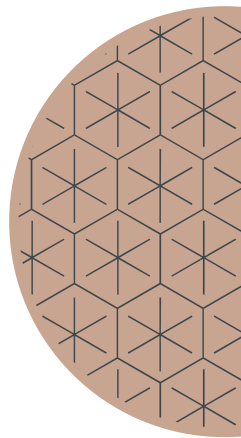


# **Modelo de objetos predefinidos en JavaScript**

# Modelo de objetos predefinidos en JavaScript

En esta unidad veremos los siguientes conceptos:

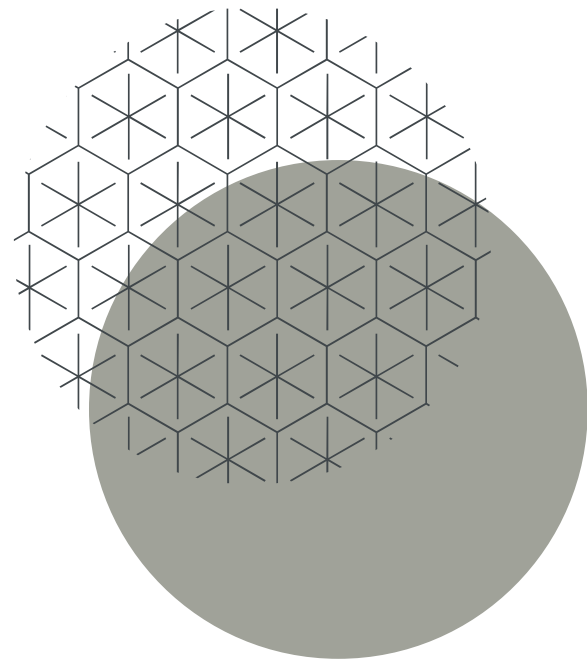
- Objetos nativos de JavaScript: Date, Math, Number, String, Boolean, RegExp.
- Generación de elementos HTML desde código JavaScript.
- Gestión de las ventanas.





# Date

---



# Objeto Date

Los objetos Date de JavaScript representan un único momento en el tiempo en un formato independiente de la plataforma. Los objetos Date contienen un número que representa milisegundos desde el 1 de enero de 1970 UTC.

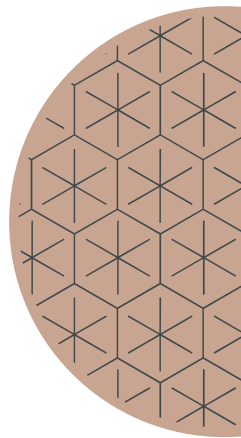
Es importante tener en cuenta que aunque el valor de la hora de un objeto Date es UTC, los métodos básicos para obtener la fecha y la hora o sus componentes funcionan todos en la zona horaria local (es decir, la del cliente).

# Formato y conversiones por zona horario

Existen varios métodos para obtener una fecha en varios formatos, así como para realizar conversiones de zonas horarias.

## **Date.now()**

Devuelve el valor numérico correspondiente a la hora actual: el número de milisegundos transcurridos desde el 1 de enero de 1970 a las 00:00:00 UTC.



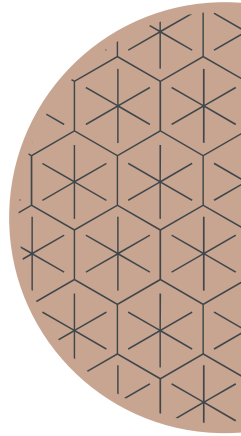
# Formato y conversiones por zona horario

## **Date.parse()**

Analiza una representación de cadena de una fecha y devuelve el número de milisegundos desde el 1 de enero de 1970, 00:00:00 UTC.

## **Date.UTC()**

Acepta los mismos parámetros que la forma más larga del constructor (es decir, de 2 a 7) y devuelve el número de milisegundos desde el 1 de enero de 1970, 00:00:00 UTC.



# Métodos de instancia

## **Date.prototype.getDate()**

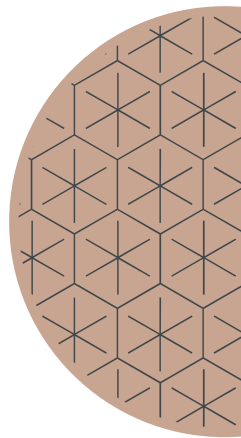
Devuelve el día del mes (1 - 31) según la hora local.

## **Date.prototype.getDay()**

Devuelve el día de la semana (0 - 6) según la hora local.

## **Date.prototype.getHours()**

Devuelve la hora (0 - 23) según la hora local.



