

Introducción a Javascript

...

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

- Strings

- Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

- Objeto window

- Objeto document

- Objeto navigator

- Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Introducción

En el inicio de la era de Internet, **las páginas web eran estáticas**, mostraban un contenido fijo sin posibilidad de interacción con el usuario.

Pero cuando apareció "La Web 2.0" **se incrementó la interacción y usabilidad**, mejorando la experiencia de el usuario en la navegación.

Aquí es donde el lenguaje Javascript tiene un rol importante. La programación en el lado del cliente codificada en las páginas web, es la responsable de hacer páginas web atractivas tal y como las conocemos hoy en día.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Introducción

Es importante decir que el lenguaje ha evolucionado por dos lados:

- Por el propio lenguaje: Ha ido incorporando operadores, estructuras de control, reglas, etc.
- Por los navegadores: Han ido incorporando nuevas instrucciones.

Los autores de JavaScript, propusieron en 1997 a la European Computer Manufacturers Association (ECMA) que el lenguaje se definiera como estándar.

Así es como nació la **ECMAScript**, que es el estándar de definición del lenguaje Javascript. Desde entonces se han ido publicando distintas versiones de ECMAScript. En los inicios de Internet había bastantes problemas de compatibilidad entre navegadores, lo que hizo tomar conciencia de la importancia en la adopción de estándares.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

- Strings

- Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

- Objeto window

- Objeto document

- Objeto navigator

- Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Introducción

A partir de 2012, todos los navegadores actuales soportan completamente ECMAScript 5.1. Desde 2015, ECMA International, publicó la versión número 6, que oficialmente se llama ECMAScript 2015, y que inicialmente se llamó **ECMAScript 6** o ES6.

Con el fin de avanzar en la **compatibilidad entre navegadores web**, no sólo es importante que implementen el motor de JavaScript según los estándares ECMAScript, sino que los diferentes elementos que hay en una web (botones, cajas de texto, enlaces...) comporten de la misma manera y respondan a los mismos eventos (eventos).

Por eso el W3C definió el **Documento Object Model (DOM)**, o Modelo de Objetos del Documento, que define un estándar de objetos para representar documentos HTML.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Introducción

¿Qué es JavaScript?

JavaScript es un lenguaje de guiones (script, en inglés). Es conocido sobre todo por su uso en páginas web, pero también se puede utilizar para realizar tareas de programación y administración que nada tienen que ver con la web. **Javascript mejora una página HTML, añadiendo dinamismo y interacción del usuario.**

- **Lenguajes interpretados o de guiones:** Los lenguajes de programación se dividen, en cuanto a la forma de ejecutarse, entre los interpretados y los compilados. Algunos lenguajes interpretados se llaman también lenguajes de guiones (scripts en inglés). Los lenguajes interpretados se ejecutan mediante un intérprete, que procesa las órdenes que incluye el programa una a una. Los lenguajes compilados necesitan del compilador como paso previo a la ejecución.

Hay que tener en cuenta que Javascript es un lenguaje single-thread, es decir, tiene sólo un hilo de ejecución, por lo que las tareas se van ejecutando secuencialmente.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Introducción

Ventajas de Javascript en las páginas web

- Permite una gran cantidad de efectos dentro de las páginas. Entre otros: ventanas emergentes (Pop-up), desplazamientos (scrolls), transiciones de imágenes, etc.
- Añade interactividad con el usuario.
- Proporciona integración con otros conectores o extensiones (plugins): no proporciona sólo acceso a los objetos HTML, también proporciona acceso a objetos específicos del navegador como Adobe Acrobat, Media Player, etc.
- Permite validación de los formularios en el lado del cliente. Una validación inicial de los formularios es posible para eliminar simples errores, como por ejemplo: cómo asegurarse de que el formato de fecha, DNI, correo electrónico o teléfono son correctos. Como resultado, el usuario tiene una respuesta más rápida que si el control se hiciera en el lado del servidor. Permite acceder a información del sistema.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Introducción

Desventajas de JavaScript en las páginas web

- La **seguridad** es el principal problema en JavaScript. Los fragmentos de código JavaScript que se añaden a las páginas web se descargan en los navegadores y se ejecutan en el lado del cliente, permitiendo así la posibilidad de que uno malware se pueda ejecutar en la máquina cliente y así explotar alguna vulnerabilidad de seguridad conocida en las aplicaciones, navegadores o fines todo del sistema operativo.
- El **rendimiento** del lenguaje es inferior al de un lenguaje compilado.
- Tiende a introducir una enorme cantidad de fragmentos de código en nuestros sitios web. Esto se resuelve fácilmente almacenando el código JavaScript en archivos externos con extensión JS. Así, la página web queda mucho más limpia y legible.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Introducción

Es importante saber que Javascript distingue entre mayúsculas y minúsculas, es decir, es **case-sensitive**.

- Existen dos formas de separar instrucciones. La primera es a través del carácter punto y coma ; y la segunda es a través de un **salto de línea**. Por tanto el punto y coma al final de la instrucción no es obligatorio si la siguiente comienza a una nueva línea. Si están en la misma línea será obligatorio ponerlo. De todas formas, es recomendable hacer uso de él siempre.
- No tendrá en cuenta los espacios en blanco sucesivos ni los saltos líneas sucesivos.
- Utiliza el conjunto de caracteres Unicode.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

¿Dónde ubicamos el código?

El código Javascript se define a través del elemento **<script>**.

Éste tiene un atributo de tipo (type), que se utiliza para indicar el tipo de lenguaje que utilizaremos (en este caso será Javascript). En HTML 4 y XHTML 1.x el tipo de atributo era obligatorio, en cambio con HTML5 no lo es.

Hay dos maneras:

- **Javascript dentro de la página:** Para situar el código JavaScript directamente en la página web, se coloca dentro del elemento `<script>`, en el `<head>` o en el `<body>`.
- **Javascript externo:** Es conveniente en muchos casos, escribir el código JavaScript en un archivo externo con la extensión ".js". Este archivo se llama desde la página web usando el elemento `<script>` usando el atributo `src`, y puede ir en el `<head>` o en el `<body>`.

Ejemplo: `<script src="archivo.js"> </script>`

El elemento `<script>` se debe cerrar siempre con `</script>`.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

¿Dónde ubicamos el código?

Exemple: Javascript dins la pàgina al head.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
    <script>
      alert("Bienvenido a la página.");
    </script>
  </head>
  <body>
  </body>
</html>
```

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

¿Dónde ubicamos el código?

Exemple: Javascript dins la pàgina al body.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <script>
      alert("Bienvenido a la página");
    </script>
  </body>
</html>
```

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

¿Dónde ubicamos el código?

Exemple: Javascript externo en el head.

`<> index.html`



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
    <script src="archivo.js"></script>
  </head>
  <body>
  </body>
</html>
```

`JS archivo.js`



```
alert("Bienvenido a la página");
```

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

¿Dónde ubicamos el código?

Exemple: Javascript externo en el body.

`index.html`



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <script src="archivo.js"></script>
  </body>
</html>
```

`JS archivo.js`



```
alert("Bienvenido a la página");
```

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Comentarios

Un comentario permite añadir anotaciones legibles al programador en el código del programa que son ignoradas por el compilador o el intérprete.

En javascript cuando el comentario ocupa **una sola línea** se escribirá a partir de dos barras //, y cuando el comentario se escribe en **más de una línea** irá entre /* */.

```
// Este es un comentario de una única línea
alert ("Ejemplo de comentarios en una línea");

alert ("Otro ejemplo en una línea") //Aquí hay otro comentario en una línea

alert ("Ejemplo de comentario multilínea");
/* Una línea de comentario que
tiene otra línea de comentario,
y que puede tener más */

alert ("Otro de comentario multilínea"); /* Una línea de comentario que
tiene otra línea de comentario,
y que puede tener más */
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Visualización y recogida de datos

Se pueden mostrar datos de distintas formas:

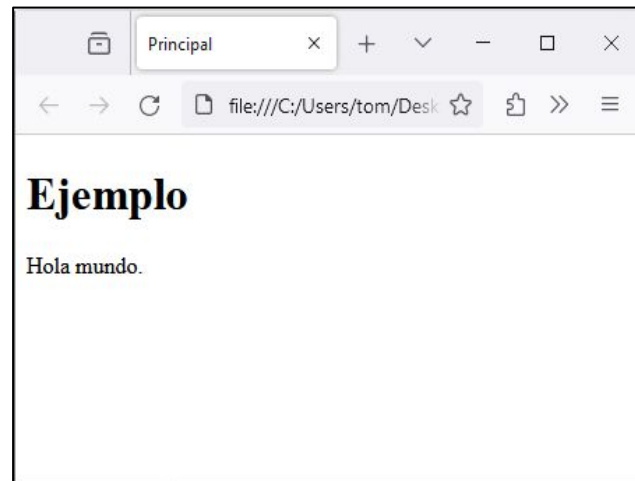
- Escribiendo en la salida de HTML utilizando el método de acceso al DOM: "**document.write()**". El DOM nos permite acceder al código HTML y modificarlo.
- Creando un cuadro o ventana emergente utilizando: **window.alert()**, o simplemente **alert()**. Éste se utiliza para interaccionar con el navegador. Es lo mismo utilizar "window.alert()" o directamente "alert()", ya que window hace referencia al objeto global.
- Escribiendo directamente en la consola del navegador utilizando: **console.log()**.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos**
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Visualización y recogida de datos

Ejemplo: Uso de document.write()

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      document.write("Hola mundo.");
    </script>
  </body>
</html>
```

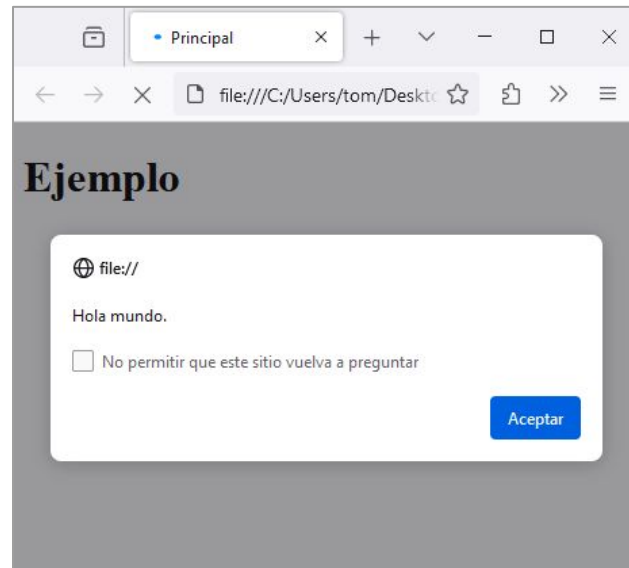


- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos**
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Visualización y recogida de datos

Ejemplo: Uso de window.alert()

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      window.alert("Hola mundo.");
    </script>
  </body>
</html>
```



- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos**
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Visualización y recogida de datos

Ejemplo: Uso de console.log()

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      console.log("Hola mundo.");
    </script>
  </body>
</html>
```



Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Visualización y recogida de datos

Confirmación de la salida de datos.

- Uso de `window.confirm()`

Este método nos muestra una ventana de diálogo con un mensaje y dos botones: Aceptar y Cancelar, por lo que permite que el usuario acepte continuar o cancelar la acción.

Sintaxis:

```
resultado = window.confirm(mensaje);
```

Donde:

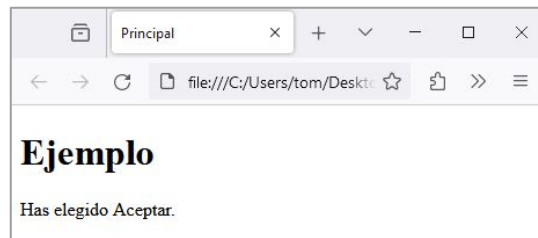
- mensaje: es la cadena que se muestra opcionalmente en el cuadro de diálogo.
- resultado: Es la variable donde se guardará el valor resultante de elegir Aceptar o Cancelar y que será `true` si ha elegido Aceptar y `false` si ha elegido Cancelar.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos**
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Visualización y recogida de datos

Ejemplo: Uso de windows.confirm()

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      resultado=window.confirm("¿Eliminar el archivo?");
      if(resultado == true) {
        // resultado = true
        document.write("Has elegido Aceptar.");
      }
      else {
        // resultado = false
        document.write("Has elegido Cancelar.");
      }
    </script>
  </body>
</html>
```



Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Visualización y recogida de datos

Recoger datos con prompt.

- Uso de prompt()

Nos muestra un cuadro de diálogo que nos permite mostrar el mensaje con un ventana emergente, a la vez que recoger **un valor de tipo cadena de texto** (un string).

Sintaxis:

```
info = prompt(texto);
```

```
info = prompt(texto, "Valor de texto por defecto");
```

Donde:

- texto: Es el mensaje que se mostrará por pantalla pidiendo una información o un valor.
- Valor de texto por defecto (Opcional): Para escribir un valor o una información por defecto por si el usuario no escribe. Puede ser un texto literal o una variable.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

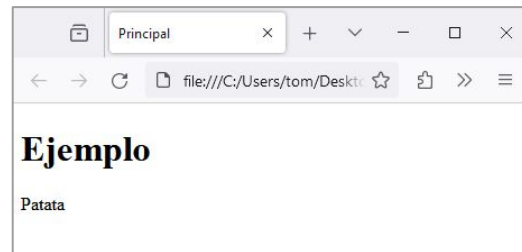
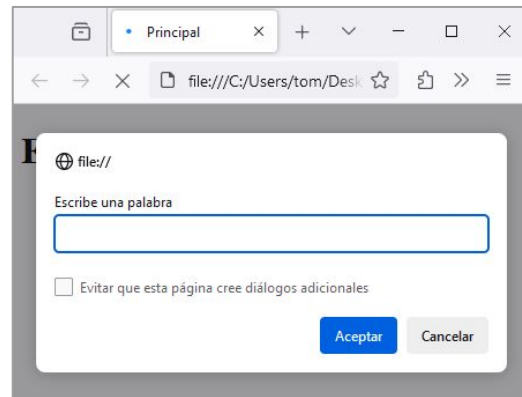
Validación de formularios

Expresiones regulares

Visualización y recogida de datos

Ejemplo: Uso de `prompt()` sin valor por defecto. No aparece valor por defecto y podemos elegir cualquier valor.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      info = prompt("Escribe una palabra");
      document.write(info);
    </script>
  </body>
</html>
```



Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

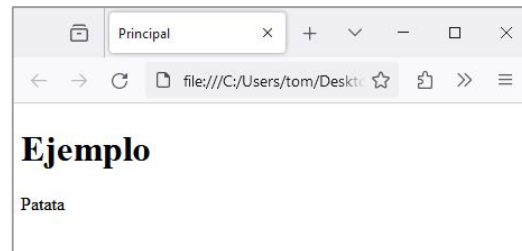
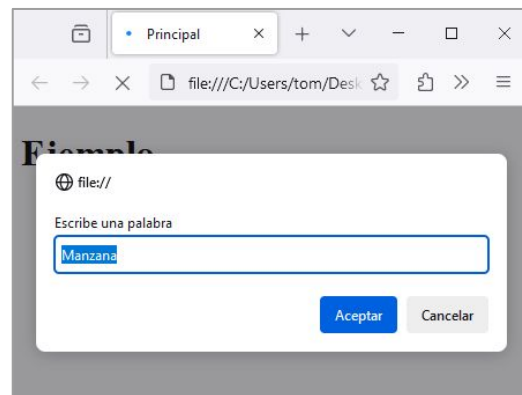
Validación de formularios

Expresiones regulares

Visualización y recogida de datos

Ejemplo: Uso de `prompt()` con valor por defecto. Aparece el valor por defecto pero podemos elegir cualquier otro valor.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      info = prompt("Escribe una palabra","Manzana");
      document.write(info);
    </script>
  </body>
</html>
```



Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Variables

Identificadores

Los identificadores **son nombres y deben ser únicos**. Se utilizan identificadores para nombrar variables, palabras clave, funciones y etiquetas.

Las normas para nombres legales o permitidos, son lo mismo en la mayoría de lenguajes de programación:

- El primer carácter debe ser una letra, un guión bajo _ o un signo de dólar \$.
- Los caracteres posteriores pueden ser letras, dígitos, guiones bajos o signos de dólar.
- No se permiten números como primer carácter.

Ejemplo:

- identificadores válidos: contador, suma3, \$manzanas, _total, suma_total
- identificadores no válidos: 3manzanas, num#, %total

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Variables

La creación de una variable en Javascript se llama **declarar una variable**. Declarar una variable significa que se reserva espacio de almacenamiento en memoria.

Las variables deben tener identificadores únicos y deben cumplir las siguientes normas:

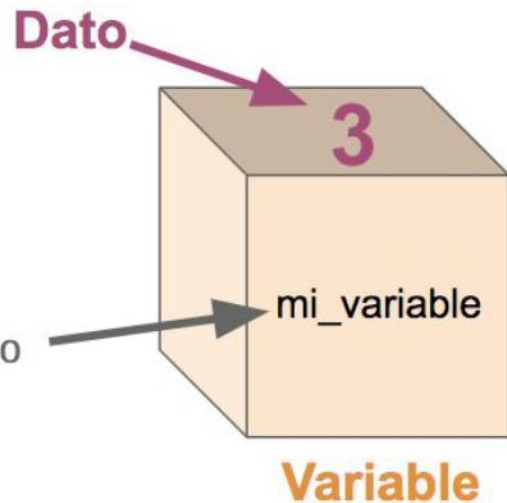
- Sólo pueden contener letras, dígitos, signo de subrayado _ y el signo de dólar \$.
- Deben empezar con una letra, o bien con el signo de subrayado _ o el signo de dólar \$.
- Hay que recordar que se distingue entre mayúsculas y minúsculas (la variable x es diferente de la variable X).
- No se pueden utilizar las palabras reservadas de Javascript (while, for, next, etc. que se indicaran más adelante).

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
- Variables**
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Variables

Variables JS

Cada variable tiene un nombre, de modo que podamos acceder a ese dato siempre que necesitemos.



Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Variables

Declaración de variables.

Antes de ES6, la única forma de crear variables en Javascript estaba con la palabra `var`. Pero a partir de este estándar hay cuatro tipos de declaraciones en JavaScript:

- Directamente declara variables locales o globales en función del contexto de ejecución.

```
info = "Información";
```

- **var:** Declara variables locales o globales en función del contexto de ejecución.

```
var info = "Información";
```

- **let:** Declara variables locales con ámbito de **bloque**.

```
let info = "Información";
```

- **const:** Declara constantes de sólo lectura con ámbito de bloque.

```
const info = "Información";
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables**
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Variables

Ámbito de las variables

El ámbito de las variables es el lugar en el que están disponibles.

Cuando declaras una variable fuera de cualquier función, se denomina **variable global**, porque está disponible para cualquier otro código en el documento actual.

Cuando declaras una variable dentro de una función, se llama **variable local**, porque solo está disponible dentro de esa función. En el caso de usar `let` también son variables locales si se declaran dentro de un bloque `{}`.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Variables

Ejemplo: Ámbito de variables declaradas directamente

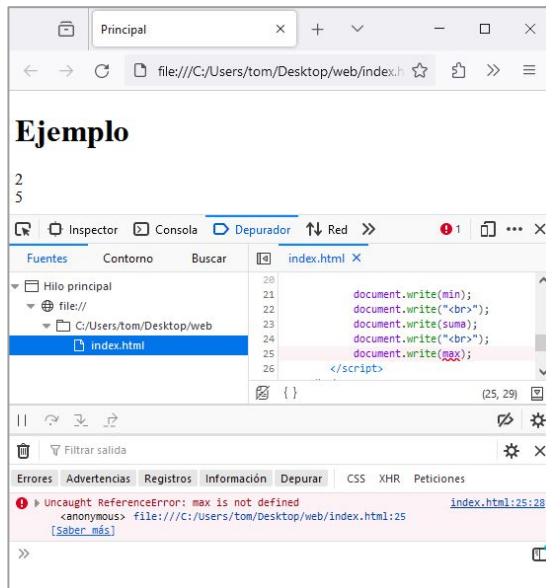
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      min = 2;

      if(true)
      {
        suma = 5
      }

      function funcion() {
        max = 10;
      }

      document.write(min);
      document.write("<br>");
      document.write(suma);
      document.write("<br>");
      document.write(max);
    </script>
  </body>
</html>
```

Tenemos la variable min declarada fuera de una función y de un bloque {}, una variable llamada suma declarada dentro de un bloque {} y una variable max declarada dentro de una función.



Podemos ver que puede acceder a las variables globales min y suma, en cambio da un error al acceder a la variable local max ya que no está en su ámbito.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Variables

Ejemplo: Ámbito de variables declaradas con **var**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      var min = 2;

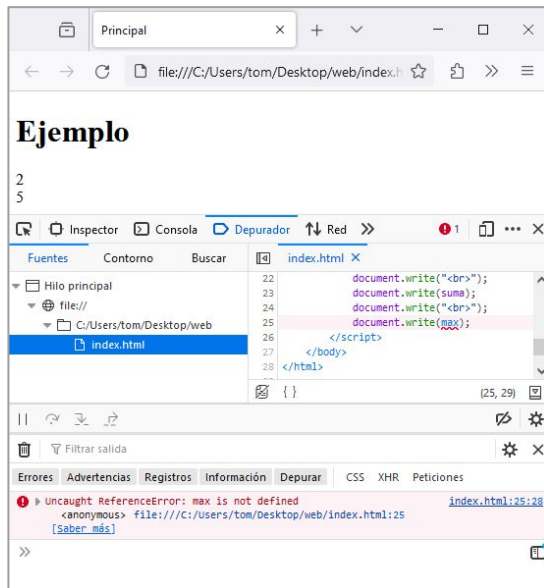
      if(true)
      {
        var suma = 5
      }

      function funcion() {
        var max = 10;
      }

      document.write(min);
      document.write("<br>");
      document.write(suma);
      document.write("<br>");
      document.write(max);

    </script>
  </body>
</html>
```

Tenemos la variable `min` declarada fuera de una función y de un bloque {}, una variable llamada `suma` declarada dentro de un bloque {} y una variable `max` declarada dentro de una función.



Podemos ver que puede acceder a las variables globales `min` y `suma`, en cambio da un error al acceder a la variable local `max` ya que no está en su ámbito.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Variables

Ejemplo: Ámbito de variables declaradas con **let**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      let min = 2;

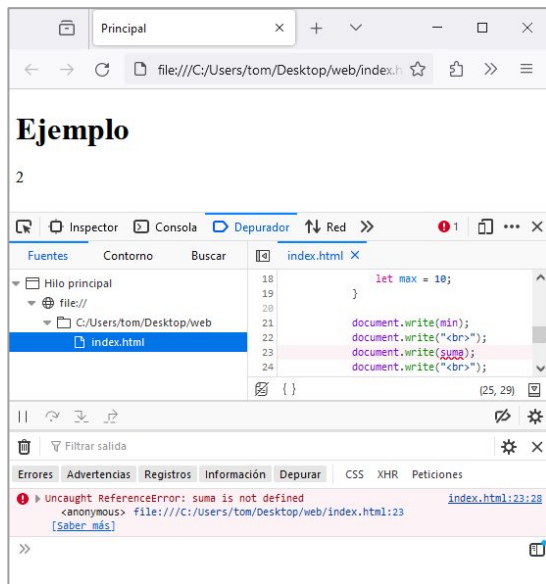
      if(true)
      {
        let suma = 5
      }

      function funcion() {
        let max = 10;
      }

      document.write(min);
      document.write("<br>");
      document.write(suma);
      document.write("<br>");
      document.write(max);

    </script>
  </body>
</html>
```

Tenemos la variable `min` declarada fuera de una función y de un bloque {}, una variable llamada `suma` declarada dentro de un bloque {} y una variable `max` declarada dentro de una función.



Podemos ver que puede acceder a la variable global `min`, en cambio da un error al acceder a la variable local `suma` ya que no está en su ámbito.

Aunque no lo indica ya que se queda en el primer error, también daría error al acceder a la variable local `max`.

Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Variables

Ejemplo: Ámbito de variables declaradas con **const**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      const min = 2;

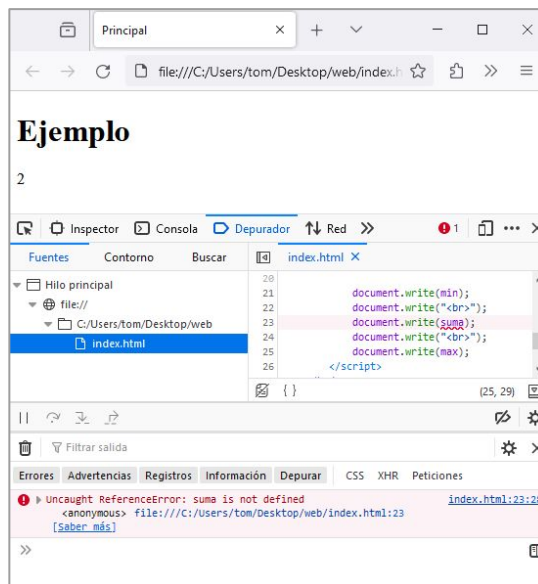
      if(true)
      {
        const suma = 5
      }

      function funcion() {
        const max = 10;
      }

      document.write(min);
      document.write("<br>");
      document.write(suma);
      document.write("<br>");
      document.write(max);

    </script>
  </body>
</html>
```

Tenemos la variable `min` declarada fuera de una función y de un bloque {}, una variable llamada `suma` declarada dentro de un bloque {} y una variable `max` declarada dentro de una función. Todas son de sólo lectura.



Podemos ver que puede acceder a la variable global de sólo lectura `min`, en cambio da un error al acceder a la variable local de sólo lectura `max` ya que no está en su ámbito.

Aunque no lo indica ya que se queda en el primer error, también daría error al acceder a la variable local de sólo lectura `max`.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables**
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Variables

Redeclaración de las variables

Las variables declaradas **directamente** y las variables declaradas con **var** se pueden volver a declarar.

Las variables declaradas con **let** y **const** no se pueden volver a declarar ya que se producirá un error.

El hecho de que se puedan redeclarar variables puede producir errores en algunos casos difíciles de encontrar, por lo que se recomienda el uso de **let** y **const** para declarar las variables.

