome da class inicia-se com maiúscula)

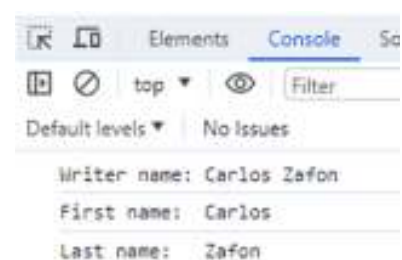
*constructor* (propriedades)

método

## class

- Criar um objeto passa pela criação de uma **instância da class**

```
→ const writerInstance=new Writer('Carlos',' Zafon')  
  
writerInstance.showName()  
  
console.log('First name: ' + writerInstance.fname)  
console.log('Last name: ' + writerInstance.lname)
```



# Objetos

- As classes permitem herança
  - uma classe pode fazer o **extend** de outra classe (herdar as suas propriedades e métodos)

```
class Person{
  constructor(a,b){
    this.age=a,
    this.country=b }
  showFeatures(){
    console.log('Age: ' + this.age + ' Country: ' + this.country)
  }
}
```

*Person* (Classe Pai)



```
class Writer extends Person{
  constructor(firstName,lastName,c,d){
    super(c,d)
    this.fname=firstName,
    this.lname=lastName }
  showName(){
    console.log('Writer name: ' + this.fname + this.lname)
  }
}
```

*Writer* (Classe Filho)

**super()** na classe filha é obrigatório, executado para invocar o constructor da classe pai

# Objetos

- Herança

```
const writerInstance=new Writer('Carlos',' Zafon', 50, "Spain")
```

```
writerInstance.showName()
```

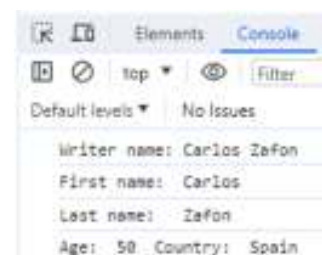
```
console.log('First name: ' + writerInstance.fname)
```

```
console.log('Last name: ' + writerInstance.lname)
```

```
writerInstance.showFeatures()
```

*Person*

*Writer*





## Exercício 5<sub>(Objeto)</sub>

### Objetos

- Com base no exemplo anterior, crie uma nova instância da class Writer.
  - Deve mostrar na consola o nome completo do escritor (José Luís Peixoto) assim como a idade (45 anos) e o respetivo país (Portugal).
    - Considere os métodos `showName()` e `show Features()`

# Criação de Objetos em JS

## Resumo

## Objetos

Forma Literal

- Sintaxe específica

- Pouco eficiente quando se pretendem criar múltiplas instâncias, obriga a definição repetida dos métodos

new + Object()

- Permite a criação de um objeto vazio ao qual se adicionam propriedades e métodos

class

- Definir uma classe e criar múltiplas instâncias dessa classe
- Solução mais eficaz quando se pretende a definição de múltiplos objetos (instâncias)
- Só disponível no ES6

# Propriedades / Métodos

## Resumo

## (.)dot notation

### ■ Propriedades

- ***objectName.propertyName***

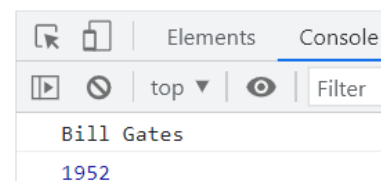
```
console.log(bill.name);
```

```
const bill={  
  name: 'Bill Gates',  
  age:72,  
  height:185,  
  calculateYearBirth: function(a){  
    return a-this.age;  
  }  
}
```

### ■ Métodos

- ***objectName.methodName( )***

```
console.log(bill.calculateYearBirth(2024));
```



# Reference Types

- Adicionar Propriedades
  - As propriedades podem ser adicionadas quando o objeto é criado ou então posteriormente em qualquer momento
    - Por defeito, em JavaScript os objetos podem ser sempre modificados

```
const books={  
  title:'Javascript',  
  editor: 'Packt Books',  
}
```

```
books.pages=456  
console.log(books)
```



