

## *Adicionar / Remover / Detetar Propriedades*

### 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

```
book={  
  title:"Javascrit",  
  year:2018,  
  editor:"O'Reilly",  
  
  showDetails:function(){  
    return ("Book title: " + this.title + " ;    Book editor: " + this.editor);  
  }  
}
```

```
...  
}  
  
book.pages=250;  
  
alert("Number of pages: " + book.pages);
```

Number of pages: 250

OK

## Objetos

### ■ Remover propriedades

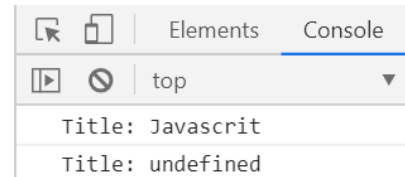
- Da mesma forma que é possível criar também é possível remover propriedades em qualquer altura através do operador ***delete***

```
book={
  title:"Javascrit",
  year:2018,
  editor:"O'Reilly"
}

console.log('Title: ' + book.title);

delete book.title;

console.log('Title: ' + book.title);
```



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

## Objetos

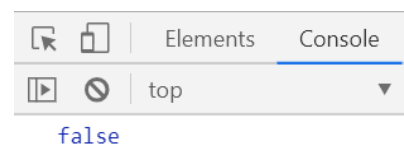
### ■ Detetar propriedades

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

```
book={
  title:"Javascrit",
  year:2018,
  editor:"O'Reilly"
}

delete book.title;

console.log('title' in book);
```



# JavaScript Built-in Objects

## 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

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.

## JavaScript Built-in Objects

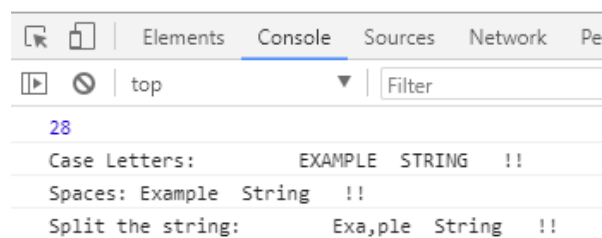
### ■ String

```
<script>

var exStr="      Example String  !!"

console.log(exStr.length);

console.log("Case Letters: " + exStr.toUpperCase());
console.log("Spaces: " + exStr.trim());
console.log("Split the string:" + exStr.split('m'));
</script>
```



## JavaScript Built-in Objects

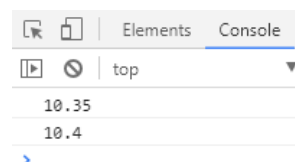
### ■ Number

Método	Descrição
toExponential()	conversão para notação exponencial
toPrecision()	arredonda o número para um dado nº de dígitos
toFixed()	arredonda o número para um dado nº de casas decimais
toString()	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

- 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))

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

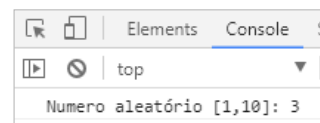
## JavaScript Built-in Objects

### ■ Math

- Gera um número aleatório entre 1 e 10
  - é gerado um número aleatório entre 0 e 1
    - *Math.random()*
  - multiplicado por 10 e arredondado para o número inteiro imediatamente anterior
    - *Math.floor()*
  - número entre 0 e 9 ao qual se adiciona 1

```
<script>
  var x = Math.floor(Math.random()*10)+1;

  console.log("Numero aleatório [1,10]: " + x);
</script>
```



# 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

Date Get Methods		Date Set Methods	
Get methods are used for getting a part of a date. Here are the most common (alphabetically):		Set methods are used for setting a part of a date. Here are the most common (alphabetically):	
Method	Description	Method	Description
getDate()	Get the day as a number (1-31)	setDate()	Set the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)	setFullYear()	Set the year (optionally month and day yyyy.mm.dd)
getFullYear()	Get the four digit year (yyyy)	setHours()	Set the hour (0-23)
getHours()	Get the hour (0-23)	setMilliseconds()	Set the milliseconds (0-999)
getMilliseconds()	Get the milliseconds (0-999)	setMinutes()	Set the minutes (0-59)
getMinutes()	Get the minutes (0-59)	setMonth()	Set the month (0-11)
getMonth()	Get the month (0-11)	setSeconds()	Set the seconds (0-59)
getSeconds()	Get the seconds (0-59)	setTime()	Set the time (milliseconds since January 1, 1970)
getTime()	Get the time (milliseconds since January 1, 1970)		