En la siguiente tabla tenemos un resumen:

	Directamente	var	let	const
Permite modificar los valores	Sí	Sí	Sí	No
Se puede redeclarar la variable	Sí	Sí	No	No

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
- Variables**
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Variables

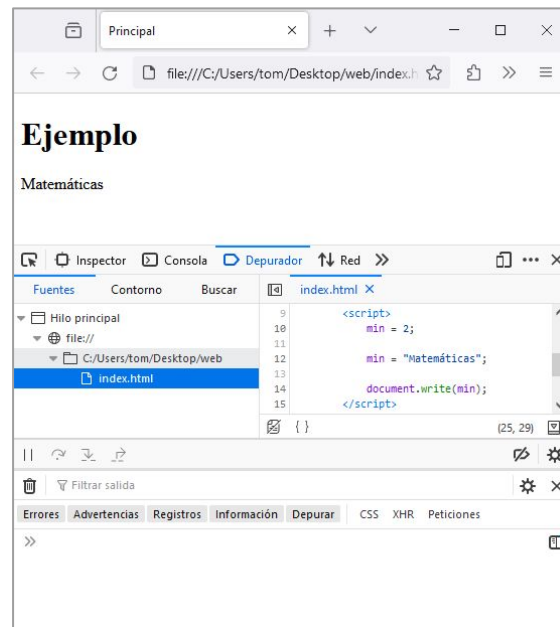
Ejemplo: Redeclaración de variables declaradas **directamente**

No hay problema al redeclarar la variable declarada anteriormente.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      min = 2;

      min = "Matemáticas";

      document.write(min);
    </script>
  </body>
</html>
```



- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables**
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Variables

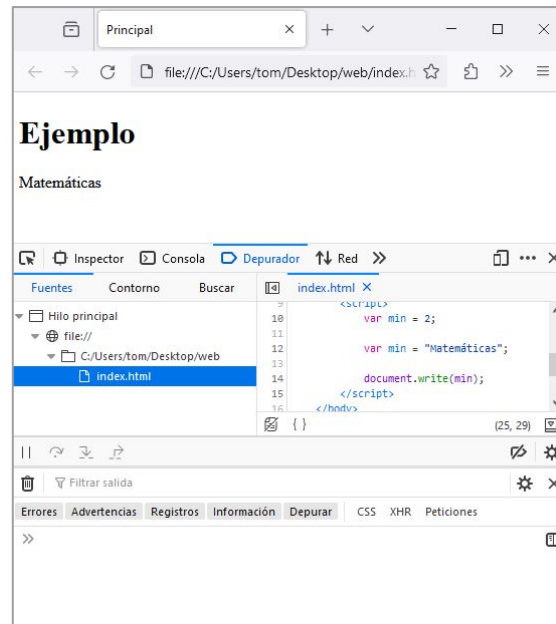
Ejemplo: Redeclaración de variables declaradas con **var**

No hay problema al redeclarar la variable declarada anteriormente con var.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      var min = 2;

      var min = "Matemáticas";

      document.write(min);
    </script>
  </body>
</html>
```



Introducción

¿Dónde ubicamos el código?

Comentarios

Visualización y recogida de datos

Variables

Valores

Expresiones

Palabras reservadas de JavaScript

Tipos de datos

Operadores

Conversión de tipos

Estructuras de control

Estructuras básicas de datos

Strings

Arrays

Funciones

Manejo de errores

Objetos

LocalStorage

Introducción al DOM

Objeto window

Objeto document

Objeto navigator

Acceso al DOM

Eventos

Validación de formularios

Expresiones regulares

Variables

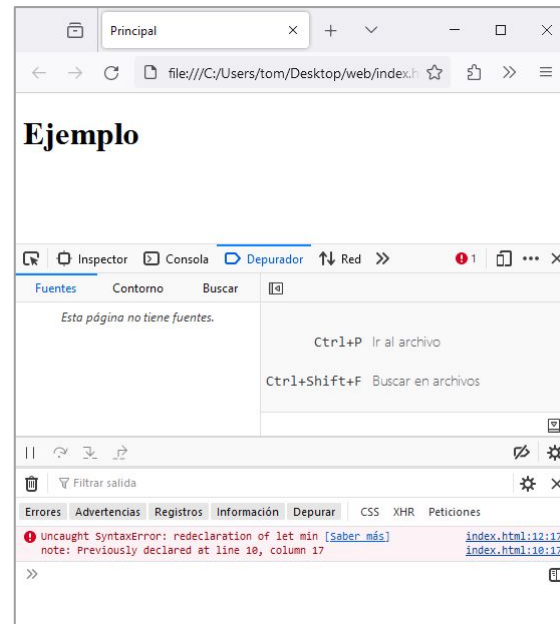
Ejemplo: Redeclaración de variables declaradas con **let**

Se produce un error al redeclarar la variable declarada anteriormente con **let**.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      let min = 2;

      let min = "Matemáticas";

      document.write(min);
    </script>
  </body>
</html>
```



- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables**
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Variables

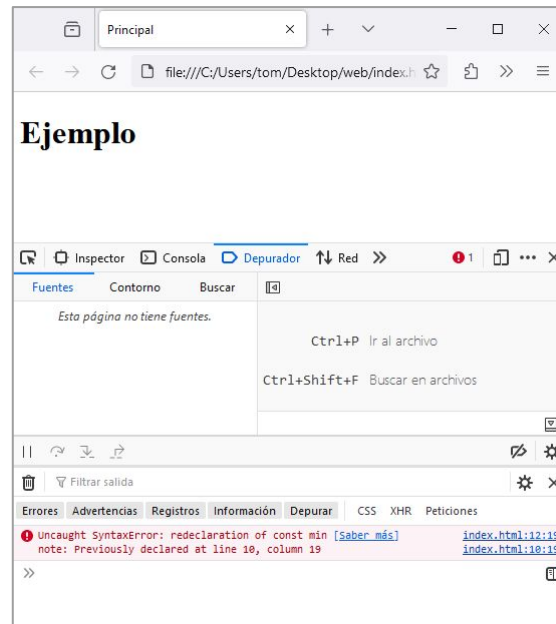
Ejemplo: Redeclaración de variables declaradas con **let**

Se produce un error al redeclarar la variable declarada anteriormente con **let**.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Principal</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      const min = 2;

      const min = "Matemáticas";

      document.write(min);
    </script>
  </body>
</html>
```

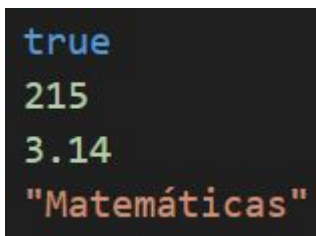


Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Valores

La sintaxis de JavaScript define dos tipos de valores: valores fijos y valores variables:

- Los **valores fijos** se llaman literales. Se utilizan para representar valores en Javascript. Cómo su nombre indica son literalmente proporcionados por el programador en el código.



```
true
215
3.14
"Matemáticas"
```

- Los **valores variables** son las propias variables que almacenan valores susceptibles de ser cambiados.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
Strings
Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
Objeto window
Objeto document
Objeto navigator
Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Expresiones

Una expresión es una combinación de valores, variables y operadores, que al calcularse se reduce a un valor.

- El cálculo se llama **evaluación**.
- Los valores pueden ser de varios tipos, como números y cadenas.

Ejemplo: Expresiones y su evaluación a un valor resultado.

Expresión	Resultado de la evaluación
<code>true && false</code>	<code>false</code>
<code>2*(3+5)</code>	<code>16</code>
<code>3.14 + 2 * 5</code>	<code>13.14</code>
<code>"Matemáticas" + " son fáciles"</code>	<code>"Matemáticas son fáciles"</code>

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Palabras reservadas de JavaScript

Palabras reservadas de Javascript (ES6):

- break
- case
- catch
- class
- const
- continue
- debugger
- default
- delete
- do
- else
- export
- extends
- finally
- for
- function
- if
- import
- in
- instanceof
- new
- return
- super
- switch
- this
- throw
- try
- typeof
- var
- void
- while
- with
- yield

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos**
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Tipos de datos

Los **tipos de datos** en JavaScript son algo diferentes en comparación con otros lenguajes de programación como C o JAVA. JavaScript es un **lenguaje débilmente tipado**, esto significa que no es necesario definir el tipo de dato. Pero esto **no significa que no tenga un tipo de dato**, puesto que éste se define tiempo de ejecución por JavaScript.

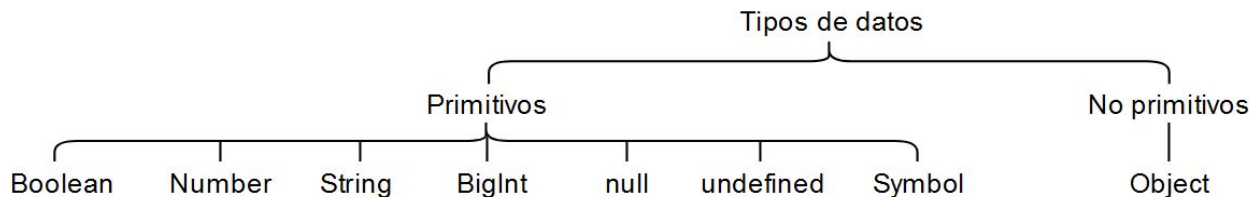
Este comportamiento y especificación de los tipos de datos aplica para cualquier lugar donde se ejecute JavaScript.

Se dice que JavaScript tiene un sistema de tipos de datos dinámico, esto es porque las variables que se definen pueden recibir cualquier otro tipo de dato en cualquier momento, e ir cambiando de tipos según el valor que guardan. Lo que realmente nos interesa es el valor y que podemos hacer con ese valor, no tanto el tipo.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
 LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Tipos de datos

Como podemos ver tenemos dos grandes tipos de datos: los tipos primitivos y los tipos no primitivos.



Tipos primitivos

- Este tipo de datos son inmutables.
- Sus valores contienen sólo un dato.
- Después de asignar a una variable un valor primitivo, si se quiere cambiar este valor, es necesario reasignar uno nuevo.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Tipos de datos

- **null:** Se utiliza para indicar que algo existe pero no tiene valor, su valor será **null**. Entonces diremos que es del tipo null.

```
null
```

- **undefined:** Representa algo que no está definido. Es el valor automático de las variables, parámetros y propiedades de objetos que todavía no tienen ningún valor definido.

```
undefined
```

- **Symbol:** Se utiliza principalmente para garantizar que su valor sea único e inmutable. Puede utilizarse como propiedades de objetos para reducir accesos a estas propiedades.

```
Symbol(3)  
Symbol("Hola")  
Symbol(suma)
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Tipos de datos

- **Object:** El Object se refiere a una estructura de datos que contiene datos e instrucciones para trabajar con los datos. Los objetos también permiten interaccionar con otros objetos.

Algunas veces los objetos se asemejan a cosas del mundo real, por ejemplo, un objeto coche en el caso de un juego de carreras. Tiene unas propiedades (datos) que lo caracterizan y unas acciones (instrucciones) que se pueden realizar.

Trataremos los objetos más adelante.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

Los **operadores** son acciones que se realizan sobre tipos de datos a los que llamamos **operandos** y que normalmente dan como resultado otro operando. Usando los tipos de datos y los operandos podemos crear las expresiones de las que hemos hablado con anterioridad.

JavaScript tiene operadores **unarios**, **binarios** y un operador **ternario** especial, el operador condicional.

Los operadores binarios tienen un solo operando, los binarios requieren dos operandos y el ternario requiere tres operadores.

Existe una precedencia de operadores que indica el orden en el que se deben ejecutar los operadores en una expresión.

- Operadores aritméticos
- Operadores de comparación
- Operadores de asignación
- Operadores lógicos
- Operadores de cadena
- Operador condicional (ternario)
- Operadores bit a bit
- Operador coma
- Operadores unarios
- Operadores relacionales

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores aritméticos:** Un operador aritmético toma valores numéricos (ya sean literales o variables) como sus operandos y devuelve un solo valor numérico.

Operador	Descripción	Ejemplo
Residuo (%)	Operador binario. Devuelve el resto entero de dividir los dos operandos.	12 % 5 devuelve 2.
Incremento (++)	Operador unario. Agrega uno a su operando. Si se usa como operador prefijo (++x), devuelve el valor de su operando después de agregar uno; si se usa como operador sufijo (x++), devuelve el valor de su operando antes de agregar uno.	Si x es 3, ++x establece x en 4 y devuelve 4, mientras que x++ devuelve 3 y, solo entonces, establece x en 4.
Decremento (--)	Operador unario. Resta uno de su operando. El valor de retorno es análogo al del operador de incremento.	Si x es 3, entonces --x establece x en 2 y devuelve 2, mientras que x-- devuelve 3 y, solo entonces, establece x en 2.
Negación unaria (-)	Operador unario. Devuelve la negación de su operando.	Si x es 3, entonces -x devuelve -3.
Positivo unario (+)	Operador unario. Intenta convertir el operando en un número, si aún no lo es.	+"3" devuelve 3. +true devuelve 1.
Operador de exponenciación (**)	Calcula la base a la potencia de exponente, es decir, baseexponente	2 ** 3 returns 8. 10 ** -1 returns 0.1.

Introducción
 ¿Dónde ubicamos el código?
 Comentarios
 Visualización y recogida de datos
 Variables
 Valores
 Expresiones
 Palabras reservadas de JavaScript
 Tipos de datos
Operadores
 Conversión de tipos
 Estructuras de control
 Estructuras básicas de datos
 Strings
 Arrays
 Funciones
 Manejo de errores
 Objetos
 localStorage
 Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
 Eventos
 Validación de formularios
 Expresiones regulares

Operadores

- **Operadores de comparación:** Un operador de comparación compara sus operandos y devuelve un valor lógico en función de si la comparación es verdadera (true) o falsa (false). En los ejemplos var1 tiene valor 3 y var2 valor 4.

Operador	Descripción	Ejemplos que devuelven true
Igual (==)	Devuelve true si los operandos son iguales.	3 == var1 "3" == var1 3 == '3'
No es igual (!=)	Devuelve true si los operandos no son iguales.	var1 != 4 var2 != "3"
Estrictamente igual (===)	Devuelve true si los operandos son iguales y del mismo tipo. Consulta también Object.is y similitud en JS .	3 === var1
Desigualdad estricta (!==)	Devuelve true si los operandos son del mismo tipo pero no iguales, o son de diferente tipo.	var1 !== "3" 3 !== '3'
Mayor que (>)	Devuelve true si el operando izquierdo es mayor que el operando derecho.	var2 > var1 "12" > 2
Mayor o igual que (>=)	Devuelve true si el operando izquierdo es mayor o igual que el operando derecho.	var2 >= var1 var1 >= 3
Menor que (<)	Devuelve true si el operando izquierdo es menor que el operando derecho.	var1 < var2 "2" < 12
Menor o igual que (<=)	Devuelve true si el operando izquierdo es menor o igual que el operando derecho.	var1 <= var2 var2 <= 5

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos

Operadores

Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores de asignación:** Un operador de asignación asigna un valor a su operando izquierdo basándose en el valor de su operando derecho.

Nombre	Operador abreviado	Significado
Asignación	<code>x = y</code>	<code>x = y</code>
Asignación de adición	<code>x += y</code>	<code>x = x + y</code>
Asignación de resta	<code>x -= y</code>	<code>x = x - y</code>
Asignación de multiplicación	<code>x *= y</code>	<code>x = x * y</code>
Asignación de división	<code>x /= y</code>	<code>x = x / y</code>
Asignación de residuo	<code>x %= y</code>	<code>x = x % y</code>
Asignación de exponenciación	<code>x **= y</code>	<code>x = x ** y</code>
Asignación de desplazamiento a la izquierda	<code>x <<= y</code>	<code>x = x << y</code>
Asignación de desplazamiento a la derecha	<code>x >>= y</code>	<code>x = x >> y</code>
Asignación de desplazamiento a la derecha sin signo	<code>x >>>= y</code>	<code>x = x >>> y</code>
Asignación AND bit a bit	<code>x &= y</code>	<code>x = x & y</code>
Asignación XOR bit a bit	<code>x ^= y</code>	<code>x = x ^ y</code>
Asignación OR bit a bit	<code>x = y</code>	<code>x = x y</code>
Asignación AND lógico	<code>x &&= y</code>	<code>x && (x = y)</code>
Asignación OR lógico	<code>x = y</code>	<code>x (x = y)</code>
Asignación de anulación lógica	<code>x ??= y</code>	<code>x ?? (x = y)</code>

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores lógicos:** Los operadores lógicos se utilizan normalmente con valores booleanos (lógicos); cuando lo son, devuelven un valor booleano.

Sin embargo, los operadores `&&` y `||` en realidad devuelven el valor de uno de los operandos especificados, por lo que si estos operadores se utilizan con valores no booleanos, pueden devolver un valor no booleano.

Operador	Uso	Descripción
AND lógico (<code>&&</code>)	<code>expr1 && expr2</code>	Devuelve <code>expr1</code> si se puede convertir a <code>false</code> ; de lo contrario, devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code>&&</code> devuelve <code>true</code> si ambos operandos son <code>true</code> ; de lo contrario, devuelve <code>false</code> .
OR lógico (<code> </code>)	<code>expr1 expr2</code>	Devuelve <code>expr1</code> si se puede convertir a <code>true</code> ; de lo contrario, devuelve <code>expr2</code> . Por lo tanto, cuando se usa con valores booleanos, <code> </code> devuelve <code>true</code> si alguno de los operandos es <code>true</code> ; si ambos son falsos, devuelve <code>false</code> .
NOT lógico (<code>!</code>)	<code>!expr</code>	Devuelve <code>false</code> si su único operando se puede convertir a <code>true</code> ; de lo contrario, devuelve <code>true</code> .

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores de cadena:** Además de los operadores de comparación, que se pueden usar en valores de cadena, el operador de concatenación (+) concatena dos valores de cadena, devolviendo otra cadena que es la unión de los dos operandos de cadena.