# Ejemplos de crear objeto

*const today = new Date()*

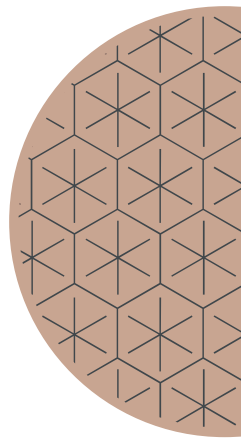
*const birthday = new Date('December 17, 1995 03:24:00') // DISCOURAGED: may not work in all runtimes*

*const birthday2 = new Date('1995-12-17T03:24:00') // This is ISO8601-compliant and will work reliably*

*const birthday3 = new Date(1995, 11, 17) // the month is 0-indexed*

*const birthday4 = new Date(1995, 11, 17, 3, 24, 0)*

*const birthday5 = new Date(628021800000) // passing epoch timestamp*





# Ejemplos de mostrar objeto

```
const date = new Date("2020-05-12T23:50:21.817Z");
```

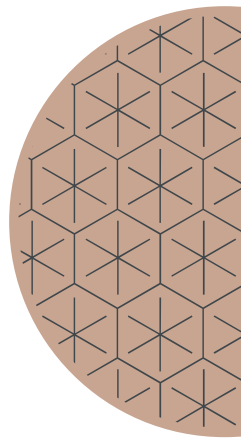
```
date.toString()           // Tue May 12 2020 18:50:21 GMT-0500 (Central Daylight Time)
```

```
date.toDateString()       // Tue May 12 2020
```

```
date.toTimeString()       // 18:50:21 GMT-0500 (Central Daylight Time)
```

```
date.toISOString()        // 2020-05-12T23:50:21.817Z
```

```
date.toUTCString()        // Tue, 12 May 2020 23:50:21 GMT
```



# Ejemplos de mostrar objeto

```
const date = new Date("2020-05-12T23:50:21.817Z");
```

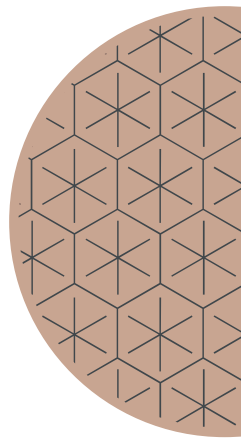
```
date.toGMTString()      // Tue, 12 May 2020 23:50:21 GMT
```

```
date.toJSON()           // 2020-05-12T23:50:21.817Z
```

```
date.toLocaleString()   // 5/12/2020, 6:50:21 PM
```

```
date.toLocaleDateString() // 5/12/2020
```

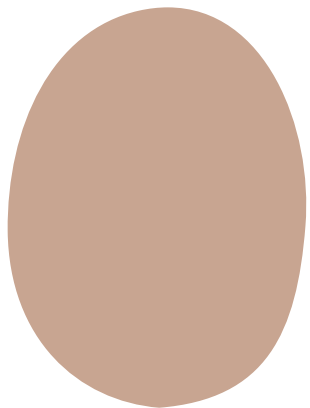
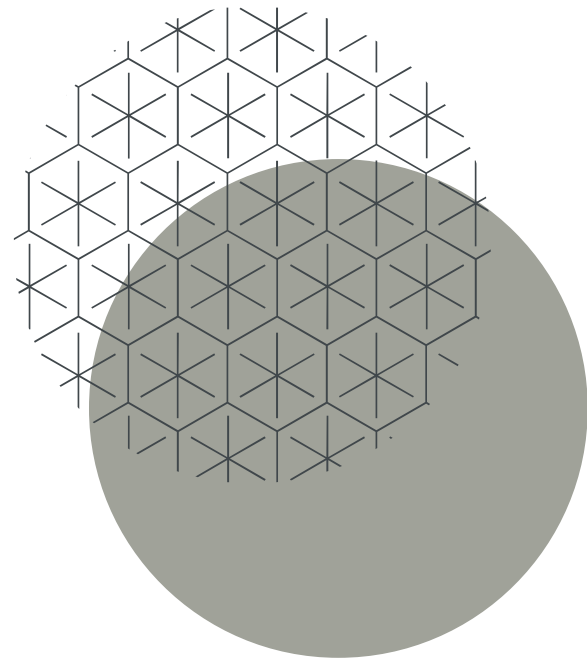
```
date.toLocaleTimeString() // 6:50:21 PM
```





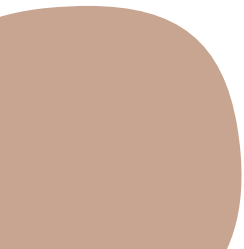
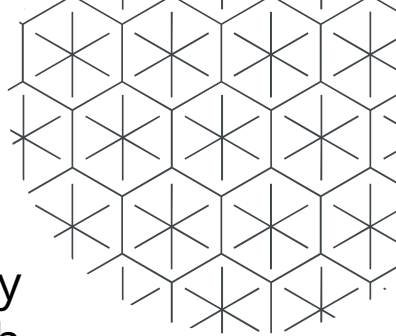
# Math

---



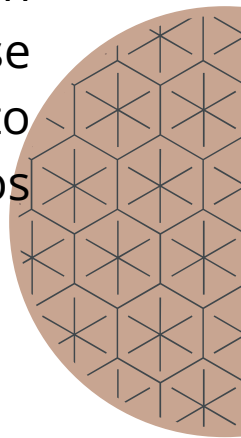
# Math

Math es un objeto incorporado que tiene propiedades y métodos para las constantes y funciones matemáticas. Math funciona con el tipo Number, no funciona con BigInt.