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

# JavaScript Built-in Objects

## ■ Date

```
<script>

var tdyDate = new Date();

console.log("Year: " + tdyDate.getFullYear());
console.log("Month: " + (tdyDate.getMonth()+1));
console.log("Day: " + tdyDate.getDate());

</script>
```

Elements	
top	
Year: 2018	
Month: 3	
Day: 19	

### Date Get Methods

Get methods are used for getting a part of a date. Here are the most common (alphabetically):

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

## JavaScript Built-in Objects

### ■ Array

- permite armazenar diferentes tipos de elementos

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

Método	Descrição
<i>indexOf()</i>	pesquisa um elemento e retorna a sua posição
<i>pop()</i>	remove o último elemento de um <i>array</i>
<i>push()</i>	adiciona um elemento ao <i>array</i>
<i>shift()</i>	remove o primeiro elemento de um <i>array</i>
<i>sort()</i>	ordena os elementos de um <i>array</i>
<i>unshift()</i>	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

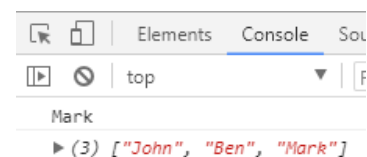
```
<script>

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

console.log(names[2]);

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

</script>
```





# JavaScript Built-in Objects

## ■ Arrays

### ■ métodos (exemplo):

```
<script>

var john = ['John', 'Smith', 1990, 'designer', false];

john.push('blue');
john.unshift('Mr. ');

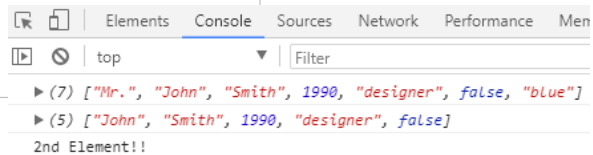
console.log(john);

john.pop();
john.shift();

console.log(john);

if (john.indexOf('Smith') === 1) {
    console.log('2nd Element!!');
}

</script>
```



# JavaScript Built-in Objects

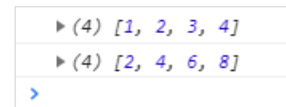
## ■ `map()`

- 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);
```



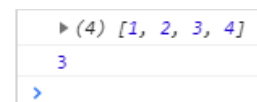
## ■ `find()`

- 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);
```

```
▸ (4) [1, 2, 3, 4]  
10  
>
```

### ▪ `concat()`

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

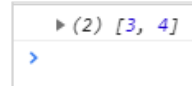
```
const numbers =[1,2,3,4];  
  
const addNumbers = [5,6];  
  
console.log(numbers.concat(addNumbers));
```

```
▸ (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));
```