```
var nombre = "Pepe";  
var saludos = "Hola" + nombre;  
console.log(saludos);  
// La consola registra la cadena "Hola Pepe"
```

- **Operador condicional (ternario):** Se trata de un if comprimido en una línea. El operador según si la **condición** es verdadera tomará el **primer valor** y si es falsa el **segundo valor**. Los operandos pueden ser literales o variables. La sintaxis es:

condición ? valor1 : valor2

```
var a = 3;  
var b = 5;  
var mayor = (a > b) ? a : b;  
console.log("El mayor de los números es " + mayor + ".");  
// La consola registra: El mayor de los números es 5.
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos

Operadores

Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores bit a bit:** Un operador bit a bit trata a sus operandos como un conjunto de 32 bits (ceros y unos), en lugar de números decimales, hexadecimales u octales.

Operador	Uso	Descripción
AND a nivel de bits	<code>a & b</code>	Devuelve un uno en cada posición del bit para los que los bits correspondientes de ambos operandos son unos.
OR a nivel de bits	<code>a b</code>	Devuelve un cero en cada posición de bit para el cual los bits correspondientes de ambos operandos son ceros.
XOR a nivel de bits	<code>a ^ b</code>	Devuelve un cero en cada posición de bit para la que los bits correspondientes son iguales. [Devuelve uno en cada posición de bit para la que los bits correspondientes son diferentes].
NOT a nivel de bits	<code>~ a</code>	Invierte los bits de su operando.
Desplazamiento a la izquierda	<code>a << b</code>	Desplaza <code>a</code> en representación binaria <code>b</code> bits hacia la izquierda, desplazándose en ceros desde la derecha.
Desplazamiento a la derecha de propagación de signo	<code>a >> b</code>	Desplaza <code>a</code> en representación binaria <code>b</code> bits a la derecha, descartando los bits desplazados.
Desplazamiento a la derecha de relleno cero	<code>a >>> b</code>	Desplaza <code>a</code> en representación binaria <code>b</code> bits hacia la derecha, descartando los bits desplazados y desplazándose en ceros desde la izquierda.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores coma:** El operador coma , simplemente evalúa ambos operandos y devuelve el valor del último operando.

Este operador se utiliza principalmente dentro de un bucle for, para permitir que se actualicen múltiples variables cada vez a través del bucle. Se considera de mal estilo usarlo en otros lugares, cuando no es necesario.

```
for(let suma = 1, i=1; i<=5; i++, suma += i) {  
  console.log("Iteración " + i);  
  console.log("Suma " + suma);  
}
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores**
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Operadores

- **Operadores unarios:** Una operación unaria es una operación con un solo operando.
 - **typeof:** Devuelve una cadena que indica el tipo de dato del operando. Los paréntesis son opcionales.

```
var myFun = new Function("5 + 2");
var shape = "round";
var size = 1;
var num = 43;
var certain = true;
var foo = ["Apple", "Mango", "Orange"];
var today = new Date();

typeof myFun; // devuelve "function"
typeof shape; // devuelve "string"
typeof size; // devuelve "number"
typeof (num); // devuelve "number"
typeof certain; // devuelve "boolean"
typeof foo; // devuelve "object"
typeof today; // devuelve "object"
typeof doesntExist; // devuelve "undefined"
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores unarios: ...**

- **delete:** El operador delete elimina la propiedad de un objeto.

Si el operador delete tiene éxito, elimina la propiedad del objeto. El operador delete devuelve true si la operación es posible; devuelve false si la operación no es posible.

Dado que los arrays son objetos, técnicamente es posible eliminar elementos del array con delete. Sin embargo, esto se considera una mala práctica, trata de evitarlo. Cuando eliminas una elemento de un array con delete, la longitud del array no se ve afectada y los otros elementos no se vuelven a indexar.

```
var persona = {nombre: "Pepe", edad: 28};

delete persona.edad;

console.log(persona);
```

```
>> var persona = {nombre: "Pepe", edad: 28};
← undefined
>> console.log(persona);
  ► Object { nombre: "Pepe", edad: 28 }
← undefined
>> delete persona.edad;
← true
>> console.log(persona);
  ► Object { nombre: "Pepe" }
← undefined
```

Operadores

- **Operadores unarios: ...**

- **void:** El operador void especifica una expresión que se evaluará sin devolver un valor.

Los paréntesis que rodean la expresión son opcionales, pero es un buen estilo usarlos.

```
>> suma = void (3+5)
```

```
← undefined
```

```
>> suma
```

```
← undefined
```

```
>> void ("hola")
```

```
← undefined
```


Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Operadores

- **Operadores relacionales:** Un operador relacional compara sus operandos y devuelve un valor Boolean basado en si la comparación es verdadera.
 - **in:** El operador in devuelve **true** si la propiedad especificada está en el objeto especificado y **false** en el caso contrario.

```
var persona = {nombre: "Pepe", edad: 28};  
  
console.log(edad in persona);
```

```
>> var persona = {nombre: "Pepe", edad: 28};  
← undefined  
  
>> persona  
← ▶ Object { nombre: "Pepe", edad: 28 }  
  
>> console.log("edad" in persona);  
true  
← undefined  
  
>> console.log("dni" in persona);  
false  
← undefined
```

Operadores

- **Operadores relacionales: ...**

- **instanceof:** El operador instanceof devuelve **true** si el objeto especificado es del tipo de objeto especificado y **false** en el caso contrario.

```
var persona = {nombre: "Pepe", edad: 28};
var nombre = new String("Juan");
var apellido = "Tapia";

console.log(persona instanceof Object); // Registra en la consola el valor true
console.log(persona instanceof String); // Registra en la consola el valor false
console.log(nombre instanceof Object); // Registra en la consola el valor true
console.log(nombre instanceof String); // Registra en la consola el valor true
console.log(apellido instanceof Object); // Registra en la consola el valor false
console.log(apellido instanceof String); // Registra en la consola el valor false
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores**
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Operadores

Precedencia de los operadores

La precedencia de los operadores determina el orden en que se aplican al evaluar una expresión.

Puedes redefinir la precedencia de los operadores mediante el uso de **paréntesis**.

La siguiente tabla describe la precedencia de los operadores, de mayor a menor.

Tipo de operador	Operadores individuales
miembro	<code>.</code> <code>[]</code>
llamar / crear instancia	<code>()</code> <code>new</code>
negación / incremento	<code>!</code> <code>~</code> <code>-</code> <code>++</code> <code>--</code> <code>typeof</code> <code>void</code> <code>delete</code>
multiplicar / dividir	<code>*</code> <code>/</code> <code>%</code>
adición / sustracción	<code>+</code> <code>-</code>
desplazamiento bit a bit	<code><<</code> <code>>></code> <code>>>></code>
relacional	<code><</code> <code><=</code> <code>></code> <code>>=</code> <code>in</code> <code>instanceof</code>
igualdad	<code>==</code> <code>!=</code> <code>===</code> <code>!==</code>
AND bit a bit	<code>&</code>
XOR bit a bit	<code>^</code>
OR bit a bit	<code> </code>
AND lógico	<code>&&</code>
OR lógico	<code> </code>
condicional	<code>?:</code>
asignación	<code>=</code> <code>+=</code> <code>-=</code> <code>*=</code> <code>/=</code> <code>%=</code> <code><<=</code> <code>>>=</code> <code>>>>=</code> <code>&=</code> <code>^=</code> <code> =</code> <code>&&=</code> <code> =</code> <code>??=</code>
coma	<code>,</code>

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos**
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Conversión de tipos

JavaScript realiza conversión automática de tipos cuando se utilizan juntos diferentes tipos de datos en una operación.

Ejemplos de conversiones automáticas:

```
console.log('2' * 3);    // Escribe 6
console.log('2' + 3);    // Escribe 23
console.log('Hola' * 3); // Escribe NaN

console.log(null * 3);   // Escribe 0
console.log(true * 3);   // Escribe 3 (true se asocia al 1)
console.log(false * 3);  // Escribe 0 (true se asocia al 0)
console.log(undefined * 3); // Escribe NaN
```

En el segundo caso, al estar definido el operador + para strings, hace la conversión de 3 a string y luego concatena en lugar de convertir el 2 a número y luego hacer la suma.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos**
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Conversión de tipos

No siempre nos van a interesar las conversiones automáticas por lo que deberemos realizar nosotros las conversiones de tipo.

Conversión de entero a cadena

Por ejemplo usando el constructor `String()`.

```
let suma = 345.2;  
let total = String(suma);  
console.log(typeof total); // Escribe string
```

string
>> total
← "345.2"

También podemos usar la conversión automática para conseguirlo.

```
let suma = 345.2;  
let total = "" + suma;  
console.log(typeof total); // Escribe string
```

string
>> total
← "345.2"

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos**
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Conversión de tipos

Conversión de cadena a entero

Por ejemplo usando el constructor `Number()`.

```
let suma = "345.2";  
let total = Number(suma);  
console.log(typeof total); // Escribe number
```

number
>> total
← 345.2

Obviamente no todo valor se puede pasar a número.

```
let suma = "Hola";  
let total = Number(suma);  
console.log(typeof total);  
// Escribe number pero su valor es NaN
```

number
>> total
← NaN

Existen funciones específicas para convertir cadenas a enteros (**parseInt**) o a decimales (**parseFloat**) que permiten cambiar de base.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos**
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Conversión de tipos

¿Cómo saber si una expresión es numérica?

A veces comprobar si una expresión es NaN puede complicarse, cómo en el siguiente ejemplo:

```
>> "Hola"*4
← NaN
>> "Hola"*4==NaN
← false
```

Podemos ver que la expresión da un resultado no numérico pero si lo comparamos con NaN produce un resultado falso. Para ello tenemos la función **isNaN()** que devolverá verdadero si la expresión es numérica y falso en el caso contrario.

```
>> isNaN("Hola"*4)
← true
>> isNaN(NaN)
← true
>> NaN==NaN
← false
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

Sentencias Condicionales

La sentencia condicional **if** permite evaluar una condición y ejecutar un grupo de instrucciones si la condición resulta ser verdadera:

```
if (condición) {  
    instrucciones;  
}
```

Ejemplo: Al ser 2 menor que 6 escribirá por consola “2 es menor que 6”.

```
let num1 = 2;  
let num2 = 6;  
if(num1<num2) {  
    console.log(num1 + " es menor que " + num2);  
}
```


- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

La sentencia condicional **if** junto con **else** permite evaluar una condición y ejecutar un grupo de instrucciones si la condición resulta ser verdadera y otro grupo de instrucciones si la condición resulta ser falsa:

```
if (condición) {  
    instrucciones;  
}  
else {  
    instrucciones;  
}
```

Ejemplo:

```
let num1 = 10;  
let num2 = 6;  
if(num1<num2) {  
    console.log(num1 + " es menor que " + num2);  
}  
else {  
    document.write(num1 + " es mayor que " + num2);  
}
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

La sentencia condicional **if** se puede anidar tantas veces cómo nos interesa.

Ejemplo:

```
if (condición1) {  
    instrucciones;  
}  
else if (condición2) {  
    instrucciones;  
}  
else {  
    instrucciones;  
}
```

```
let num1 = 2;  
let num2 = 6;  
if(num1<num2) {  
    console.log(num1 + " es menor que " + num2);  
}  
else if(num1>num2) {  
    console.log(num1 + " es mayor que " + num2);  
}  
else {  
    console.log(num1 + " es igual que " + num2);  
}
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

Sentencias Condicionales

La sentencia condicional **switch** permite comprobar por una misma variable distintos valores. Según el valor de la condición del switch, se evalúa un grupo de sentencias. El último valor de la secuencia es "default" por si no encontramos un valor coincidente.