# Math

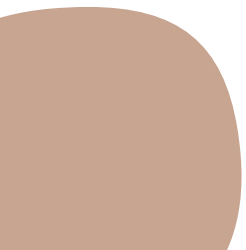
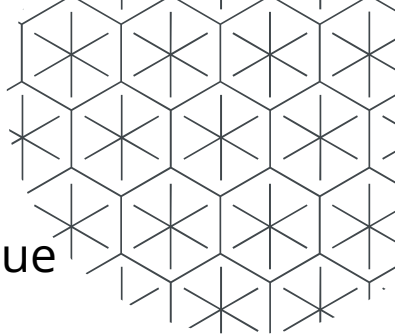
A diferencia de muchos otros objetos globales, Math no tiene un constructor. Todas las propiedades y métodos de Math son estáticos. Para utilizarla constante pi como se utiliza `Math.PI` y se llama a la función seno como `Math.sin(x)`, donde `x` es el argumento del método. Las constantes se definen con toda la precisión de los números reales en JavaScript.



# Math

Muchas funciones matemáticas tienen una precisión que depende de la implementación.

Esto significa que diferentes navegadores pueden dar un resultado diferente. Incluso el mismo motor de JavaScript en un sistema operativo o arquitectura diferente puede dar resultados diferentes.



# Math

## **Math.abs()**

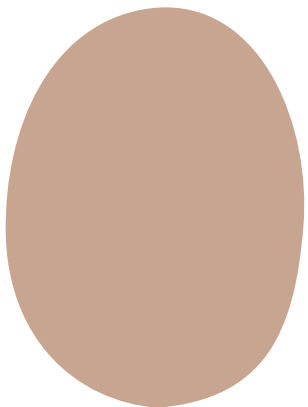
Devuelve el valor absoluto de x.

## **Math.ceil()**

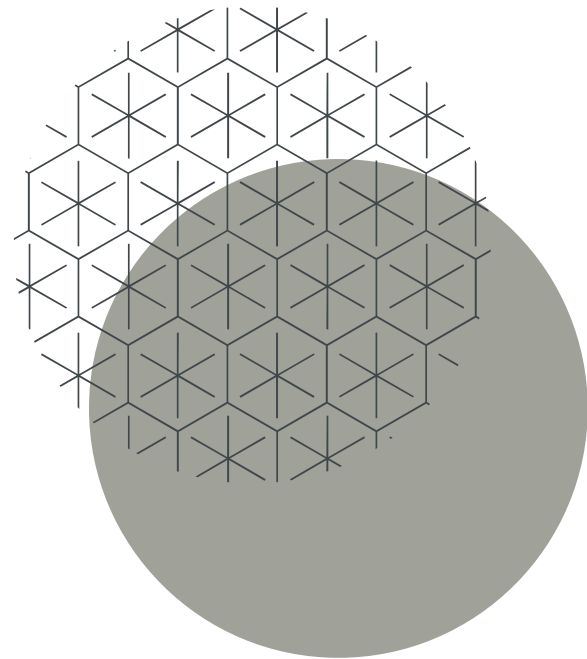
Devuelve el menor número entero mayor o igual a x.

## **Math.floor()**

Devuelve el mayor número entero menor o igual a x.



# Number

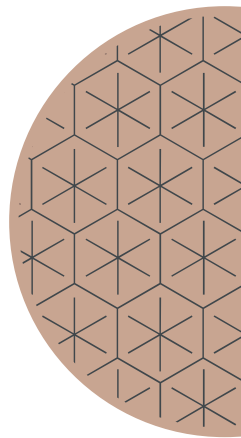




# Number

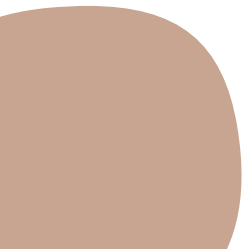
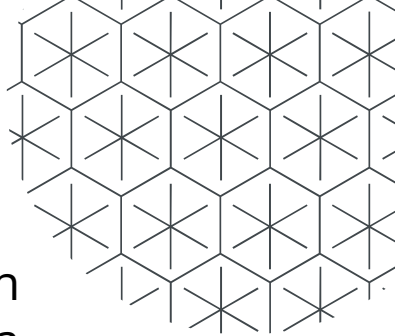
Number es un objeto wrapper primitivo utilizado para representar y manipular números como 37 o -9,25.

El constructor Number contiene constantes y métodos para trabajar con números. Los valores de otros tipos pueden convertirse en números mediante la función Number().



# Number

El tipo Number de JavaScript es un valor de doble precisión en formato binario IEEE 754 de 64 bits, como double en Java o C#. Esto significa que puede representar valores fraccionarios, pero hay algunos límites a la magnitud y precisión del número almacenado.



# Number métodos estáticos

## **Número.isNaN()**

Determina si el valor pasado es NaN.

