Unió Europea
Fons Social Europeu
L'FSE inverteix en el teu futur

GENERALITAT VALENCIANA
Conselleria d'Educació, Universitats i Ocupació

1º DAW

Module

Markup languages and information management systems

Unió Europea
Fons Social Europeu
L'FSE inverteix en el teu futur

GENERALITAT VALENCIANA
Conselleria d'Educació, Universitats i Ocupació

WU 5.

Manipulation of web documents: JavaScript

1. Lenguajes de script del lado cliente

1.1 Introducción

Los lenguajes de script del lado cliente se ejecutan en el navegador web (cliente).

Se incluyen en el código HTML o como archivos independientes.

Permiten ejecutar acciones dinámicas según la interacción del usuario.

1.2 Lenguajes de script del lado cliente

Los lenguajes más utilizados son JavaScript, VBScript y TypeScript.

Se ejecutan en el navegador y complementan los lenguajes del lado servidor.

AJAX combina tecnologías para actualizar contenido sin recargar la página.

1.2 Lenguajes de script del lado cliente

En el caso del lenguaje de script

- no existe proceso de compilación,
- no hay archivo binario,
- no hay ejecutable
- y el archivo fuente es directamente ejecutado (Interpretado) en la máquina cliente.

1.3 JavaScript. Características generales

JavaScript es un lenguaje de script orientado a objetos, con tipado débil y sin necesidad de compilación.

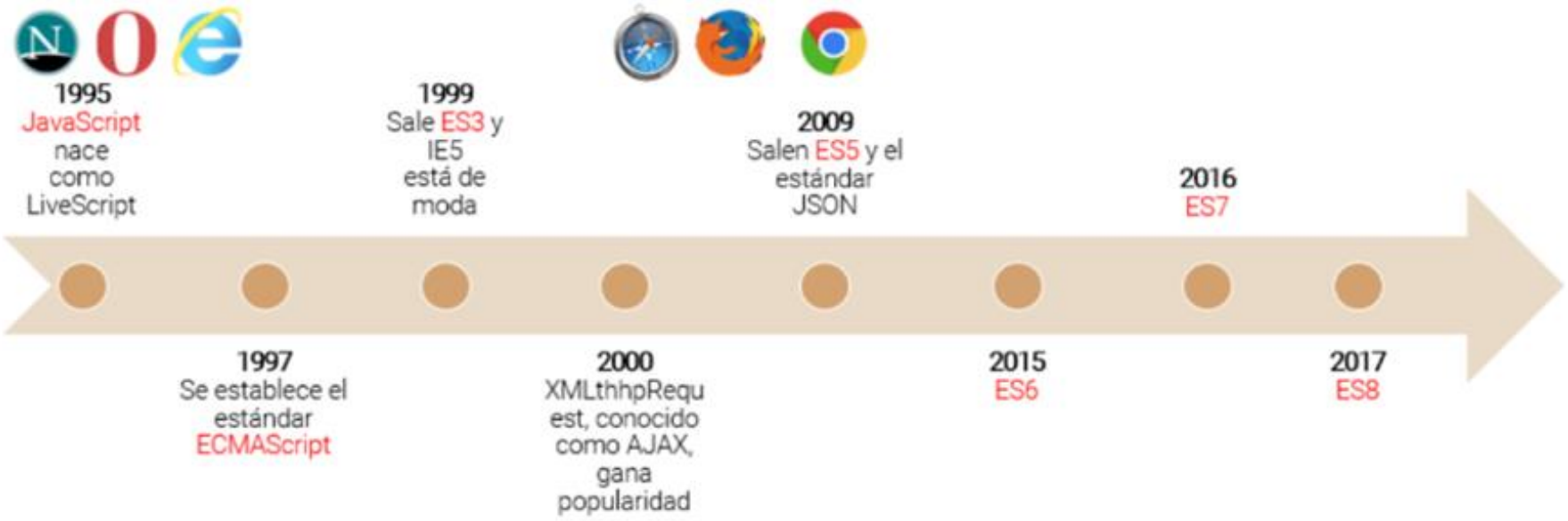
Se usa para agregar interactividad y dinamismo a las páginas web.

1.3 JavaScript. Características generales

JavaScript y ECMAScript están estrechamente relacionados, ya que **ECMAScript** es el estándar sobre el cual se basa JavaScript.

Aunque JavaScript incluye las características definidas por ECMAScript, también agrega funciones específicas, como APIs del navegador para interactuar con el DOM, eventos, y otras funcionalidades no especificadas en ECMAScript.

1.3 JavaScript. Características generales



1.4 Estructura de un programa en JavaScript

Un programa JavaScript incluye:

- variables,
- funciones,
- instrucciones y
- comentarios.

La sintaxis básica permite estructurar el código para la interacción con el navegador.

1.5 Integración de JavaScript en HTML5

JavaScript se puede integrar de manera in-line, embebido o en archivos externos.