# Objetos

- Remover propriedades
  - Da mesma forma que é possível criar também é possível remover propriedades em qualquer altura através do operador **delete**

```
const books={  
  title:'Javascript',  
  editor: 'Packt Books',  
}
```

```
books.pages=456
```

```
console.log(books)
```

```
delete(books.editor)
```

```
console.log(books)
```

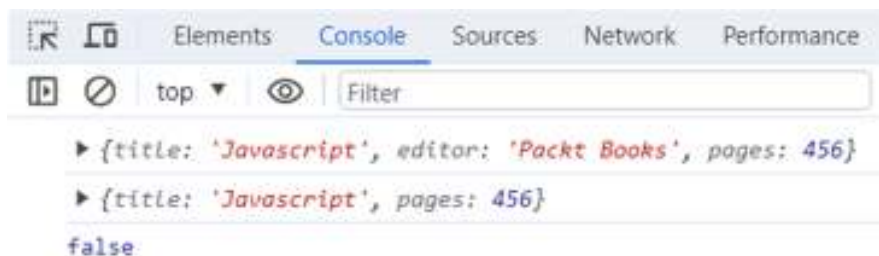
A screenshot of a web browser's developer console showing two log entries. The first entry is {title: 'Javascript', editor: 'Packt Books', pages: 456}. The second entry, which is highlighted, is {title: 'Javascript', pages: 456}, showing that the 'editor' property has been removed from the object.

- A propriedade foi removida como tal não se encontra definida
- Existem métodos que impedem a possibilidade de alterações nas propriedades de um objeto:
  - *Object.preventExtensions()* ; *Object.seal()*; *Object.freeze()*

# Objetos

- Detetar propriedades
  - Existem várias formas de detetar propriedades mas a forma mais fiável passa por utilizar o operador **in**

```
const books={  
  title:'Javascript',  
  editor: 'Packt Books',  
}  
  
books.pages=456  
  
console.log(books)  
  
delete(books.editor)  
  
console.log('editor' in books)
```



## JavaScript Built-in Objects

## JavaScript Reference

The references describe the properties and methods of all JavaScript objects, along with examples.

Array	Boolean	Date	Error	Global
JSON	Math	Number	Operators	RegExp
Statements	String			

<https://www.w3schools.com/jsref/default.asp>

# JavaScript Built-in Objects

- **String** (\*Primitive Wrapper Types)
- **Number** (\*Primitive Wrapper Types)
- **Math**
- **Date**
- **Array**
- **Boolean** (\*Primitive Wrapper Types)
- **RegExp**
- ...

*Standard Built-in  
Objects*

# JavaScript Built-in Objects: String

## JavaScript Built-in Objects

### ■ String

Propriedade	Descrição
length	retorna o número de caracteres que constituem a string

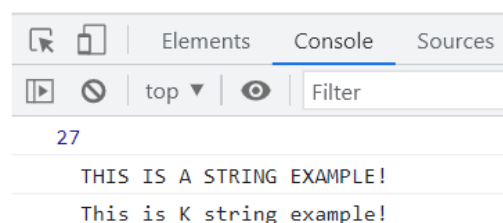
Método	Descrição
toUpperCase()	Conversão para maiúsculas
toLowerCase()	Conversão para minúsculas
trim()	Remove espaços em branco do início e do fim da string
split()	Permite dividir uma string e guardar cada componente numa string
replace()	Considera um valor que deve ser substituído por outro
substring()	Retorna os caracteres entre dois índices
charAt()	Retorna um carácter numa determinada posição
indexOf()	Retorna a posição da primeira ocorrência de um dado valor na string, retorna -1 se o valor nunca é detetado. É case sensitive.