## **Número.isFinite()**

Determina si el valor pasado es un número finito.

## **Número.isInteger()**

Determina si el valor pasado es un número entero.

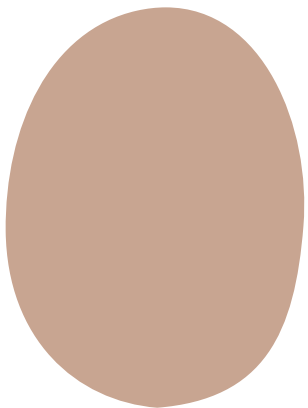
# Number métodos estáticos

## **Number.parseFloat()**

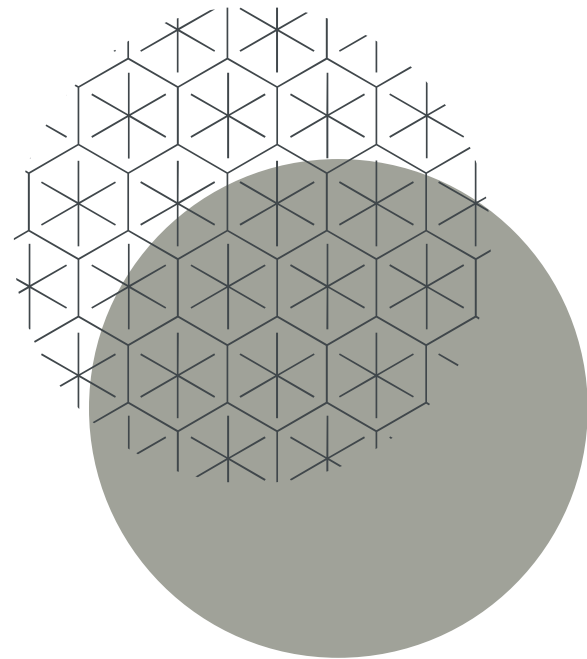
Es la misma que la función global parseFloat().

## **Número.parseInt()**

Es la misma que la función global parseInt().

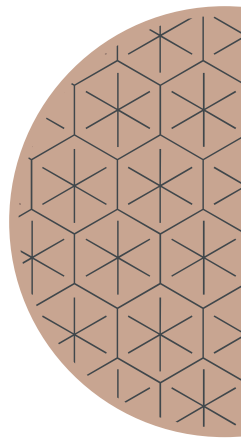


# String



# String

Los String son útiles para contener datos que pueden representarse en forma de texto. Algunas de las operaciones más utilizadas sobre las cadenas son comprobar su longitud, construirlas y concatenarlas mediante los operadores de cadena + y +=, comprobar la existencia o ubicación de subcadenas con el método indexOf() o extraer subcadenas con el método substring().



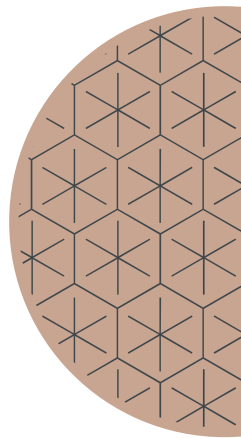
# String acceso caracteres

Hay dos maneras de acceder a un carácter individual en una cadena. La primera es el método `charAt()`:

*'cat'.charAt(1) // da el valor "a"*

La otra forma es tratar la cadena como un objeto tipo array, donde los caracteres individuales corresponden a un índice numérico:

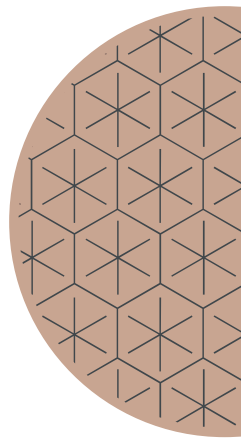
*'cat'[1] // da el valor "a"*



# Comparar string

Tenga en cuenta que todos los operadores de comparación, incluidos `===` y `==`, comparan las cadenas distinguiendo entre mayúsculas y minúsculas. Una forma común de comparar cadenas sin distinguir entre mayúsculas y minúsculas es convertir ambas al mismo caso (mayúsculas o minúsculas) antes de compararlas.

```
function areEqualCaseInsensitive(str1, str2) {  
  
    return str1.toUpperCase() === str2.toUpperCase();  
  
}
```