El "break" es opcional, pero si no se pone, se ejecutarán las sentencias que se encuentran en los siguientes valores del valor coincidente.

```
switch (condición) {  
    case valor1 : instrucciones; break;  
    case valor2 : instrucciones; break;  
    case valor3 : instrucciones; break;  
    default: instrucciones; break;  
}
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

Ejemplo:

```
let institut = prompt("Indica en que instituto te gustaría cursar tus estudios:");

switch (institut) {
  case "CIFP Francesc de Borja Moll": document.write("El mejor"); break;
  case "IEDIB": document.write("A distancia"); break;
  case "IES Na Camel·la": document.write("No tiene ciclos de informática"); break;
  default: document.write("El instituto indicado no existe"); break;
}
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

Sentencias Iterativas

La sentencia iterativa **while** permite ejecutar varias veces un bloque de código mientras se cumpla una condición. Se conoce como bucle.

```
while (condición) {  
    instrucciones;  
}
```

Ejemplo:

```
document.write("<h2>Tabla de multiplicar del 7</h2>");  
var cont = 1;  
while (cont <= 10) {  
    document.write("7 * " + cont + " = " + (7*cont) + "<br>");  
    cont = cont + 1;  
}
```

Tabla de multiplicar del 7

7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

Sentencias Iterativas

La sentencia iterativa **do while** es exactamente igual que el anterior con while pero con la diferencia que la condición se comprueba al final.

```
do {  
    instrucciones;  
} while (condición)
```

Ejemplo:

```
document.write("<h2>Tabla de multiplicar del 7</h2>");  
var cont = 1;  
do {  
    document.write("7 * " + cont + " = " + (7*cont) + "<br>");  
    cont = cont + 1;  
} while (cont <= 10)
```

Tabla de multiplicar del 7

7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

Sentencias Iterativas

La sentencia iterativa **for** repite el bloque de instrucciones hasta que una condición especificada se evalúa como falsa, para ello usa una variable que se inicializa a un valor y de va modificando a cada iteración.

```
for (inicialización; condición; modificación) {  
    instrucciones;  
}
```

Ejemplo:

```
document.write("<h2>Tabla de multiplicar del 7</h2>");  
for(let cont=1;cont<=10;cont++) {  
    document.write("7 * " + cont + " = " + (7*cont) + "<br>");  
}
```

Tabla de multiplicar del 7

7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control**
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras de control

Instrucciones break y continue

La instrucción break para la ejecución de un bucle y sale de él. La instrucción continue sirve para terminar la iteración actual y pasar a la siguiente.

Ejemplo:

```
document.write("<h2>Tabla de multiplicar del 2 </h2>");
for(let cont=1;cont<=10;cont++) {
    if(cont==4) {
        break;
    }
    document.write("2 * " + cont + " = " + (2*cont) + "<br>");
}

document.write("<h2>Tabla de multiplicar del 8 </h2>");
for(let cont=1;cont<=10;cont++) {
    if(cont==4) {
        document.write("<br>");
        continue;
    }
    document.write("8 * " + cont + " = " + (8*cont) + "<br>");
}
```

Tabla de multiplicar del 2

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6

Tabla de multiplicar del 8

8 * 1 = 8
8 * 2 = 16
8 * 3 = 24

8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
Strings
Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
Objeto window
Objeto document
Objeto navigator
Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Estructuras básicas de datos: Strings

El tipo **String** de JavaScript se utiliza para representar datos textuales. Es un conjunto de "elementos" de valores enteros sin signo de 16 bits (unidades de código UTF-16). Cada elemento de la cadena de caracteres ocupa una posición en la cadena.

El primer elemento está en el índice 0, el siguiente en el índice 1, y así sucesivamente. Puedes acceder a los caracteres usando `[]` junto al índice. La longitud de una cadena es el número de elementos que contiene.

```
let texto="Hola";
console.log(texto);
console.log(texto[0]);
console.log(texto[3]);
console.log(texto.length);
```

Hola
H
a
4
>>

Puedes crear cadenas simples utilizando comillas simples o dobles:

```
console.log("Hola");
console.log("Hola"[2]);
console.log("Hola".length);
```

Hola
1
4
>>

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings**
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras básicas de datos: Strings

Comparación de strings

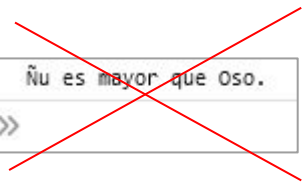
Se pueden comparar igual que los números. Se usa el orden en la tabla Unicode. Hay que tener en cuenta que las minúsculas son mayores que las mayúsculas en la tabla Unicode. También hay problemas con los caracteres especiales de cada lengua.

```
let texto1 = "Casa";
let texto2 = "casa";
if(texto1 == texto2) {
    console.log("Son iguales.");
}
else {
    console.log("No son iguales.");
}
```

No son iguales.
>>

```
let texto1 = "Oso";
let texto2 = "Ñu";
if(texto1 > texto2) {
    console.log("Oso es mayor que Ñu.");
}
else {
    console.log("Ñu es mayor que Oso.");
}
```

Ñu es mayor que Oso.
>>



Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Estructuras básicas de datos: Arrays

Un **array** es una lista ordenada de valores a los que te refieres con un **nombre** y un **índice**.

JavaScript no tiene un tipo de dato array explícito. Sin embargo, puedes utilizar el objeto Array predefinido y sus métodos para trabajar con arrays en tus aplicaciones.

Las siguientes declaraciones crean arrays de strings equivalentes:

```
let personas1 = new Array("Pedro","Juan","Ramiro");  
let personas2 = Array("Pedro","Juan","Ramiro");  
let personas3 = ["Pedro","Juan","Ramiro"];
```

```
>> personas1  
← ▶ Array(3) [ "Pedro", "Juan", "Ramiro" ]  
>> personas2  
← ▶ Array(3) [ "Pedro", "Juan", "Ramiro" ]  
>> personas3  
← ▶ Array(3) [ "Pedro", "Juan", "Ramiro" ]  
>> |
```

La sintaxis de corchetes [] se denomina array literal. Es más corto que otras formas de creación de arrays. La propiedad **length** del array indica el número de elementos del array.

```
console.log(personas1[1]);  
console.log(personas1.length);
```

```
Juan  
3  
>>
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays**
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Estructuras básicas de datos: Arrays

Puedes llenar un arreglo asignando valores a sus elementos. Por ejemplo: Creamos un array vacío y añadimos unos elementos.

```
let personas = [];  
personas[1] = "Pedro";  
personas[2] = "Juan";  
personas[3] = "Ramiro";
```

```
>> personas  
← ▶ Array(4) [ <1 empty slot>, "Pedro", "Juan", "Ramiro" ]  
>>
```

Hay que ir con cuidado con los arrays ya que son objetos y las asignaciones funcionan como referencias al objeto. Por eso en el siguiente ejemplo el resultado es 400 ya que tanto `datos` como `datos2` hacen referencia al mismo array.

```
let datos = [1,2,3,4,5];  
let datos2 = datos;  
  
datos[0] = 400;  
console.log(datos[0]);
```

```
400  
  
>>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Estructuras básicas de datos: Arrays

Podemos eliminar elementos de un array con **delete** cómo en el siguiente ejemplo. Cuando eliminamos un elemento de un array no se reindexa el array y ese elemento pasa a ser **undefined**.

```
let datos = [1,2,3,4,5];

delete datos[3];
for(let i=0;i<datos.length;i++) {
  console.log(datos[i]);
}
```

1
2
3
undefined
5
>>

Podemos tener arrays con diferentes tipos de datos.

```
let datos = [1,true,"Tom",4.15,52347861287346n];

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

1
true
Tom
4.15
52347861287346n
>> datos
← ▶ Array(5) [1, true, "Tom", 4.15, 52347861287346n]
>>

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Funciones

Las funciones son uno de los bloques de construcción fundamentales en JavaScript. Una función en JavaScript es similar a un conjunto de instrucciones que realiza una tarea o calcula un valor, esta puede tomar alguna entrada y devolver una salida donde hay alguna relación obvia entre la entrada y la salida. Para usar una función, debes definirla en algún lugar del ámbito desde el que deseas llamarla.

Definir una función no la ejecuta. Definirla simplemente nombra la función y especifica qué hacer cuando se llama a la función. Llamar a la función en realidad lleva a cabo las acciones especificadas con los parámetros indicados.

Una definición de función consta de la palabra clave **function**, seguida de:

- El nombre de la función.
- Una lista de parámetros de la función, entre paréntesis y separados por comas (puede ser una lista vacía).
- Las declaraciones de JavaScript que definen la función, encerradas entre llaves, { }.
- Un valor de retorno indicado con la palabra **return** (es opcional).

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones**
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Funciones

Sintaxis:

```
function nombreFuncion (parámetros) {  
    instrucciones;  
    return valor;  
}
```

Ejemplo: Función que calcula el cuadrado de un número.

```
function cuadrado(numero) {  
    return numero * numero;  
}  
  
console.log(cuadrado(5)); // Escribe 25
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones**
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Funciones

Ejemplo: Función que muestra por consola el área de un rectángulo.

```
function areaRectangulo(lado1, lado2) {  
    console.log(lado1 * lado2);  
}  
  
areaRectangulo(3,4); // Escribe 12
```

Ejemplo: Función que muestra la fecha por consola.

```
function muestraFecha() {  
    let fecha = new Date();  
    console.log(fecha);  
}  
  
muestraFecha();
```


- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones**
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Funciones

Funciones predefinidas de JavaScript

Javascript contiene una gran cantidad de funciones en sus librerías. Muchas de las librerías se implementan a través de objetos, por ejemplo el objeto **Math** y el objeto **String**.

Por ejemplo **Math**, se utiliza para realizar cálculos matemáticos. Para utilizarlos se opera a través de la clase, en lugar del objeto, para trabajar con la clase **Math** no se utiliza la instrucción **new**, sino que se utiliza el nombre de la clase para acceder a sus propiedades y métodos.

También hay funciones de JavaScript que no están asociadas a ninguna objeto.

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones**
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Funciones

Ejemplos de funciones predefinidas de JavaScript

isNaN() La función isNaN() determina si un valor es NaN o no. Devuelve true o false.

<pre>>> isNaN(12) ← false >> isNaN("34") ← false</pre>	<pre>>> isNaN(2/0) ← false >> isNaN(NaN) ← true</pre>	<pre>>> isNaN(0/0) ← true >> isNaN(Math.sqrt(-1)) ← true</pre>
--	---	--

parseFloat() La función parseFloat() procesa un argumento de cadena y devuelve un número de punto flotante.

<pre>>> parseFloat(35) ← 35 >> parseFloat("12.5") ← 12.5</pre>	<pre>>> parseFloat(5.678) ← 5.678 >> parseFloat("Hola") ← NaN</pre>
--	---

parseInt() La función parseInt() procesa un argumento de cadena y devuelve un número entero de la base especificada (la base en los sistemas numéricos matemáticos).

```
>> parseInt(35)
← 35
>> parseInt("12")
← 12
>> parseInt("Hola")
← NaN
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones**
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Funciones

Ejemplos de funciones predefinidas para Arrays

push(*elemento*) La función push() añade un elemento al final del array.

```
let datos = [10,20,30];
datos.push(5);
```

```
>> datos
< ▶ Array(4) [ 10, 20, 30, 5 ]
>>
```

pop() La función pop() retira el último elemento del array y lo devuelve.

```
let datos = [10,20,30];
let valor = datos.pop();
```

```
>> datos
< ▶ Array [ 10, 20 ]
>> valor
< 30
```

shift(*elemento*) La función shift() retira un elemento al principio de un array y lo devuelve.

```
let datos = [10,20,30];
let valor = datos.shift();
```

```
>> datos
< ▶ Array [ 20, 30 ]
>> valor
< 10
```

unshift() La función unshift() añade un elemento del principio del array.

```
let datos = [10,20,30];
datos.unshift(5);
```

```
>> datos
< ▶ Array(4) [ 5, 10, 20, 30 ]
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores**
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Manejo de errores

Los errores en Javascript son objetos que se muestran cada vez que se produce un error de programación. Estos objetos contienen mucha información sobre el tipo de error.

Javascript también permite crear errores personalizados para proporcionarle información adicional.

El tipo de estructura que nos sirve para controlar y comprobar el flujo de un programa ante comportamientos inesperados es **try ... catch**. Nos va a permitir manejar los errores.

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
- Manejo de errores**
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Manejo de errores

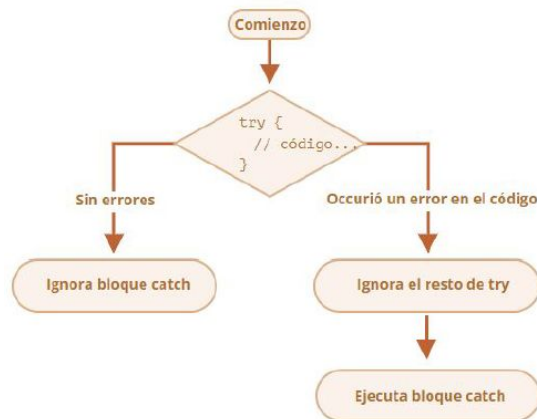
Los errores en Javascript son objetos que se muestran cada vez que se produce un error de programación. Estos objetos contienen mucha información sobre el tipo de error.

Javascript también permite crear errores personalizados para proporcionarle información adicional.