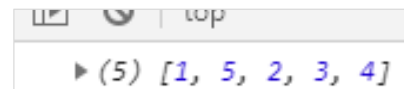


► (2) [3, 4]  
>

### ■ `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);
```

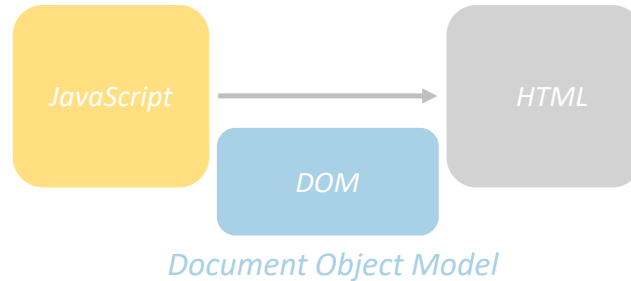


► (5) [1, 5, 2, 3, 4]

## 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



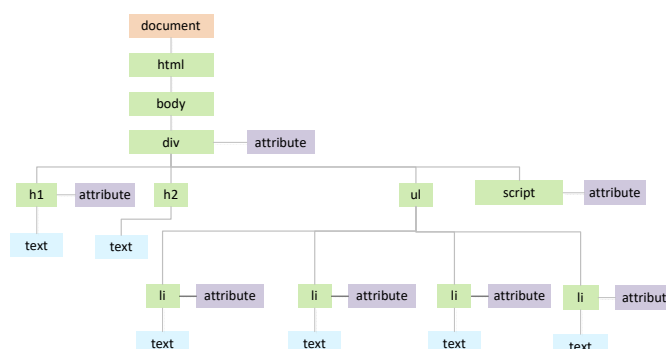
The **HTML DOM** is a standard for how to get, change, add, or delete HTML elements.

[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**

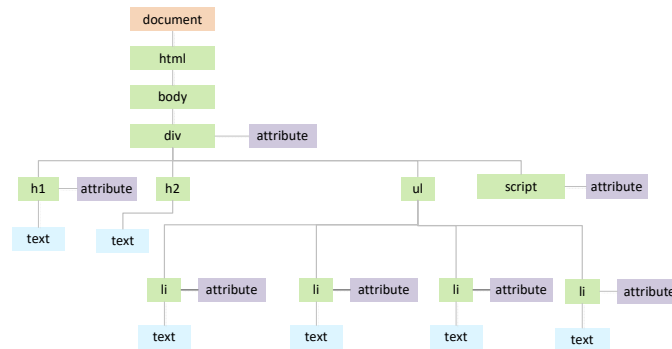
- É o *root node* do document HTML e o "owner" de todos os outros nós:
  - element nodes; text nodes; attribute nodes; comment nodes
- O *document object* disponibiliza propriedades e métodos para aceder a todos os nós com base em JavaScript.



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**



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

## 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 Methods

Método ( <i>DOM Queries</i> )	Descrição
<code>document.getElementById()</code>	retorna o <i>element object</i> que tem o id com o valor especificado
<code>document.querySelector()</code>	retorna o primeiro <i>elemento object</i> referenciado pelo seletor CSS
<code>document.querySelectorAll()</code>	retorna o conjunto de elementos referenciados pelo seletor CSS
<code>document.getElementsByTagName()</code>	retorna conjunto de elementos ( <i>HTML Collection</i> ) cuja designação da tag corresponde ao nome especificado
<code>document.getElementsByClassName()</code>	retorna um conjunto de elementos ( <i>HTML Collection</i> ) que tem o atributo class com o nome especificado
...	... <a href="http://www.w3schools.com/jsref/dom_obj_document.asp">http://www.w3schools.com/jsref/dom_obj_document.asp</a>

- 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');
```

## Element Object

# 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

## Document Object Model (DOM)

- Element Object
  - Representa um elemento HTML (ex: <div>, <form>, <a>, ...)
  - Um conjunto importante de métodos e propriedades associadas

Propriedades	Descrição
<i>innerHTML</i>	acesso/alteração do conteúdo HTML de um elemento
<i>textContent</i>	Acesso/alteração de texto
<i>className</i>	Retorna/altera o valor do atributo <i>class</i> de um elemento
<i>classList</i>	Retorna todos os valores do atributo <i>class</i> de um elemento
<i>id</i>	acesso/alteração do atributo id de um elemento
...	...

[https://www.w3schools.com/jsref/dom\\_obj\\_all.asp](https://www.w3schools.com/jsref/dom_obj_all.asp)



## HTMLCollection Object

## 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	<u>HTMLCollection</u>	Location
Navigator	Screen	Style	Window	Storage

### ■ HTMLCollection Object

- Representa uma lista de elementos HTML
- Permite indexação como se de um array se tratasse (**não é um array!**)
  - Não permite a utilização de métodos do objeto array.
  - Possui a propriedade length (muito útil!)

#### ■ Método *item()*

```
var elements=document.getElementsByClassName('vegetables');  
  
if(elements.length>=1){  
    var firstItem=elements.item(0);}
```

#### ■ *Array syntax*

```
var elements=document.getElementsByClassName('vegetables');  
  
if(elements.length>=1){  
    var firstItem=elements[0];}
```

## Document Object

### *Seleção de Elementos* (DOM nodes)

## Seleção de Elementos

- **document.getElementById()**
  - a forma mais rápida e mais eficaz de aceder a um único elemento
  - retorna o nó cujo elemento tem o id especificado na *DOM Query*

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var veg = document.getElementById('two');
  veg.className='promo';
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Garlic  
Cabage

## Seleção de Elementos

- **document.querySelector()**
  - Baseado em seletores CSS
  - Retorna o **primeiro elemento** que verifica o seletor CSS definido como argumento