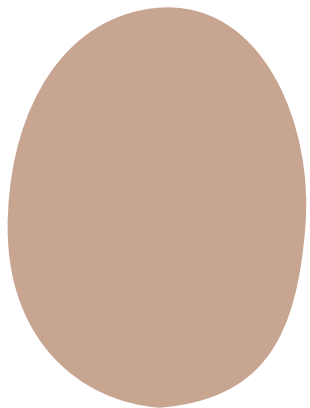
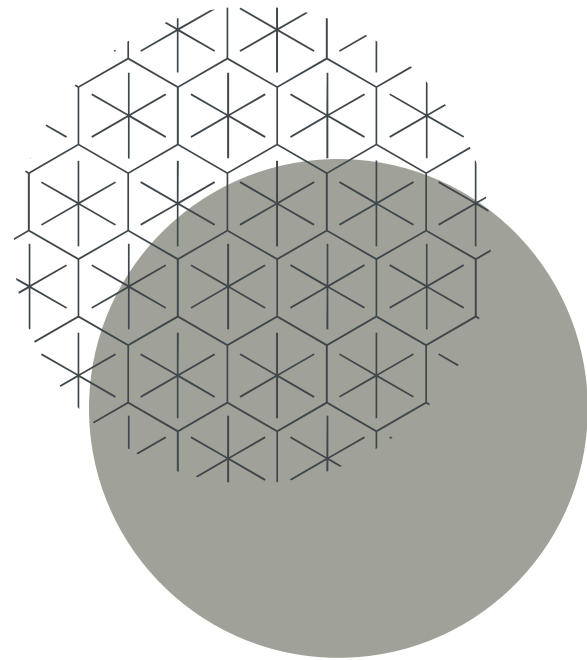






# Boolean

---



# Boolean

El objeto booleano representa un valor de verdad: verdadero o falso. Es importante no confundirlo con los valores primitivos `true` y `false`.

Cualquier objeto, incluyendo un objeto booleano cuyo valor es falso, se evalúa como verdadero cuando se pasa a una sentencia condicional.

# Asignación de variables

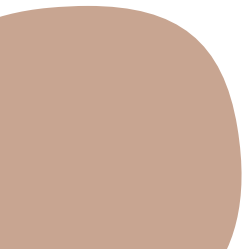
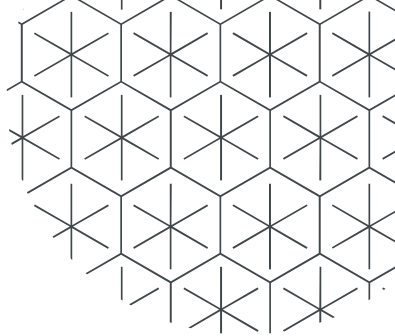
```
const x = new Boolean(false);
```

```
if (x) {
```

```
// este código se ejecuta.
```

```
}
```

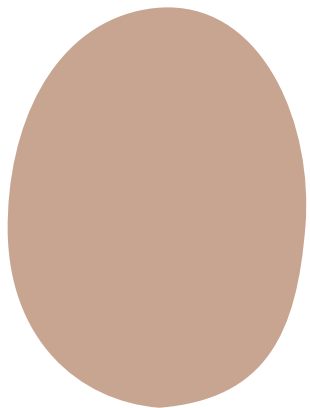
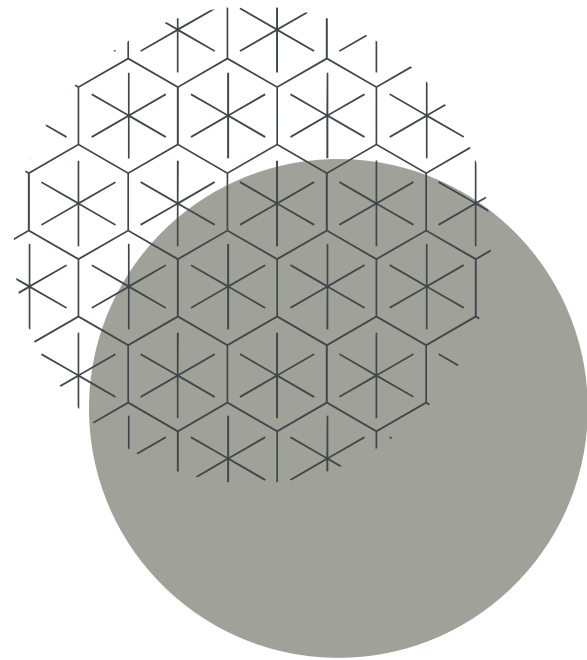
Este comportamiento no se aplica a las primitivas booleanas. Por ejemplo, si en vez de usar `Boolean` usamos el tipo primitivo `false` entonces no se ejecutaría.





# RegExp

---



# RegExp

Las expresiones regulares son patrones que se utilizan para hacer coincidir combinaciones de caracteres en las cadenas. En JavaScript, las expresiones regulares son también objetos. Estos patrones se utilizan con los métodos `exec()` y `test()` de `RegExp`, y con los métodos `match()`, `matchAll()`, `replace()`, `replaceAll()`, `search()` y `split()` de `String`.