El tipo de estructura que nos sirve para controlar y comprobar el flujo de un programa ante comportamientos inesperados es **try ... catch**. Nos va a permitir manejar los errores.

Tiene dos bloques principales:

```
try {  
    // instrucciones;  
} catch (error) {  
    // instrucciones;  
}
```



- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores**
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Manejo de errores

Primero, se ejecuta el código en **try**.

Si no ha habido errores, se ignora **catch(error)**, es decir, la ejecución llega al final de **try** y continúa, omitiendo **catch**.

Si se produce un error, la ejecución de **try** se detiene y el control va al principio **catch(error)**. La variable o parámetro "error" contendrá un objeto de error con detalles sobre ese error.

Conclusión, un error dentro del bloque **try {...}**, no mata el script, tenemos la oportunidad de reaccionar al error en el **catch**.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Manejo de errores

Ejemplos:

- Sin errores

```
try {  
  console.log('Hola');  
  console.log('Adios');  
} catch (error) {  
  console.log('Se produjo un error: ' + error.message);  
}
```

Hola
Adios
>>

- Con errores

```
try {  
  console.log('Hola');  
  hola  
  console.log('Adios');  
} catch (error) {  
  console.log('Se produjo un error: ' + error.message);  
}
```

Hola
Se produjo un error: hola is not defined
>>

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Manejo de errores

try ... catch sólo funciona para errores en tiempo de ejecución. Por ejemplo, No funcionará si el código es sintácticamente incorrecto, por ejemplo, si hay llaves sin cerrar. Entonces, **try ... catch** sólo puede manejar errores que ocurren en un código válido.

Si utilizamos **let** para declarar una variable dentro de **try**, esta variable no estará disponible para ser utilizada dentro de **catch**, esto es por lo que ya sabemos que son declaraciones de ámbito local.

Cuando se produce un error, Javascript genera un objeto que contiene los detalles sobre ese error. El objeto tiene dos propiedades principales:

- **name**: Es el nombre del error. Por ejemplo, para una variable indefinida es `ReferenceError`.
- **message**: Es el mensaje de texto sobre los detalles del error.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores**
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Manejo de errores

También podemos tener **try ... catch ... finally**. En este caso el bloque **finally** se ejecutará tanto si el código se ejecuta correctamente como si se produce un error.

```
try {  
    console.log('Hola');  
    console.log('Adios');  
} catch (error) {  
    console.log('Se produjo un error: ' + error.message);  
}  
finally {  
    console.log("Finalmente.");  
}
```

Hola
Adios
Finalmente.
>>

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores**
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Manejo de errores

Se pueden lanzar excepciones mediante la sentencia **throw**. En este caso usamos directamente error ya que lo que lanzamos en el ejemplo no es un objeto error.

```
let nota = 3;
try {
  if(nota < 5) {
    throw "Hay que recuperar.";
  }
  else {
    console.log("Enhorabuena has aprobado.");
  }
} catch (error) {
  console.log('Se produjo un error: ' + error);
}
finally {
  console.log("Terminó la evaluación.");
}
```

Se produjo un error: Hay que recuperar.

Terminó la evaluación.

>>

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos**
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Objetos

Los **objetos** en JavaScript, como en tantos otros lenguajes de programación, se pueden comparar con objetos de la vida real. El concepto de Objetos en JavaScript se puede entender con objetos tangibles de la vida real. En JavaScript, un objeto es un entidad independiente con **propiedades y métodos**.

Los objetos son dinámicos, esto significa que sus propiedades no tienen por qué ser definidas en el momento en que se crea el objeto, podemos añadir nuevas propiedades al objeto en tiempo de ejecución

Un objeto de JavaScript tiene propiedades asociadas a él. Una propiedad de un objeto se puede explicar como una variable adjunta al objeto. Las propiedades de un objeto definen las características del objeto. Accedes a las propiedades de un objeto con una simple notación de puntos:

```
var myCar = new Object();  
myCar.make = "Ford";  
myCar.model = "Mustang";  
myCar.year = 1969;
```

```
var myCar = {  
  make: "Ford",  
  model: "Mustang",  
  year: 1969,  
};
```

También puedes acceder o establecer las propiedades de los objetos en JavaScript mediante la notación de corchetes []

```
myCar["make"] = "Ford";  
myCar["model"] = "Mustang";  
myCar["year"] = 1969;
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos**
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Objetos

Un método es una función asociada a un objeto, o, simplemente, un método es una propiedad de un objeto que es una función. Los métodos se definen normalmente como una función, con excepción de que tienen que ser asignados como la propiedad de un objeto.

```
function imprimeCoche(){
    console.log("Esto es un coche");
}

var myCar = new Object();
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
myCar.imprime = imprimeCoche;
```

```
var myCar = {
    make: "Ford",
    model: "Mustang",
    year: 1969,
    imprime: function (){
        console.log("Esto es un coche.");
    }
};
```

JavaScript tiene una palabra clave especial, **this**, que puedes usar dentro de un método para referir al objeto actual.

```
var myCar = {
    make: "Ford",
    model: "Mustang",
    year: 1969,
    imprime: function (){
        console.log("Este coche es un: " + this.make + " " + this.model + " del " + this.year);
    }
};
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos**
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Objetos

Para crear objetos puedes usar una función **constructora**.

```
function Car(make, model, year) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
}  
  
var kenscar = new Car("Nissan", "300ZX", 1992);  
var vpgscar = new Car("Mazda", "Miata", 1990);
```

A partir de ES2015 también existe el concepto de **clase**.

```
class Car {  
    constructor(make, model, year) {  
        this.make = make;  
        this.model = model;  
        this.year = year;  
    }  
}  
  
var kenscar = new Car("Nissan", "300ZX", 1992);  
var vpgscar = new Car("Mazda", "Miata", 1990);
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos**
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Objetos

JSON - Javascript Object Notation

JSON es un formato para almacenar y transportar datos.

JSON es texto, y el texto puede transportarse a cualquier lugar y leerse mediante cualquier lenguaje de programación.

Los objetos JavaScript se pueden convertir a JSON y JSON se puede volver a convertir en objetos JavaScript.

Ej: Un objeto persona con tres propiedades y su versión en JSON.

```
>> var myCar = {  
    make: "Ford",  
    model: "Mustang",  
    year: 1969  
};
```

```
'{"make":"Ford","model":"Mustang","year":1969}'
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
 LocalStorage
 Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
 Eventos
 Validación de formularios
 Expresiones regulares

Objetos

Tenemos dos funciones que nos van a permitir obtener un objeto de un JSON y obtener un JSON de un objeto:

- **JSON.stringify(objeto)** Devuelve una representación JSON de un objeto.

```
var myCar = new Object();  
myCar.make = "Ford";  
myCar.model = "Mustang";  
myCar.year = 1969;  
  
console.log(JSON.stringify(myCar));
```

```
>> var myCar = new Object();  
    myCar.make = "Ford";  
    myCar.model = "Mustang";  
    myCar.year = 1969;  
  
    console.log(JSON.stringify(myCar));  
  
{"make":"Ford","model":"Mustang","year":1969}
```

- **JSON.parse(JSON)** Devuelve un objeto de una representación JSON.

```
let miJSON = '{"make":"Ford","model":"Mustang","year":1969}';  
var myCar = JSON.parse(miJSON);  
console.log(myCar);
```

```
>> let miJSON = '{"make":"Ford","model":"Mustang","year":1969}';  
    var myCar = JSON.parse(miJSON);  
    console.log(myCar);  
  
▶ Object { make: "Ford", model: "Mustang", year: 1969 }
```

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos**
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Objetos

JSON - Javascript Object Notation

JSON es un formato para almacenar y transportar datos.

JSON es texto, y el texto puede transportarse a cualquier lugar y leerse mediante cualquier lenguaje de programación.

Los objetos JavaScript se pueden convertir a JSON y JSON se puede volver a convertir en objetos JavaScript.

Ej: Un objeto persona con tres propiedades y su versión en JSON.

```
>> var myCar = {  
    make: "Ford",  
    model: "Mustang",  
    year: 1969  
};
```

```
"{"make":"Ford","model":"Mustang","year":1969}"
```


Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Objetos

El objeto Date.

Date es un objeto. Nos permite trabajar tanto con fechas como con horas. Se puede utilizar para almacenar horas de creación o modificación, medir el tiempo, etc. Zonas horarias: En JavaScript, trabajaremos o bien con la hora local (la que tendrá nuestro dispositivo), o bien con la hora universal coordinada (UTC). De forma predeterminada, casi siempre, en JavaScript se devuelve la fecha/hora local. Para obtener UTC, debe especificarse. Si se escribe sin parámetros devuelve la fecha y la hora actual del sistema.

Podemos hacer:

- `new Date()`: Crea un objeto con hora y fecha actual. Mostrará una fecha como una cadena de texto.
- `new Date(milisegundos)`: Crea un objeto de fecha de tiempo cero (desde las 00:00:00 UTC del 1 de enero de 1970 más los milisegundos especificados).
- `new Date(cadenaData)`: Crea un objeto de fecha a partir de un string de fecha.
- `new Date(año, mes, día, horas, minutos, segundos, milisegundos)`: Crea un objeto de fecha con una fecha y hora específicas.

```
>> new Date();  
← ▶ Date Sat Jan 20 2024 16:19:18 GMT+0100 (hora estándar de Europa central)  
  
>> new Date(155000000);  
← ▶ Date Fri Jan 02 1970 20:03:20 GMT+0100 (hora estándar de Europa central)  
  
>> new Date("12/05/1983");  
← ▶ Date Mon Dec 05 1983 00:00:00 GMT+0100 (hora estándar de Europa central)  
  
>> new Date(2024, 1, 8, 15, 30, 27, 783);  
← ▶ Date Thu Feb 08 2024 15:30:27 GMT+0100 (hora estándar de Europa central)
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

LocalStorage

El objeto **localStorage** nos permite almacenar información de la forma clave/valor.

El objeto `localStorage` almacena datos sin fecha de caducidad.

Los datos no se eliminan al cerrar el navegador y están disponibles para futuras sesiones.

Existe también el objeto **sessionStorage** que almacena datos para una sesión. En este caso los datos se eliminan cuando se cierra la ventana del navegador.

Tiene los siguientes métodos y propiedades:

- `localStorage.length`
- `localStorage.setItem(key, value);`
- `let value = localStorage.getItem(key);`
- `localStorage.removeItem(key);`
- `localStorage.clear();`

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

LocalStorage

- Ejemplo:

```
document.write(localStorage.length + "<br>")

let miarray = [1,2,3,"hola", true];

localStorage.setItem("tom", "Hola Mundo");
localStorage.setItem(25, 3.141592);
localStorage.setItem("un array", JSON.stringify(miarray));

let nom = localStorage.getItem("tom");
let pi = localStorage.getItem(25);
let valores = JSON.parse(localStorage.getItem("un array"));

document.write(nom + "<br>");
document.write(pi + "<br>");
document.write(valores + "<br>");
document.write(valores[3] + "<br>");
document.write(valores[4] + "<br>");

localStorage.removeItem("tom");
let nom2 = localStorage.getItem("tom");
document.write(nom2 + "<br>");