## Exercício 5 (propriedades/Métodos String)

### JavaScript Built-in Objects

- declare a string "This is a string example!"
- Mostre o número de caracteres
- Converta a string para maiúsculas
- Substitua o primeiro 'a' por 'K'

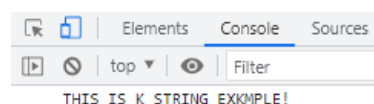
```
var s=" This is a string example!"  
  
console.log(s.length)  
console.log(s.toUpperCase())  
console.log(s.replace('a','K'))
```



Todas as ocorrências:

```
var s=" This is a string example!"  
s=s.toUpperCase().replaceAll('A','K')  
console.log(s)
```

methods chaining





# JavaScript Built-in Objects: Number

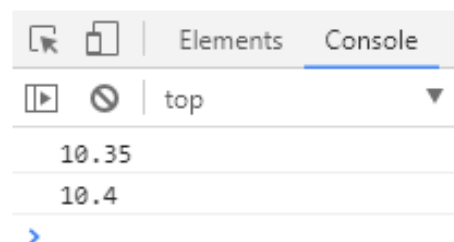
## JavaScript Built-in Objects

### ■ Number

Método	Descrição
<code>toExponential()</code>	conversão para notação exponencial
<code>toPrecision()</code>	arredonda o número para um dado nº de dígitos
<code>toFixed()</code>	arredonda o número para um dado nº de casas decimais
<code>toString()</code>	converte para uma <i>string</i>
...	...

```
<script>
  var x=10.354;

  console.log(x.toFixed(2));
  console.log(x.toPrecision(3));
</script>
```



# JavaScript Built-in Objects: Math

## JavaScript Built-in Objects

- **Math**
  - Operações matemáticas, disponibiliza funções matemáticas avançadas (trigonometria, estatística, etc.)
  - As propriedades/métodos do Math são chamadas **sem criar de forma explícita um objecto** do tipo Math (exemplo: Math.round(x)) **Exceção!**

Propriedade	Descrição
Math.PI	retorna aproximadamente 3.14159265359

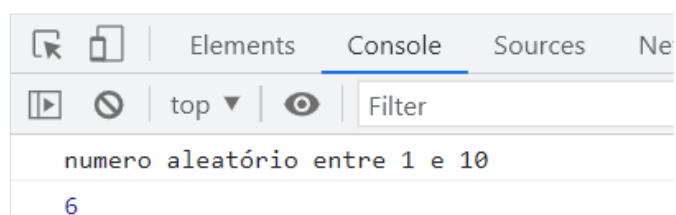
Método	Descrição
Math.round()	arredonda o número para o inteiro mais próximo
Math.sqrt()	Raiz quadrada de um número positivo
Math.ceil()	arredonda o número para o inteiro imediatamente seguinte
Math.floor()	arredonda o número para o inteiro imediatamente anterior
Math.random()	Gera um número aleatório entre 0 e 1

## Exercício 6 (propriedades/Métodos Math)

### Exercício 6

- Gera um número aleatório entre 1 e 10
  - `Math.random()` (gera um número aleatório no intervalo  $[0,1[$ )
  - `*` (multiplicação)
  - `Math.floor()`
  - `+` (adição)

```
var num = Math.floor(Math.random()*10)+1  
console.log('numero aleatório entre 1 e 10')  
console.log(num)
```



# JavaScript Built-in Objects: Date

## JavaScript Built-in Objects

- Date
  - Disponibiliza métodos e atributos para aceder e manipular horas e datas
  - **É necessário instanciar** um objeto deste tipo para aceder aos seus métodos

```
var data = new Date(); //sem parâmetros o objeto é criado com a data atual
```

### JavaScript Date Methods and Properties

Name	Description
<code>constructor</code>	Returns the function that created the Date object's prototype
<code>getDate()</code>	Returns the day of the month (from 1-31)
<code>getDay()</code>	Returns the day of the week (from 0-6)
<code>getFullYear()</code>	Returns the year
<code>getHours()</code>	Returns the hour (from 0-23)
<code>getMilliseconds()</code>	Returns the milliseconds (from 0-999)
<code>getMinutes()</code>	Returns the minutes (from 0-59)
<code>getMonth()</code>	Returns the month (from 0-11)
<code>getSeconds()</code>	Returns the seconds (from 0-59)