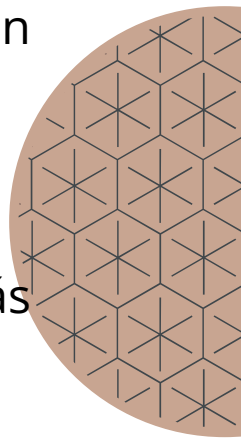
# RegExp: Constructor

Una expresión regular se construye de dos maneras:

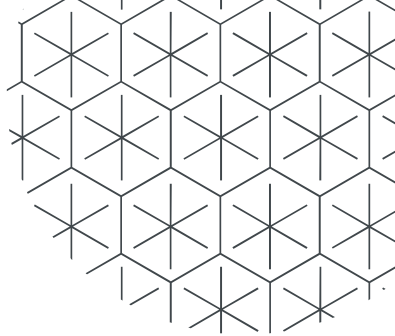
Utilizando un literal de expresión regular, que consiste en un patrón encerrado entre barras, como se indica a continuación:

```
const re = /ab+c/;
```

Si la expresión regular permanece constante, su uso es el más recomendable.



# RegExp: Constructor



O llamando a la función constructora del objeto RegExp, como sigue

```
const re = new RegExp('ab+c');
```

Utilice la función constructora cuando sepa que el patrón de la expresión regular va a cambiar, o cuando no conozca el patrón y lo obtenga de otra fuente, como la entrada del usuario.

# RegExp: patrones sencillos

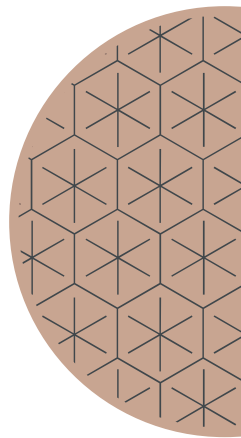
Los patrones simples se construyen con caracteres para los que se desea encontrar una coincidencia directa. Por ejemplo, el patrón `/abc/` coincide con combinaciones de caracteres en cadenas sólo cuando se produce la secuencia exacta "abc" (todos los caracteres juntos y en ese orden).



# RegExp: caracteres especiales

Cuando la búsqueda de una coincidencia requiere algo más que una coincidencia directa, como encontrar una o más b, o encontrar espacios en blanco, puede incluir caracteres especiales en el patrón.

Por ejemplo, para buscar una sola "a" seguida de cero o más "b" seguidas de "c", se utilizaría el patrón `/ab*c/`: el `*` después de la "b" significa "0 o más apariciones del elemento anterior". En la cadena "cbbabbbbbcdebc", este patrón coincidirá con la subcadena "abbbbc".



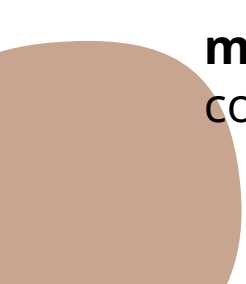
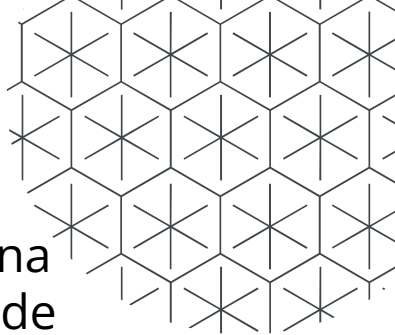
# RegExp en javascript

**exec():** Ejecuta una búsqueda de una coincidencia en una cadena. Devuelve un array de información o null en caso de no coincidencia.

**test():** Comprueba si hay una coincidencia en una cadena. Devuelve true o false.

**match():** Devuelve una matriz que contiene todas las coincidencias, incluidos los grupos de captura, o null si no se encuentra ninguna coincidencia.

**matchAll():** Devuelve un iterador que contiene todas las coincidencias, incluidos los grupos de captura.

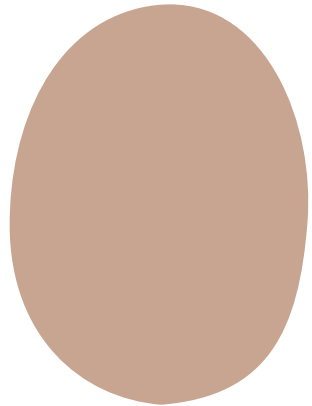


# RegExp en javascript

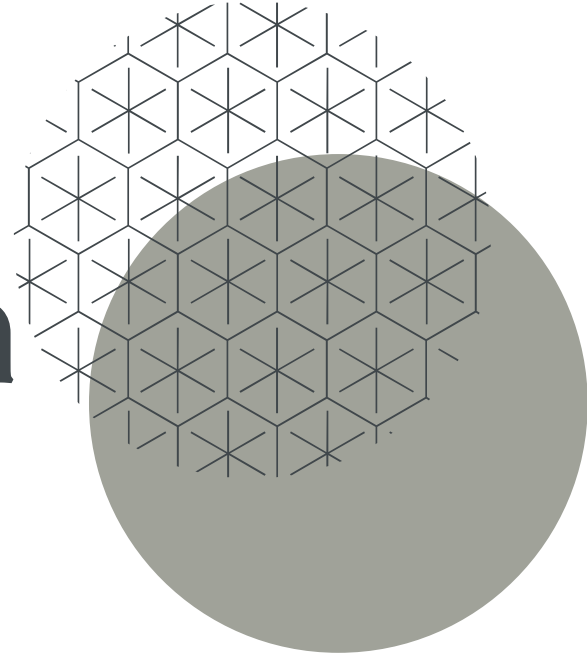
**search():** Busca una coincidencia en una cadena. Devuelve el índice de la coincidencia, o -1 si la búsqueda falla.