document.write(localStorage.length + "<br>");
localStorage.clear();
document.write(localStorage.length + "<br>");
```

1
Hola Mundo
3.141592
1,2,3,hola,true
hola
true
null
3
0

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window**
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios
- Expresiones regulares

Objeto Window

El objeto window representa la ventana que contiene un documento DOM; la propiedad document apunta al DOM document cargado en esa **ventana**.

En un navegador con pestañas, cada pestaña contiene su propio objeto window. Esto significa que el objeto window no se comparte entre diferentes pestañas de la misma ventana del navegador.

Es el objeto de la jerarquía más alto. Todos los demás objetos posibles de una página web están incluidos en él.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Objeto Window

Algunas propiedades son:

- `window.devicePixelRatio` Devuelve la relación entre los píxeles físicos y los píxeles independientes del dispositivo en la pantalla actual.
- `window.fullScreen` Esta propiedad indica si la ventana está desplegada en pantalla completa o no.
- `window.location` Obtiene/fija la ubicación, o URL actual, del objeto de la ventana.
- `window.navigator` Devuelve una referencia al objeto del navegador.
- `window.screen` Devuelve una referencia al objeto de pantalla asociado con la ventana.
- `window.innerHeight` Obtiene la altura del área de contenido del navegador incluyendo la barra de scroll horizontal si existe.
- `window.innerWidth` Obtiene el ancho del área de contenido de la ventana del navegador incluyendo la barra de scroll vertical si existe.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Objeto Window

Algunos métodos son:

- `window.alert()` Muestra una pequeña ventana de alerta.
- `window.confirm()` Muestra una ventana de confirmación con respuestas: confirmar y cancelar.
- `window.prompt()` Devuelve el texto introducido por el usuario.
- `window.atob()` Decodifica de base64 a binario.
- `window.btoa()` Codifica de binario a base64.
- `window.back()` Carga la página anterior del historial.
- `window.forward()` Carga la siguiente dirección del historial.
- `window.home()` Carga la página de inicio.
- `window.setInterval()` Establece un intervalo.
- `window.clearInterval()` Cancela un intervalo .
- `window.setTimeout()` Establece un Timeout.
- `window.clearTimeout()` Cancela un Timeout

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Objeto window

Algunos métodos son:

- `window.open()` Abre una nueva ventana.
- `window.close()` Cierra la ventana actual.
- `window.maximize()` Maximiza la ventana.
- `window.minimize()` Minimiza la ventana.
- `window.print()` Abre la ventana de impresión del documento actual.
- `window.find()` Busca un string dado en una ventana.

- `window.moveBy()` Mueve la ventana actual por una cantidad especificada.
- `window.moveTo()` Mueve la ventana actual a una posición especificada.
- `window.resizeBy()` Redimensiona la ventana actual una cantidad dada.
- `window.resizeTo()` Redimensiona la ventana actual.
- `window.scrollBy()` Desliza el documento en la ventana una cierta cantidad.
- `window.scrollTo()` Desliza el documento en la ventana a una posición concreta.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Objeto document

Este objeto es el que contiene toda la página actual que se está visualizando en el momento. Depende del objeto window. Nos permitirá añadir de forma dinámica contenido en nuestra página.

Algunas propiedades son:

- `alinkColor` Es el color de los enlaces activos.
- `anchors` Array con los enlaces del documento.
- `bgColor` Se refiere al color de fondo del documento.
- `cookie` A través de ella podemos crear, leer, modificar y eliminar cookies.
- `domain` Nombre del dominio del servidor que ha servido el documento.
- `fgColor` Propiedad que contiene el color del texto.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Objeto document

Algunas propiedades son:

- `forms` Array con todos los formularios del documento. Estos a su vez tienen sus elementos que ya veremos en su día.
- `images` Array que contiene las imágenes del documento.
- `lastModified` Contiene la fecha de la última modificación del documento
- `linkColor` Almacena el color de los enlaces.
- `links` Array con los enlaces externos.
- `location` Cadena que contiene la URL del documento.
- `title` Cadena que contiene el título del documento actual.
- `vlinkColor` Color de los enlaces que ya han sido visitados.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Objeto navigator

El objeto navigator nos permite conocer datos como el navegador que se utiliza, la su versión, el sistema operativo, etc.

Algunas propiedades son:

- `appName` Devuelve el código del nombre del navegador.
- `appName` Devuelve el nombre del navegador.
- `appVersion` Devuelve la versión del navegador.
- `cookieEnabled` Determina si las cookies están o no habilitadas.
- `platform` Devuelve la plataforma sobre la que se está ejecutando el navegador.
- `userAgent` Devuelve una información completa sobre el agente de usuario (normalmente es el navegador).

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
- Eventos
- Validación de formularios
- Expresiones regulares

Acceso al DOM

El modelo de objetos del documento (DOM), es una interfaz de programación para los documentos HTML y XML.

Es una representación completamente orientada al objeto de la página web y puede ser modificado con un lenguaje de script como Javascript.

Facilita una **representación estructurada del documento** y define cómo los programas pueden acceder, a fin de modificar, tanto su estructura como el estilo y contenido.

Da una representación del documento como un grupo de nodos y objetos estructurados que poseen propiedades y métodos. Esencialmente, conecta las páginas web en scripts o lenguajes de programación.

Fue diseñado para ser independiente de cualquier lenguaje de programación particular.

Todas las propiedades, métodos y eventos disponibles para la manipulación y la creación de páginas web está organizada dentro de objetos.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
- Eventos
- Validación de formularios
- Expresiones regulares

Acceso al DOM

El DOM no es un lenguaje de programación, pero sin él, el lenguaje Javascript no tiene ningún modelo o noción de las páginas web HTML ni de las páginas XML.

Cada elemento (por ejemplo, todo el documento, el título, las tablas, los títulos de las tablas, etc.), es parte del modelo de objeto del documento, de esta forma, se puede acceder y manipularlos (mediante el DOM) y con un lenguaje de escritura como es Javascript.

Al principio, Javascript y DOM estaban herméticamente enlazados, pero después se desarrollaron como entidades separadas.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
- Eventos
- Validación de formularios
- Expresiones regulares

Acceso al DOM

¿Cómo se accede a DOM?

No se debe hacer nada especial para empezar a utilizar el DOM.

Los distintos navegadores tienen directrices DOM diferentes, y estas directrices tienen varios grados de conformidad al actual estándar DOM, pero todos los navegadores web emplean el modelo de objeto de documento para hacer accesibles las páginas web a el script.

En Javascript, la forma de acceder al DOM, es a través de sus nodos, vamos a ver los tipos de nodos que podemos encontrar.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Tipo de nodos del DOM

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas HTML, se pueden manejar con sólo 4 o 5 tipos de nodos:

- **documento:** El nodo document es el nodo raíz, a partir del cual derivan el resto de nodos.
- **elemento:** Son los nodos definidos por etiquetas html. Por ejemplo, una etiqueta "div" genera un nodo. Si dentro del "div" tenemos 3 etiquetas "p", estas etiquetas definen nodos hijos de la etiqueta "div".
- **texto:** El texto dentro de un nodo elemento se considera un nuevo nodo hijo de tipo texto.
- **attr:** Los atributos de las etiquetas definen nodos, aunque con Javascript no los veremos como nodos, sino que se consideran información asociada al nodo de tipo elemento.
- **comment:** Los comentarios y otros elementos como son las declaraciones "doctype" en la cabecera de los documentos HTML generan también nodos.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

¿Cómo se accede a los nodos?

DOM proporciona dos métodos alternativos para acceder a un nodo específico:

- a través del nodo padre (la raíz)
- o de forma directa.

Es más rápido acceder de forma directa, por tanto veremos las funciones para acceder directamente.

Es importante saber que para acceder, modificar o eliminar algún nodo, sólo se puede hacer cuando el árbol DOM se ha construido por completo, es decir, **cuando la página se ha cargado por completo.**

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Métodos de acceso directo:

- **document.getElementById():** Es el método o función más empleada cuando se hacen aplicaciones web dinámicas. Permite acceder directamente a un nodo para leer y/o modificar sus propiedades. Devuelve un elemento HTML que tiene un atributo id que coincide con lo indicado en el parámetro de la función.

Sintaxis:

```
let variable=document.getElementById("valor_id") ;
```


- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
- Eventos
- Validación de formularios
- Expresiones regulares

Acceso al DOM

Ejemplo: Cambiar el color de un título h2

```
<html>

<head>
  <title>Ejemplo getElementById</title>
</head>

<body>
  <h2 id="clik">Texto que ha de cambiar de color</h2>
  <button onclick="canviarcolor('orange');">Naranja</button>
  <button onclick="canviarcolor('green');">Verde</button>
  <script>
    function canviarcolor(pcolor) {
      var x = document.getElementById('clik'); // Pone ddentro de la variable
                                              // "x" un objeto que referencia
                                              // al elemento HTML con id = "clik".
      x.style.color = pcolor; // Ponemos este elemento en el color
                             // que indica el parámetro "pcolor".
                             // Accedemos a la propiedad
                             // style.color del elemento.
    }
  </script>
</body>

</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **document.getElementsByTagName():** Obtiene, en forma de array, todos los elementos de la página HTML donde la etiqueta sea igual que el parámetro que se le ocurre a la función.

Sintaxis:

```
let                                     variable                                     =  
document.getElementsByTagName("nombre_etiqueta_elemento");
```

El valor que devuelve es un objeto de tipo "nodelist" (similar al array) con todos los nodos que tienen la etiqueta igual al parámetro que ocurre el método. Para obtener los valores de este objeto, se realizará igual que con los arrays:

Ejemplo: Obtener los títulos h1 de la página

```
var título = document.getElementsByTagName("h1");
```

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Acceso al DOM

Ejemplo: Cambiar el color del fondo de los h1

```
<html>

<head>
  <meta charset="UTF-8">
  <title>h1 en rojo</title>
</head>

<body>
  <h1>Primer título</h1>
  <h1>Segundo título</h1>
  <script>
    var titol = document.getElementsByTagName("h1");
    for (var i = 0; i < titol.length; i++) {
      titol[i].style.backgroundColor = 'red'
    }
  </script>
</body>

</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **document.getElementsByClassName():** Devuelve un array con todos los elementos con el nombre de la clase especificada.

Sintaxis:

```
var x = document.getElementsByClassName("nombre_clase");
```

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Acceso al DOM

Ejemplo: Cambiar el estilo de un título <h3> y de un párrafo con la misma clase.

```
<html>

<head>
  <meta charset="UTF-8">
  <title>Exemple</title>
</head>

<body>
  <h3 class="canvi">Título h3 de la página con la clase llamada "canvi"</h3>
  <p> Párrafo de la página sin clase.</p>
  <p class="canvi">Párrafo de la página con la clase llamada "canvi"</p>
  <button type="button" onclick="classes()"> clica aquí</button>
  <script>
    function classes() {
      var classe = document.getElementsByClassName('canvi');
      for (i = 0; i < classe.length; i++) {
        classe[i].style.color = "orange";
        classe[i].style.fontSize = "25px";
        classe[i].style.backgroundColor = "yellow";
        classe[i].style.borderStyle = "solid";
      }
    }
  </script>
</body>

</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Métodos para crear nodos:

- **document.createElement():** Crea un nodo tipo de elemento. El parámetro será por tanto una etiqueta HTML.

Sintaxis:

```
var x = document.createElement("etiqueta_element");
```

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Acceso al DOM

Ejemplo: Añadimos un elemento "p" al body.

```
<html>

<body>

  <h1>Haz click en el botón y se creará un botón</h1>
  <button onclick="crearboton()">Haz click</button>
  <script>
    function crearboton() {
      let boton = document.createElement("p"); // Creamos el elemento p.
      boton.innerHTML = "Texto del párrafo"; // InnerHTML es donde se guardará el text del elemento.

      // Añadimos los nodos p en el primer nodo body que encontramos
      document.getElementsByTagName("body")[0].appendChild(boton);
    }
  </script>
</body>

</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **document.createTextNode():** Crea un nodo de tipo texto.

Sintaxis:

```
let t = document.createTextNode("texto");
```

Ejemplo: Añade elemento a una lista

```
<html>
<body>
  <h1>Haz click i se añadirán elementos a una lista</h1>
  <button onclick="añadirtexto()">Haz click</button>
  <ol id="llista">
    <li>Un elemento</li>
  </ol>
  <script>
    function añadirtexto() {
      var nodeElement = document.createElement("li");
      var nodeText = document.createTextNode("Text d'un altre element");
      nodeElement.appendChild(nodeText);

      document.getElementById("llista").appendChild(nodeElement);
    }
  </script>
</body>
</html>
```


Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Métodos para la gestión de nodos

Los dos métodos anteriores, nos sirven para crear elementos y nodos de texto, pero si no los añadimos al árbol, éstos elementos no representarán bien el contenido de la página web. Para poder añadir estos elementos tenemos los siguientes métodos:

- **node.appendChild():** Nos permite añadir un nuevo nodo elemento hijo al árbol. Como podremos ver en la sintaxis este método se vincula directamente al nodo que queremos añadir el nuevo nodo.

Sintaxis:

```
elementopadre.appendChild( nuevoelemento );
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **node.insertBefore():** Nos permite añadir un nuevo elemento antes de otro.

Sintaxis: `elementpare.insertBefore(nodeElementNou, nodeGerma);`

Ejemplo: Insertamos un párrafo como primero, es decir, antes del primer párrafo actual.

```
<html>
  <body>
    <div id="parrafo">
      <p>Primer p</p>
      <p>Segundo p</p>
    </div>
    <button onclick=añadir()> Click </button>
    <script>
      function añadir() {
        var nodeElement = document.createElement('p'); // Creamos y guardamos en una variable el nuevo párrafo
        nodeElement.appendChild(document.createTextNode('Nuevo párrafo')); // Añadimos el texto
        var nodePare = document.getElementById('parrafo'); // Donde insertamos
        var nodeGerma = nodePare.firstChild; // Guardam el primer párrafo
        nodePare.insertBefore(nodeElement, nodeGerma); // Insertamos el nuevo elemento
      }
    </script>
  </body>
</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Reemplazar/eliminar/clonar:

- **node.replaceChild():** Permite reemplazar un nodo elemento existente.

Sintaxis: `nodopadre.replaceChild(nuevonodo, nodosustituito)`

Ejemplo: Reemplazamos un párrafo por otro.

```
<html>
<body>
  <section id="s1">
    <p id="p1">Un párrafo</p>
    <p id="p2">Otro párrafo</p>
  </section>
  <script>
    var nodoNuevo = document.createElement("p");
    var nodoTexto = document.createTextNode("Nou paràgraf");
    nodoNuevo.appendChild(nodoTexto);
    var nodoPadre = document.getElementById("s1");
    var nodoHermano = document.getElementById("p1");
    nodoPadre.replaceChild(nodoNuevo, nodoHermano);
  </script>
</body>
</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **node.removeChild():** Permite borrar un nodo elemento.
- **node.cloneNode(true):** Permite realizar una copia exacta de un nodo elemento. Con el parámetro "true", lo que se hace, es decirle al método que queremos clonar el nodo y todos sus hijos.

Ejemplo: Borrar elementos de una lista

```
<!DOCTYPE html>
<html>
<body>
  <ul id="llista">
    <li>Ana</li>
    <li>Pep</li>
    <li>Leire</li>
  </ul>
  <p>Haz click en el botón para eliminar los elements de la lista</p>
  <button onclick="eliminar()">Haz click</button>
  <script>
    function eliminar() {
      var list = document.getElementById("llista");
      list.removeChild(list.childNodes[0]);
    }
  </script>
</body>
</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Algunas propiedades generales de los nodos

Independientemente del tipo de nodo, existen una serie de propiedades generales que comparten todos:

- **node.childNodes:** Nos devuelve un array con todos los nodos hijo de un elemento. Es de sólo lectura. Los nodos resultantes son tanto elementos como nodos de tipo texto.

Sintaxis: `let variable = document.element.childNodes;`

Ejemplo: Obtener los nodos del elemento body

```
var c = document.body.childNodes;  
//obtenemos un array "c" con los hijos de "body"
```

- **node.children:** Devuelve los nodos hijos de tipo elemento de un nodo padre.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **node.firstChild:** Devuelve el primer nodo hijo directo de un nodo, y null si no tiene hijos. Es de sólo lectura. En algunos navegadores, el contenido de "firstChild", también incluye como hijo posibles espacios en blanco o saltos de línea que el elemento puede contener, y esto ocasiona problemas.

Ejemplo: Obtener el primer hijo de un elemento span

```
<p id="ex">  
  <span> esto es un span </span>  
</p>  
<script>  
  var x = document.getElementById('ex');  
  var hijo = x.firstChild;  
</script>
```

- **lastChild:** Nos devuelve el último nodo hijo directo del elemento (funciona como el firstChild).

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **nextSibling:** Nos devuelve al hermano siguiente al nodo que se especifica.

Sintaxis: `let node = document.getElementById('valor_id').nextSibling`
- **previousSibling:** Nos devuelve el hermano anterior al nodo que se especifica.
Devuelve null si el nodo es el primer hijo.
- **hasChildNodes:** Indica si el nodo tiene o no hijos. Devuelve un booleano.
- **parentNode:** Devuelve el nodo padre de un elemento.
- **nodeName:** Es el nombre del nodo, que se define dependiendo del tipo de nodo (cualquier tipo de nodo). Devuelve una cadena literal.

Ejemplo: `alert(document.body.nodeName); // BODY`
- **tagName:** Lo mismo que nodeName, pero sólo para nodos de tipo elemento.

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **node.innerHTML:** Nos permite recuperar el contenido texto de un elemento o asignar nuevo contenido texto. Si asignamos nuevo texto, destruimos lo que tiene. Almacena también la información de HTML.

Sintaxis: `let contenido = document.getElementById("valor_id").innerHTML;`

Ejemplo: Cambiar el texto de un párrafo.