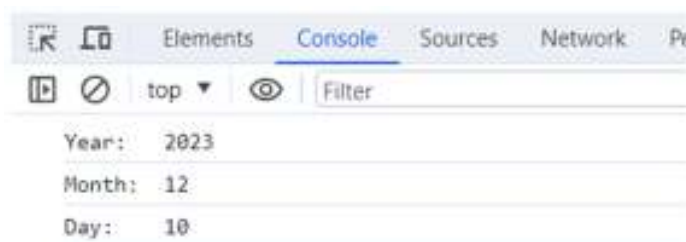
[http://www.w3schools.com/js/js\\_obj\\_date.asp](http://www.w3schools.com/js/js_obj_date.asp)

## Exercício 7 (propriedades/Métodos Date)

## Exercício 7

- Instancie um novo objeto do tipo Date (obtendo a data atual)
- Mostre na consola:
  - ano: `getFullYear()`
  - mês: `getMonth()`
  - dia: `getDate()`

```
const tdDate = new Date()  
console.log('Year: ' + tdDate.getFullYear())  
console.log('Month: ' + (tdDate.getMonth()+1))  
console.log('Day: ' + tdDate.getDate())
```



# JavaScript Built-in Objects: Array

## JavaScript Built-in Objects

- **Array**
  - permite armazenar diferentes tipos de elementos

Propriedade	Descrição
length	número de elementos de um <i>array</i>

Método	Descrição
indexOf()	pesquisa um elemento e retorna a sua posição
pop()	remove o último elemento de um <i>array</i>
push()	adiciona um elemento ao <i>array</i>
shift()	remove o primeiro elemento de um <i>array</i>
sort()	ordena os elementos de um <i>array</i>
unshift()	adiciona elementos no início de um <i>array</i>
...	...

[https://www.w3schools.com/jsref/jsref\\_obj\\_array.asp](https://www.w3schools.com/jsref/jsref_obj_array.asp)

# JavaScript Built-in Objects

## ■ Arrays

- Os *arrays* são particularmente importantes em JS precisamente pela capacidade de armazenar valores relacionados.

- declarados de forma literal ou com base no *constructor Array*
- índices iniciam em zero

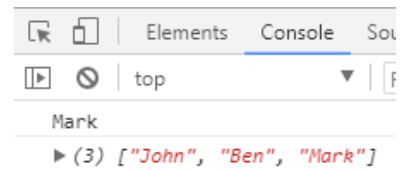
```
<script>

  var names = ['John', 'Jane', 'Mark'];
  var years = new Array(1990, 1969, 1948);

  console.log(names[2]);

  names[1] = 'Ben';
  console.log(names);

</script>
```



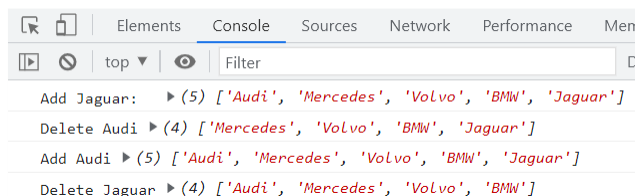
## Exercício 8 (propriedades/Métodos Array)

## JavaScript Built-in Objects

- Declare um array com os valores

“Audi”, “Mercedes”, “Volvo”, “BMW”

- Adicione ‘Jaguar’ no final: push()
- Apague o primeiro elemento: shift()
- Adicione o primeiro elemento: unshift()
- Apague o ultimo elemento: pop()



```
const cars=['Audi', 'Mercedes', 'Volvo', 'BMW']
```

```
cars.push('Jaguar')  
console.log('Add Jaguar: ', cars)
```

```
cars.shift()  
console.log('Delete Audi', cars)
```

```
cars.unshift('Audi')  
console.log('Add Audi',cars)
```

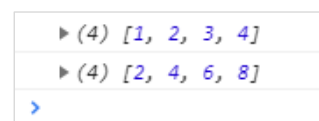
```
cars.pop()  
console.log('Delete Jaguar',cars)
```

## JavaScript Built-in Objects

### ■ map()

- Executa uma função para cada um dos elementos do array

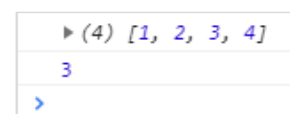
```
const numbers =[1,2,3,4];  
  
const newNumbers = numbers.map((num)=>{return num*2;})  
  
console.log(numbers);  
console.log(newNumbers);
```