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var element=document.querySelector("#three");
  element.className='promo';
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Garlic  
Cabage

## Seleção de Elementos

### ■ Métodos que retornam uma *HTML Collection*

Método ( <i>DOM Queries</i> )	Descrição
<code>document.getElementById()</code>	retorna o <i>element object</i> que tem o id com o valor especificado
<code>document.querySelector()</code>	retorna o primeiro <i>elemento object</i> referenciado pelo seletor CSS
<code>document.querySelectorAll()</code>	retorna o conjunto de elementos ( <i>HTML Collection</i> ) referenciados pelo seletor CSS
<code>document.getElementsByTagName()</code>	retorna conjunto de elementos ( <i>HTML Collection</i> ) cuja designação da tag corresponde ao nome especificado
<code>document.getElementsByClassName()</code>	retorna um conjunto de elementos ( <i>HTML Collection</i> ) 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)

## Seleção de Elementos

- `document.querySelectorAll()`
  - permite aceder aos *DOM nodes* com base num seletor CSS
  - Requer indexação mesmo que exista apenas um element ao qual se aplique o selector CSS

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
  <li id="four" class="fish">sardines</li>
  <li id="five" class="fish">codfish</li>
</ul>

<script>
  var elements,el;
  var elements = document.querySelectorAll('.fish');
  for (var i=0; i<elements.length; i++)
  {
    elements[i].className='promo';
  }
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Garlic  
Cabage  
sardines  
codfish

## Seleção de Elementos

- `document.getElementsByTagName()`
  - retorna uma *HTML Collection* com todos os elementos que verificam o nome da *tag*

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
    <li id="two" class="vegetables">Garlic</li>
    <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var elements,el;
  var elements = document.getElementsByTagName('li');
  if (elements.length >2)
  {
    el=elements[2];
    el.className='promo';
  }
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Garlic  
Cabage

- **Importante:** Os elementos tem que ser indexados, mesmo que apenas exista uma ocorrência da *tag name* (posição [0])

## Seleção de Elementos

- `document.getElementsByClassName()`
  - À semelhança do `getElementsByTagName()` retorna uma node list com os elementos com o valor do atributo *class* definido como argumento
  - Requer indexação

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
    <li id="two" class="vegetables">Garlic</li>
    <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var elements,el;
  var elements = document.getElementsByClassName('vegetables');
  if (elements.length >2)
  {
    el=elements[2];
    el.className='promo';
  }
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Garlic  
Cabage

## Seleção de Elementos

- Utilização de métodos de seleção de elementos diretamente numa condição
  - Caso o elemento exista é retornado o valor *true*
    - A presença de um *element object* é um **truthy value**:

```
if (document.getElementById('idValue'))
{
    /* processamento se o elemento existe */
}
else
{
    /* processamento se o elemento não foi encontrado*/
}
```

## Document Object

### *Criação de Elementos (DOM nodes)*

## Criação de Elementos

### ■ Métodos

Método	Descrição
<code>createElement()</code>	cria um elemento HTML <u>mas não adiciona</u> à DOM tree
<code>createTextNode()</code>	cria um nó de texto <u>mas não adiciona</u> à DOM tree
<code>createComment()</code>	cria um <i>comment node</i> com o texto especificado
...	...

## Criação de Elementos

### ■ `createElement(); / createTextNode();`

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>
```

```
<script>
  var elRef, elNew, elNewText;
```

```
elNew=document.createElement("li");
elNewText=document.createTextNode("Codfish");
```

```
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

Criação dos elementos (elemento + texto)

## Element Object

### *Alteração de Estrutura*

## *Alteração de Estrutura*

### ■ Métodos

Método	Descrição
<code>appendChild()</code>	adiciona um nó à DOM tree
<code>insertBefore()</code>	insere um elemento antes de uma dada referência
<code>replaceChild()</code>	substitui um elemento por outro
<code>removeChild()</code>	remove um elemento



## Alteração de Estrutura

### ■ `appendChild()`

- Permite a integração/inclusão de um novo nó (elemento HTML / texto) no documento HTML que se pretende alterar
- Aceita como **único argumento** o nó que se **pretende adicionar**
- **O método deve ser chamado** no elemento que se pretende que seja o **pai** (estrutura HTML) do nó a inserir.