```
<!DOCTYPE html>
<html>

<body>
  <p id="inner" onclick="canviar()">Haz click encima de este párrafo para cambiar el texto.</p>
  <script>
    function canviar() {
      document.getElementById("inner").innerHTML = "Nuevo texto";
    }
  </script>
</body>

</html>
```


Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Trabajando con atributos

Existen cuatro métodos para modificar atributos de elementos:

- **hasAttribute():** Comprueba si un elemento tiene un atributo concreto. Devuelve un booleano.

Sintaxis: `elemento.hasAttribute('nombre_atributo');`

- **getAttribute():** Muestra el valor de un atributo especificado o null.

Sintaxis: `elemento.getAttribute('nombre_atributo');`

- **setAttribute():** Agrega o actualiza el valor de un atributo especificado.

Sintaxis: `elemento.setAttribute("nombre", "valor");`

- **removeAttribute():** Elimina un atributo de un elemento.

Sintaxis: `elemento.removeAttribute("valor_atributo");`

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Acceso al DOM

Ejemplo: Comprobar que un elemento botón tiene un atributo determinado.

```
<html>

<body>

  <p>Haz click para saber si el elemento botón tiene el atributo onclick.</p>
  <button id="boto" onclick="has()">Haz click</button>
  <p id="demo"></p>
  <script>
    function has() {
      var x = document.getElementById("boto").hasAttribute("onclick");
      if (x == true) {
        document.write("el botó té l'atribut onclick")
      }
      else {
        document.write("El botó no té l'atribut onclick")
      }
    }
  </script>
</body>

</html>
```

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Acceso al DOM

Ejemplo: Mostrar valor de un atributo.

```
<!DOCTYPE html>
<html>

<head>
  <style>
    .rojo {
      color: red;
      background-color: rgb(0, 255, 255)
    }
  </style>
</head>

<body>
  <h1 class="rojo">Provamos getAttribute</h1>
  <h1> Un método del DOM</h1>
  <button onclick="comprovar()">Haz click</button>
  <p id="añadirparrafo"></p>
  <p>Añadirémos usando un párrafo, el valor del atributo class que tiene el primer título.</p>
  <script>
    function comprobar() {
      var x = document.getElementsByTagName("h1")[0].getAttribute("class");
      document.getElementById("añadirparrafo").innerHTML = x;
    }
  </script>
</body>

</html>
```

- Introducción
 - ¿Dónde ubicamos el código?
 - Comentarios
 - Visualización y recogida de datos
 - Variables
 - Valores
 - Expresiones
 - Palabras reservadas de JavaScript
 - Tipos de datos
 - Operadores
 - Conversión de tipos
 - Estructuras de control
 - Estructuras básicas de datos
 - Strings
 - Arrays
 - Funciones
 - Manejo de errores
 - Objetos
 - LocalStorage
 - Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
 - Eventos
 - Validación de formularios
 - Expresiones regulares

Acceso al DOM

Ejemplo: Agregar un atributo "class"

```
<!DOCTYPE html>
<html>

<head>
  <style>
    .rojo {
      color: ■ red;
    }
  </style>
</head>

<body>
  <h1>Aplicaremos una classe a este título</h1>
  <button onclick="aplicarclasse()">Haz click</button>
  <script>
    function aplicarclasse() {
      document.getElementsByTagName("h1")[0].setAttribute("class", "rojo");
    }
  </script>
</body>

</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

Trabajando con clases

- **className:** Es una propiedad. Obtiene o establece una clase.

Sintaxis: `element.className;`

Ejemplo: Añadir una clase a h1

```
<html>
  <head>
    <style>
      |   .estilo_nuevo { color: ■ green; }
    </style>
  </head>
  <body>
    <h1 id="ex">Añadimos una clase</h1>
    <script>
      |   document.getElementById("ex").className = "estilo_nuevo";
    </script>
  </body>
</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **classList.add():** Es un método que agrega, uno o más valores de clase.
Sintaxis: `elemento.classList.add("valor", ...)` ;

Ejemplo: añadir clases con classList.add()

```
<html>
  <head>
    <style>
      .estilo { width: 300px; height: 50px; background-color: red; }
      .estilo2 { color: rgb(13, 29, 251); }
    </style>
  </head>
  <body>
    <button onclick="añadirclase()">Haz click</button>
    <main id="nav">
      <p>Texto del main</p>
    </main>
    <script>
      function añadirclase() {
        document.getElementById("nav").classList.add("estilo", "estilo2");
      }
    </script>
  </body>
</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Acceso al DOM

- **classList.toggle():** Es un método que puede tener uno o dos argumentos, si hay uno, alterna el valor de la clase, es decir, si la clase existe lo elimina y devuelve false, y si no existe la añade y devuelve true. Si se indican ambos argumentos, si el segundo argumento se evalúa como true, se añade la clase y si se evalúa como false, elimina la clase.

Sintaxis: `elemento.classList.toggle("clase", "valor_true_o_false");`

- **classList.contains():** Este método comprueba si el valor de clase existe.

Sintaxis: `elemento.classList.contains('active');`

- **classList.replace():** Método que sustituye un valor de clase existente por un nuevo.

Sintaxis: `elemento.classList.replace('old', 'new');`

- **classList.remove():** Método para eliminar un valor de clase.

Sintaxis: `elemento.classList.remove('active');`

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM**
- Eventos
- Validación de formularios
- Expresiones regulares

Acceso al DOM

Algunos atributos son accesibles de forma directa, se tratan como propiedades: href, style, className, src, etc.

Ejemplos:

```
let enlace = document.getElementById("en1");
alert(enlace.href);
// mostrará la URL del enlace con id="en1")
```

```
let imagen = document.getElementById("img");
alert.log(imagen.style.margin);
// mostrará el valor del margen de la imagen con id="img"
```

```
let p = document.getElementById("par");
alert(p.className);
// mostrará el nombre de la clase del párrafo con id="par"
```

```
let imagen= document.getElementById("img");
alert(img.src);
// mostrará el archivo de imagen que el elemento imagen con id="img"
```


Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Eventos

Los eventos son cosas que suceden en el sistema que estás programando. El sistema se encarga de producir una señal de cierto tipo cuando un evento ocurre, y proporciona un mecanismo para que una acción se lleve a cabo (ejecutar código) de forma automática cuando el evento ocurra.

Los eventos se lanzan dentro de la ventana del navegador y usualmente están asociados a un elemento en específico dentro de dicha ventana. Esto puede ser un solo elemento, un grupo de elementos, el documento HTML cargado en la pestaña actual, o la ventana del navegador en su totalidad. Existen distintos tipos de eventos que pueden ocurrir.

Por ejemplo:

- El usuario selecciona, hace clic o pasa el ratón por encima de cierto elemento.
- El usuario presiona una tecla del teclado.
- El usuario redimensiona o cierra la ventana del navegador.
- Una página web terminó de cargarse.
- Un formulario fue enviado.
- Un vídeo se reproduce, se pausa o termina.
- Ocurrió un error.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos**
- Validación de formularios
- Expresiones regulares

Eventos

Para reaccionar a un evento, puedes asociarle un **manejador de eventos**. Esto es un bloque de código (normalmente una función de JavaScript) que se ejecuta cuando el evento ocurre. Cuando uno de estos bloques de código se configura para ejecutarse en respuesta de un evento, decimos que estamos registrando un manejador de eventos.

Ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <button id="boton">Cambiar el color de fondo</button>
    <script>
      let boton = document.getElementById("boton");

      boton.addEventListener("click", () => {
        document.body.style.backgroundColor = "red";
      });
    </script>
  </body>
</html>
```

Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Eventos

- **addEventListener()** Registra un evento a un objeto en específico. Permite registrar más de una acción para un evento.

Sintaxis: `target.addEventListener(event, listener[, useCapture]);`

`event` - El nombre del evento. No se debe poner “on” delante. [Aquí](#) tienes un listado eventos.

`listener` - La función que se va a ejecutar.

`useCapture` - Opcional. Por defecto es false. Indica si se ejecutará en [fase](#) de bubbling o de capture.

- **removeEventListener()** Elimina un manejador de evento.

Sintaxis: `target.removeEventListener(tipo, listener[, useCapture])`

`useCapture` - Se puede registrar un mismo manejador para la fase de bubbling y de capture, por lo que se debe eliminar indicando la fase. Por defecto asume que es false.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Aceso al DOM
- Eventos**
 - Validación de formularios
 - Expresiones regulares

Eventos

Las funciones que manejan eventos pueden recibir un parámetro de tipo event con información sobre el evento producido.

Ejemplo: pulsado de una tecla.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Ejemplo</title>
  </head>
  <body>
    <h1>Ejemplo</h1>
    <script>
      function tecla(event) {

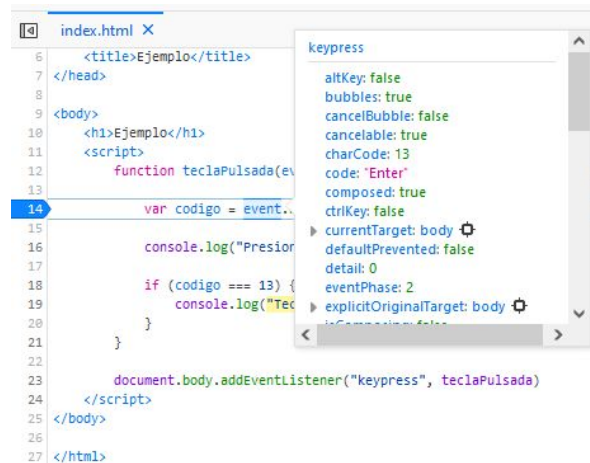
        var codigo = event.keyCode;

        console.log("Presionada: " + codigo);

        if (codigo === 13) {
          console.log("Tecla ENTER");
        }

        document.body.addEventListener("keypress", tecla);
      }
    </script>
  </body>
</html>
```

Presionada: 103
Presionada: 13
Tecla ENTER
>>



Introducción
¿Dónde ubicamos el código?
Comentarios
Visualización y recogida de datos
Variables
Valores
Expresiones
Palabras reservadas de JavaScript
Tipos de datos
Operadores
Conversión de tipos
Estructuras de control
Estructuras básicas de datos
 Strings
 Arrays
Funciones
Manejo de errores
Objetos
LocalStorage
Introducción al DOM
 Objeto window
 Objeto document
 Objeto navigator
 Acceso al DOM
Eventos
Validación de formularios
Expresiones regulares

Validación de formularios

Antes de enviar datos al servidor, es importante asegurarse de que se completan todos los controles de formulario requeridos, y en el formato correcto. Esto se denomina **validación de formulario en el lado del cliente** y ayuda a garantizar que los datos que se envían coinciden con los requisitos establecidos en los diversos controles de formulario

Hay dos tipos diferentes de validación por parte del cliente que encontrarás en la web:

- La validación de formularios incorporada utiliza características de validación de formularios HTML5, que hemos visto en muchos lugares a lo largo de este módulo. Esta validación generalmente no requiere mucho JavaScript. La validación de formularios incorporada tiene un mejor rendimiento que JavaScript, pero no es tan personalizable como la validación con JavaScript. Ejemplo: required, min, max, minlength, maxlength, pattern.
- La validación con JavaScript se codifica utilizando JavaScript. Esta validación es completamente personalizable.

- Introducción
- ¿Dónde ubicamos el código?
- Comentarios
- Visualización y recogida de datos
- Variables
- Valores
- Expresiones
- Palabras reservadas de JavaScript
- Tipos de datos
- Operadores
- Conversión de tipos
- Estructuras de control
- Estructuras básicas de datos
 - Strings
 - Arrays
- Funciones
- Manejo de errores
- Objetos
- LocalStorage
- Introducción al DOM
 - Objeto window
 - Objeto document
 - Objeto navigator
 - Acceso al DOM
- Eventos
- Validación de formularios**
- Expresiones regulares

Validación de formularios

Podemos validar un formulario usando el evento **onsubmit** del formulario. Si el evento **onsubmit** devuelve el valor **true**, el formulario se enviará normalmente, si el evento **onsubmit** devuelve **false**, el formulario no se envía y muestra una alerta con el error o la indicación de dónde la comprobación no se superó. El punto clave consiste en comprobar todos los campos y cada uno de los elementos del formulario.

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Ejemplo</h1>
    <form action="http://www.google.es" method="get" onsubmit="return validar()">
      <label for="nombre">Nombre</label>
      <input type="text" id="nombre" name="nombre">
      <input type="submit" value="Enviar datos">
    </form>
    <script>
      function validar() {
        const nombre = document.getElementById("nombre");
        return (nombre.value.length > 0);
      }
    </script>
  </body>
</html>
```