**replace():** Ejecuta una búsqueda de una coincidencia en una cadena, y sustituye la subcadena coincidente por una subcadena de reemplazo.

**replaceAll():** Ejecuta una búsqueda de todas las coincidencias en una cadena, y reemplaza las subcadenas coincidentes con una subcadena de reemplazo.



# Generación HTML

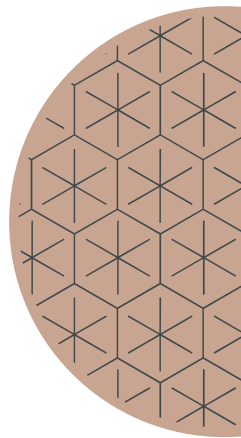


# Generación HTML

Generar HTML con Javascript a partir de algún evento lanzado por el usuario es algo muy común. Por ejemplo, que el usuario al hacer click a una cabecera, le aparezca el HTML con los elementos de esa tabla.

En esta unidad, aprenderemos a añadirlo de **tres** manera distintas:

- `Document.createElement()`
- `InnerHTML`
- `insertAdjacentHTML`



# Document.createElement()

En una página HTML, el método `document.createElement()` crea el elemento HTML especificado por `tagName`, `tagName` puede tener cualquiera de los valores de las etiquetas vistas hasta el momento.

# Document.createElement()

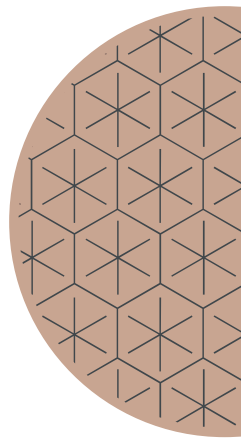
```
<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>
```

```
<script>  
const para = document.createElement("p");  
const node = document.createTextNode("This is new.");  
para.appendChild(node);  
  
const element = document.getElementById("div1");  
element.appendChild(para);  
</script>
```

# Document.createElement()

El método `appendChild()` en el ejemplo anterior, agrega el nuevo elemento como el último hijo del padre.

Si no quieres eso puedes usar el método `insertBefore()`:





# Document.createElement()

<script>

```
const para = document.createElement("p");
```

```
const node = document.createTextNode("This is new.");
```

```
para.appendChild(node);
```

```
const element = document.getElementById("div1");
```

```
const child = document.getElementById("p1");
```



```
element.insertBefore(para, child);
```

</script>

# InnerHTML

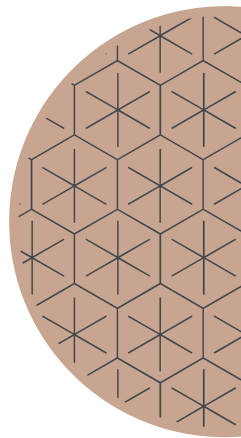
La propiedad innerHTML establece o devuelve el contenido HTML (HTML interno) de un elemento.

Este método es menos recomendado dado que añades HTML escapando con strings.



# InnerHTML

```
<!DOCTYPE html>
<html>
  <body>
    <h1>The Element Object</h1>
    <h2>The innerHTML Property</h2>
    <p id="myP">This is a p element.</p>
    <div id="myDIV">This is a div element.</div>
    <script>
      let text = "Hello Dolly";
      document.getElementById("myP").innerHTML = text;
      document.getElementById("myDIV").innerHTML = text;
    </script>
  </body>
</html>
```



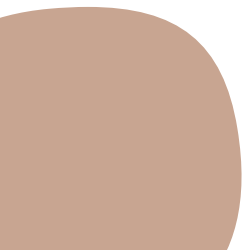
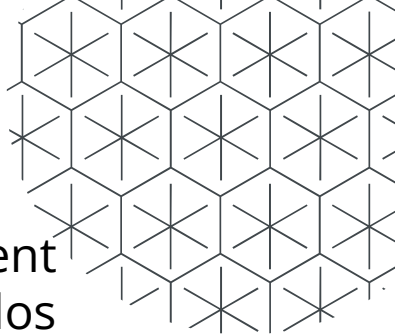
# insertAdjacentHTML

El método `insertAdjacentHTML()` de la interfaz `Element` analiza el texto especificado como HTML o XML e inserta los nodos resultantes en el árbol DOM en una posición determinada.

La sintaxis es:

*`insertAdjacentHTML(position, text)`*

Donde *text* es la cadena que debe ser analizada como HTML o XML e insertada en el árbol.