## Alteração de Estrutura

### ■ `appendChild()`

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>
```

```
<script>
  var elRef, elNew, elNewText;
```

```
elRef=document.getElementsByTagName("ul")[0]
elNew=document.createElement("li");
elNewText=document.createTextNode("Codfish");
```

```
elNew.appendChild(elNewText);
elRef.appendChild(elNew);
```

```
elNew.className="promo";
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

Ref. para posicionamento (elemento pai)

Adição dos elementos à DOM tree  
(elemento pai)

### Grocery List

Onions  
Garlic  
Cabage  
Codfish

## Alteração de Estrutura

### ▪ `insertBefore()`

- Permite a inserção de um elemento antes de um dado elemento
- Aceita dois argumentos: o primeiro é o nó a ser inserido e o segundo o nó (id) que determina o posicionamento

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var elRef, elNew, elNewText;

  elRef=document.getElementsByTagName("ul")[0];
  elNew=document.createElement("li");
  elNewText=document.createTextNode("Codfish");

  elNew.appendChild(elNewText);
  elNew.className="promo";

  elRef.insertBefore(elNew,document.getElementById("two"));
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Codfish  
Garlic  
Cabage

## Alteração de Estrutura

### ▪ `replaceChild()`

- Permite a substituição de um elemento por outro
- Aceita dois argumentos: o primeiro é o nó a ser inserido e o segundo o nó (id) a ser substituído

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var elRef, elNew, elNewText;

  elRef=document.getElementsByTagName("ul")[0];
  elNew=document.createElement("li");
  elNewText=document.createTextNode("Codfish");

  elNew.appendChild(elNewText);
  elNew.className="promo";

  elRef.replaceChild(elNew,document.getElementById("two"));
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Codfish  
Cabage

## Alteração de Estrutura

- **removeChild ()**
  - apaga um nó que aceita como argumento
  - à semelhança do **appendChild()** o método **removeChild()** tem que ser **invocado no nó pai**

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var elRef, elRmv;

  elRef=document.getElementsByTagName("ul")[0];
  elRmv=document.getElementById("two");

  elRef.removeChild(elRmv);
</script>
```

### Grocery List

Onions  
Cabage

## Alteração de Estrutura

- **insertAdjacentHTML()**
  - permite inserir um conteúdo HTML numa determinada posição. O conteúdo deve ser passado como uma string, a qual é convertida para markup HTML.
    - **element.insertAdjacentHTML(position, str)**
      - Position:
        - beforebegin: antes do próprio elemento.
        - afterbegin: logo após o início do elemento.
        - beforeend: no interior do elemento, depois do último filho.
        - afterend: depois do elemento.

```
<h1>Grocery List</h1>
<ul><li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var element;
  element='<li id="four" class="fish"><a href="https://www.fishforthought.co.uk">sardines</a></li><li id="five" class="fish">...
  document.getElementsByTagName("ul")[0].insertAdjacentHTML("afterbegin",element);
</script>
```

### Grocery List

[sardines](#)  
[codfish](#)  
Onions  
Garlic  
Cabage

# Element Object

## Alteração de Conteúdo (Propriedades)

## Alteração de Conteúdo

### ■ **innerHTML**

- a propriedade mais versátil uma vez que permite alterar/definir o conteúdo (texto/**markup**) de um dado elemento
- alterar o **markup** de um elemento (ex: adicionar um link)

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var element=document.getElementById("three");
  element.innerHTML='<a href="https://www.fishforthought.co.uk">sardines</a>';
  element.className='promo';
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

## Grocery List

Onions  
Garlic  
[sardines](https://www.fishforthought.co.uk)

# Alteração de Conteúdo

## ■ Alteração de conteúdo

### ■ *textContent*

- obtém/altera o conteúdo textual de um elemento e ignora todo o *markup* contido por esse elemento

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var element=document.getElementById("three");
  element.textContent='<a href="https://www.fishforthought.co.uk">sardines</a>';
  element.className='promo';
</script>
```

### Grocery List

Onions  
Garlic

[sardines](https://www.fishforthought.co.uk)

# Alteração de Conteúdo

### *document.write()*

#### Vantagens:

Rápido e simples de mostrar o resultado da adição de conteúdo

#### Desvantagens:

Funciona corretamente quando é feito o download da página

Se utilizado depois efetua o **overwrite** do conteúdo anterior



### *element.innerHTML*

#### Vantagens:

Permite a inserção de markup com menos código do que os métodos DOM

É mais rápido do que os métodos DOM

É a forma mais simples de remover todo o conteúdo de um elemento (string vazia).