### ■ find()

- Retorna o valor do primeiro elemento que verifica a condição

```
const numbers =[1,2,3,4];  
  
const newNumbers = numbers.find((num)=>{return num>2;})  
  
console.log(numbers);  
console.log(newNumbers);
```





## JavaScript Built-in Objects

### ■ `findIndex()`

- Retorna o índice do primeiro elemento que verifica a condição. Caso não exista nenhum elemento que verifique a condição retorna -1.

```
const numbers = [1,2,3,4];

const newNumbers = numbers.findIndex((num)=>{return num === 2;})

console.log(numbers);
console.log(newNumbers);
```

```
▶ (4) [1, 2, 3, 4]
1
>
```

### ■ `filter()`

- Cria um novo array com todos os elementos que verificam a condição.

```
const numbers = [1,2,3,4];

const newNumbers = numbers.filter((num)=>{return num > 2;})

console.log(numbers);
console.log(newNumbers);
```

```
▶ (4) [1, 2, 3, 4]
▶ (2) [3, 4]
>
```

## JavaScript Built-in Objects

### ■ `reduce()`

- Executa uma função definida pelo utilizador em cada elemento do array e cujo resultado é um valor único (ex: a soma de todos os elementos).

```
const numbers = [1,2,3,4];

const newNumbers = numbers.reduce((acc, acv)=>{return acc + acv;})

console.log(numbers);
console.log(newNumbers);
```

*acc – previous value  
acv – current value*

```
▶ (4) [1, 2, 3, 4]
10
>
```

### ■ `concat()`

- faz a concatenação de dois arrays. Os arrays originais não são alterados.

```
const numbers = [1,2,3,4];

const addNumbers = [5,6];

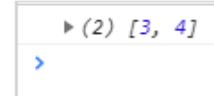
console.log(numbers.concat(addNumbers));
```

```
▶ (6) [1, 2, 3, 4, 5, 6]
>
```

### ■ `slice()`

- retorna uma cópia parcial desde uma posição inicial até ao final (a posição final não é especificada). O array original não é alterado.

```
const numbers = [1, 2, 3, 4];  
  
console.log(numbers.slice(2));
```

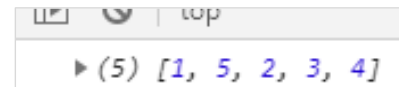


► (2) [3, 4]  
>

### ■ `splice()`

- altera o conteúdo de um array adicionando ou substituindo elementos (exemplo: remove 0 elementos a partir da posição 1, e adiciona o número 5)

```
const numbers = [1, 2, 3, 4];  
numbers.splice(1, 0, 5);  
  
console.log(numbers);
```

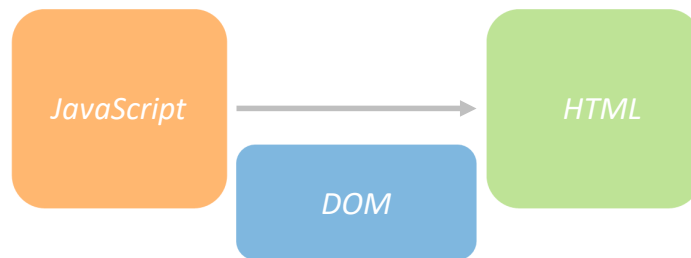


► (5) [1, 5, 2, 3, 4]

## Document Object Model - DOM

## Document Object Model (DOM)

- Um dos principais objetivos da utilização do *JavaScript* é a capacidade de manipular/alterar/controlar elementos HTML
  - DOM disponibiliza:
    - Métodos / Propriedades de forma a aceder a todo o documento HTML permitindo dessa forma a geração/alteração dinâmica de conteúdo HTML