# Parámetros insertAdjacentHTML

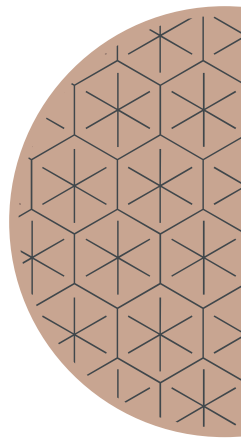
## posición

Una cadena que representa la posición relativa al elemento:

- **"beforebegin"**: Antes del elemento. Sólo es válido si el elemento está en el árbol DOM y tiene un elemento padre.
- **"afterbegin"**: Justo dentro del elemento, antes de su primer hijo.
- **"beforeend"**: Justo dentro del elemento, después de su último hijo.
- **"Afterend"**: Después del elemento. Sólo es válido si el elemento está en el árbol DOM y tiene un elemento padre.

# Parámetros insertAdjacentHTML

```
<!DOCTYPE html>
<html>
<body>
<h1>The Element Object</h1>
<h2>The insertAdjacentHTML() Method</h2>
<button onclick="myFunction()">Insert</button>
<p>Click "Insert" to insert a paragraph after the header.</p>
<h2 id="myH2">My Header</h2>
<script>
function myFunction() {
  const h2 = document.getElementById("myH2");
  let html = "<p>My new paragraph.</p>";
  h2.insertAdjacentHTML("afterend", html);
}
</script>
</body>
</html>
```





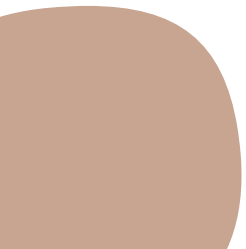
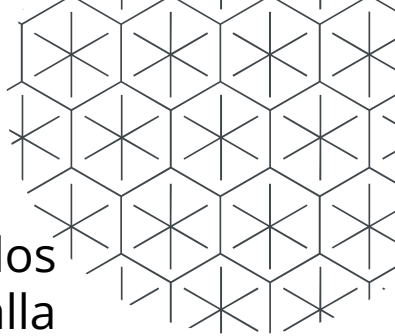
# Objeto Window

---

# Objeto Window

El objeto Window es una colección de propiedades y métodos que contienen información sobre el navegador y la pantalla del ordenador.

Por ejemplo, podemos averiguar qué navegador se está utilizando para ver una página (aunque, este método no es fiable). También podemos averiguar las dimensiones de la pantalla en la que se ve, y qué páginas se han visitado antes de la página actual.



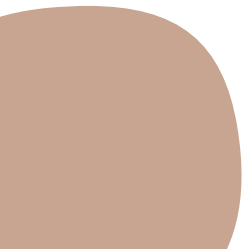
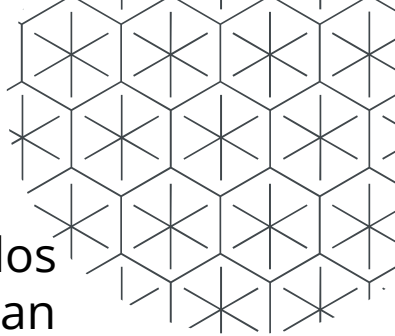


# Objeto Window

No existe un estándar oficial para la lista de materiales, aunque hay una serie de propiedades y métodos que son soportados por todos los principales navegadores, haciendo una especie de estándar de facto. Estas propiedades y métodos están disponibles a través del objeto Window. Todas las ventanas, pestañas, ventanas emergentes, marcos e iframes de los navegadores tienen un objeto Window.

# Dialogs

Hay tres funciones que producen diálogos en los navegadores: `alert()`, `confirm()` y `prompt()`. Estas no forman parte del estándar ECMAScript, aunque todos los principales navegadores las soportan como métodos del objeto `Window`.



# Información Browser

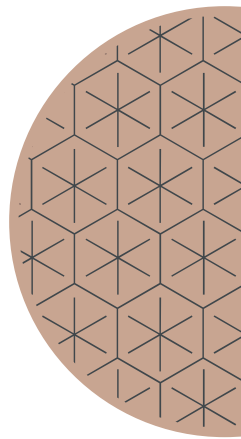
El objeto Window tiene una serie de propiedades y métodos que proporcionan información sobre el navegador del usuario.

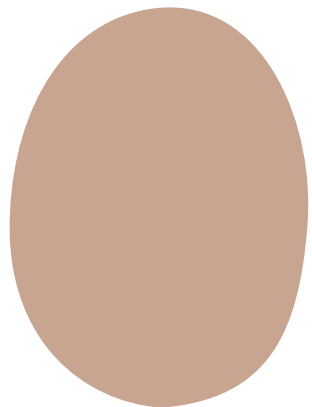
- La propiedad **navigator** que devuelve una referencia al objeto Navigator.
- La propiedad **location** es un objeto que contiene información sobre la URL de la página actual.
- La propiedad **history** se puede utilizar para acceder a la información sobre las páginas visitadas anteriormente en la sesión actual del navegador.

# Objeto Window

Por norma general, el objeto Windows no se suele utilizar, por ejemplo con la sentencia 'use strict' **no** se puede acceder al propio objeto.

Tampoco forma parte del estándar ECMAScript, aunque los principales navegadores los soportan como métodos del objeto ventana.





**FIN**