#### Desvantagens:

Pode criar problemas com os *event handlers*, uma vez que origina a eliminação de todos os *child elements* os quais podem estar a ser utilizados para disparar um evento

### Métodos DOM

#### Vantagens:

Permite aceder de forma precisa a todos os elementos da *DOM Tree*.

Não afeta os *event handlers*

Permite a adição incremental de elementos.

#### Desvantagens:

Caso se pretenda efetuar muitas alterações ao conteúdo é mais lento que *innerHTML*

Para atingir um determinado objetivo, necessita de mais código quando comparado com o *innerHTML*

## Element Object

### *Alteração de Atributos*

### *Alteração de Atributos*

Método	Descrição
<code>getAttribute()</code>	obtem o valor do atributo de um elemento
<code>hasAttribute()</code>	verifica se o elemento possui um dado atributo
<code>setAttribute()</code>	define o valor de um atributo
<code>removeAttribute()</code>	remove um atributo de um elemento

Propriedades	Descrição
<code>className</code>	permite obter/estabelecer o valor do atributo class
<code>classList</code>	permite obter o valor do atributo class de um elemento (usada em conjunto com os métodos <code>add()</code> e <code>remove()</code> , <code>toggle()</code> para adicionar ou eliminar uma class)
<code>style</code>	permite definir/obter o valor de propriedades CSS
<code>id</code>	permite obter/estabelecer o valor do atributo id

## Alteração de Atributos

- `getAttribute(); / setAttribute();`

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>
<p></p>

<script>
  var element, msg="";

  element=document.getElementById("three").getAttribute("class");
  msg+= "Old class attribute: " + element + "<br>";

  element=document.getElementById("three").setAttribute("class", "promo");
  element=document.getElementById("three").getAttribute("class");
  msg+= "New class attribute: " + element;

  document.getElementsByTagName("p")[0].innerHTML=msg;
</script>
```

### Grocery List

Onions  
Garlic  
Cabage

Old class attribute: vegetables  
New class attribute: promo

- `setAttribute()` permite múltiplas alterações: alterar imagem (src); destino (href), ...

## Alteração de Atributos

- Propriedade **style**
  - mais eficaz e expedito que os métodos anteriores para alteração de atributos

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="vegetables">Cabage</li>
</ul>

<script>
  var element=document.querySelector("#three");
  element.style.backgroundColor='lightblue';
  element.style.color='white';
</script>
```

### Grocery List

Onions  
Garlic  
Cabage

- O nome das propriedades difere ligeiramente/pontualmente das propriedades CSS originais
  - [https://www.w3schools.com/jsref/dom\\_obj\\_style.asp](https://www.w3schools.com/jsref/dom_obj_style.asp)

## Alteração de Atributos

### ■ Propriedade **classList**

- Propriedade muito importante para controlar a formatação de um elemento através de classes CSS previamente definidas.
- Esta propriedade retorna um DOMTokenList object (objecto que contém uma lista de strings)

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="Veg1 Veg2 Veg3 Veg4">Cabage</li>
</ul>

<script>
  var element=document.querySelector("#three");
  var namesClass = element.classList;
  console.log(namesClass);
</script>
```

```
▼ DOMTokenList(4)
0: "Veg1"
1: "Veg2"
2: "Veg3"
3: "Veg4"
length: 4
```

- A este DOMTokenList object podem ser aplicados métodos muito importantes para controlar quais são as classes aplicadas a um elemento:
  - add();
  - remove();
  - contains();

## Alteração de Atributos

### ■ **classList**

- exemplo de aplicação de classes pré definidas
  - *contains(); add(); remove()*

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="Veg1 Veg2 Veg3 Veg4">Cabage</li>
</ul>

<script>
  var element=document.querySelector("#three");
  if (element.classList.contains('promo'))
    {element.classList.remove('promo');}
  else
    {element.classList.add('promo');}
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

### Grocery List

Onions  
Garlic  
Cabage