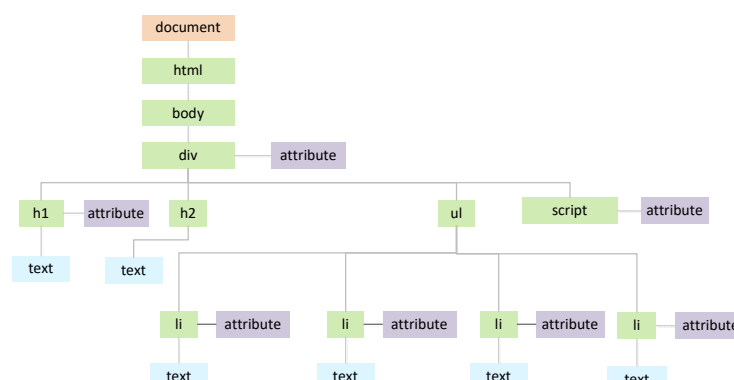
*Document Object Model*

*The **HTML DOM** is a standard for how to get, change, add, or delete HTML elements.*

[http://www.w3schools.com/js/js\\_htmlDOM.asp](http://www.w3schools.com/js/js_htmlDOM.asp)

## Document Object Model (DOM)

- Quando é feito o download de um documento HTML, este torna-se num **document object**
  - É o *root node* do document HTML e o "owner" de todos os outros nós:
    - element nodes; text nodes; attribute nodes; comment nodes
- O *document object* disponibiliza propriedades e métodos para aceder a todos os nós com base em JavaScript.



*DOM tree*

## Document Object Model (DOM)

### HTML DOM considera tudo como nó (DOM Tree)

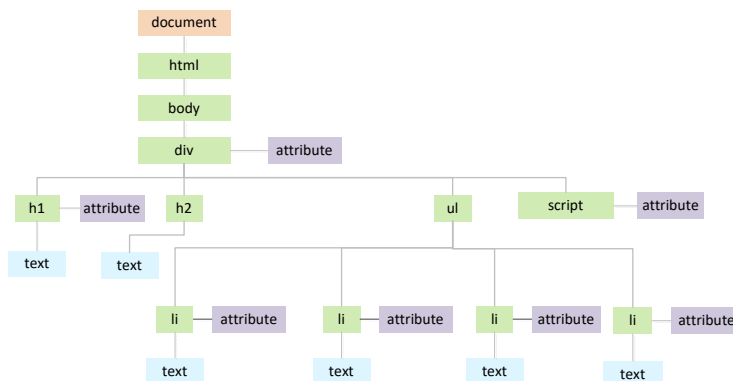
- O documento (**root element**) é um **document node**
- Qualquer elemento HTML é um **HTML node**
- O texto contido nos elementos HTML é um **text node**
- Qualquer atributo HTML é um **attribute node**

document

Element HTML

Text

Attribute



DOM tree

## Document Object Model (DOM)

### HTML DOM Reference

The references describe the properties and methods of each object, along with examples.

Attributes	Console	Document	Element	Events
Event Objects	Geolocation	History	HTMLCollection	Location
Navigator	Screen	Style	Window	Storage

# Document Object

DOM

## HTML DOM Reference

The references describe the properties and methods of each object, along with examples.

Attributes	Console	<u>Document</u>	Element	Events
Event Objects	Geolocation	History	HTMLCollection	Location
Navigator	Screen	Style	Window	Storage

## DOM: Document Object

### DOM Methods

`document.getElementById()`

Retorna o *element object* que tem o *id* com o valor especificado

`document.querySelector()`

Retorna o primeiro *elemento object* referenciado pelo seletor CSS

`document.querySelectorAll()`

Retorna o conjunto de elementos referenciados pelo seletor CSS

`document.getElementsByTagName()`

Retorna o conjunto de elementos (HTML Collection) cuja designação da *tag* corresponde ao nome especificado

`document.getElementsByClassName()`

Retorna um conjunto de elementos (HTML Collection) que tem o atributo *class* com o nome especificado

[http://www.w3schools.com/jsref/dom\\_obj\\_document.asp](http://www.w3schools.com/jsref/dom_obj_document.asp)

## DOM: Document Object

### DOM Methods

- Sempre que necessário utilizar o mesmo elemento mais do que uma vez, o *element object* deve ser referenciado por uma variável:

```
var identificaElemento = document.getElementById('one');
```

```
<p id="p1">First paragraph</p>
```

```
<div class="d1">First div</div>
```

```
<script>
```

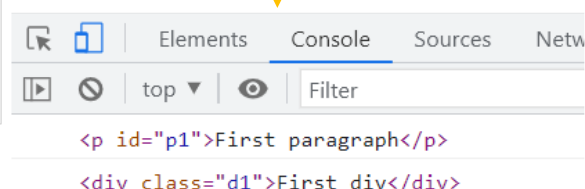
```
var a=document.getElementById('p1');
```

```
var b=document.querySelector('.d1')
```

```
console.log(a)
```

```
console.log(b)
```

Métodos DOM que referenciam elementos HTML



# Element Object

DOM

## HTML DOM Reference

The references describe the properties and methods of each object, along with examples.

Attributes	Console	Document	<u>Element</u>	Events
Event Objects	Geolocation	History	HTMLCollection	Location
Navigator	Screen	Style	Window	Storage

## DOM: Element Object

- O Element Object representa um elemento HTML

### HTML Element Objects Reference

The references describe the properties and methods of each HTML object, along with examples.

a	abbr	address	area	article	aside	audio	b	base	bdo	blockquote	body	br	button	canvas	caption	cite	code	col	colgroup
datalist	dd	del	details	dfn	dialog	div	dl	dt	em	embed	fieldset	figcaption	figure	footer	form	head	header	h1 - h6	hr
html	i	iframe	img	ins	input-button	input-checkbox	input-color	input-date	input-datetime	input-datetime-local	input-email	input-file	input-hidden	input-image	input-month	input-number	input-password	input-radio	input-range
input-reset	input-search	input-submit	input-text	input-time	input-url	input-week	kbd	label	legend	li	link	map	mark	menu	menuitem	meta	meter	nav	object
ol	optgroup	option	output	p	param	pre	progress	q	s	samp	script	section	select	small	source	span	strong	style	sub
summary	sup	table	tbody	td	tfoot	th	thead	tr	textarea	time	title	track	u	ul	var	video			

- possui propriedades/métodos
  - No entanto existem algumas propriedades/métodos adicionais que são específicas de alguns elementos
    - Exemplo:
      - propriedade **disabled** é específica do Button object

<https://www.w3schools.com/jsref/>

## DOM: Document Object

- **Element Object**
  - Representa um elemento HTML (ex: <div>, <form>, <a>, ...)
  - Tem métodos e propriedades associadas
    - Algumas Propriedades **muito importantes**:

*innerHTML*

Acesso/alteração do conteúdo HTML de um elemento

*textContent*

Acesso/alteração de texto

*className*

Retorna/altera o valor do atributo class de um elemento \*: substitui todos os valores anteriormente definidos para o atributo class

*classList*

Retorna todos os valores do atributo class de um elemento

*id*

Acesso/alteração do atributo id de um elemento



# DOM: Document Object

## ■ Element Object

### ■ Métodos importantes:

`addEventListener()`

Adiciona um event handler ao elemento

`getAttribute()`

Acesso/alteração de texto

```
<p id="p1">First paragraph</p>
```

```
<script>
```

```
var a=document.getElementById('p1');
```

```
a.textContent='CHANGED !'
```

Propriedade **textContent** permite alterar de forma dinâmica o conteúdo do element object referenciado pela variável **a**

ZoomInfo ISEC - myISEC

CHANGED !

# DOM

Javascript  
altera/controla  
elementos HTML ?

JavaScript

DOM

HTML

Document Object

Element Object

```
document.getElementById()  
document.querySelector()  
document.querySelectorAll()  
document.getElementsByTagName()  
document.getElementsByClassName()  
...
```

+

```
innerHTML  
textContent  
className  
classList  
id  
...
```

Métodos que referenciam 1 ou vários  
elementos HTML

Propriedades/métodos que permitem alterar  
conteúdos/características/estrutura HTML

# HTML Collection Object

DOM