La última opción es la más recomendada por HTML5 por su modularidad.

1.5 Integración de JavaScript en HTML5

JavaScript in-line

```
<!DOCTYPE html>
<html lang=”es”>
<head>
<meta charset=”utf-8”>
<title>Mi primera prueba con JS</title>
</head>
<body>
<section>
<span onclick=”alert('¡Hola Mundo!');”>Haz clic aquí</span>
<section>
</body>
</html>
```

1.5 Integración de JavaScript en HTML5

JavaScript embebido.

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Mi primera prueba con JS</title>
<script>
function alerta() {
alert('Hola Mundo!');
}
</script>
</head>
<body>
<section>
<span onclick="alerta();">Haz clic aquí</span>
</section>
</body>
</html>
```

1.5 Integración de JavaScript en HTML5

JavaScript externo.

```
< ! DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Mi primera prueba con JS</title>
<script src="./codigo.js"></script>
</head>
<body>
<section>
<span id="alerta">Haz clic aquí</span>
</section>
</body>
</html>
```

codigo.js

```
window.onload = function() {
    document.getElementByld('alerta').onclick = function ()
    {
    alert('Hola Mundol');
    }
}
```

1.6 Otros ambientes de ejecución

Además de los navegadores web, JavaScript puede ejecutarse en:

- Asociado a eventos, dentro del documento HTML.
- Consola del navegador web.
- Consola de Node.js, para acceder a recursos del sistema operativo.
- Herramientas en línea como JSFiddle permiten probar código rápidamente.

1.6 Otros ambientes de ejecución

Consola del navegador web:

Accedemos a la consola mediante Shift+Ctrl+i y dirigiéndonos a la pestaña Consola.

```
console.info("Cuadrados de los 10 primeros números naturales");

for (var n = 1; n <= 10; n++) {
 console.log("%d^2 = %d", n, n * n);
 }
```

1.7 Control de ejecución de scripts

Se usan los atributos defer y async para controlar el momento de ejecución.

Con defer el script espera a que el documento se cargue completamente.

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

async ejecuta el script en cuanto está disponible al mismo tiempo.

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

1.7 Control de ejecución de scripts

<script src="archivo.js" />

Si no se incluye ninguno de los dos, el script se ejecuta de forma inmediata.

Al encontrarse el código JavaScript está en un recurso externo, el navegador web debe esperar a que el contenido de este archivo se haya descargado completamente antes poder continuar desplegando el documento web.

2. Elementos del lenguaje JavaScript

Los elementos fundamentales del lenguaje JavaScript son los siguientes:

- Datos
- Variables
- Arrays
- Funciones
- Objetos

2.1 Datos

JavaScript trabaja con datos como números, cadenas de texto, valores booleanos y punteros.

Cada tipo tiene métodos y usos específicos para la manipulación de la información.

2.2 Variables y operadores

Se usan let, const y var para declarar variables.

Ejemplo con let:

```
<script>
for (let i=0;i< 2; ++i) {
console.log('Primero ', i);
   for (let i=0;i<2; ++i) {
         console.log('Segundo ', i);
   }
}
</script>
```

2.2 Variables y operadores

Ejemplo con var:

```
<script>
for (var i=0;i< 2; ++i) {
console.log('Primero ', i);
   for (var i=0;i<2; ++i) {
         console.log('Segundo ', i);
   }
}
<script>
```

2.2 Variables y operadores

Los operadores permiten realizar cálculos, comparaciones y operaciones lógicas en el código.

2.2 Variables y operadores

Tipo	Función
Comparación	Compara los valores de 2 operandos y el resultado puede ser true o false. == != === !== > >= < <=
Aritméticos	Resuelve la operación aritmética entre los operandos. El operador + suma números y concatena cadenas. + - * / % ++ - - +valor -valor
Asignación	Asigna un valor a la variable. = += -= *= /= %= <<= >= >>= >>>= &= |= ^ = []
Concatenar	Unir cadenas. Operador + `var cadena3 = "Hola" + cadena2;`
Boolean	Realiza una operación de tipo lógico entre uno/dos operandos booleanos. && || !
Bit a Bit	Sobre datos binarios realiza una operación aritmética o de desplazamiento. & | ^ ~ << >> >>>

2.2 Variables y operadores

Ejemplo:

Haz un programa, y ejecútalo en la consola del navegador web, de manera que muestre el resultado de trabajar con todos los operadores aritméticos.

```
let a= 7;

let b= 9;

console.log('La suma de %d y %d es = %d', a, b, a + b);
```

2.3 Arrays

Un array almacena múltiples valores en una sola variable.

Los índices comienzan desde 0 y permiten manipular los elementos individualmente.

2.3 Arrays

Haz un programa, y ejecútalo en la consola del navegador web, de manera que muestre el resultado de trabajar con todos los operadores aritméticos.

```
var matriculas = [];

matriculas[0] = “1234BBB”;

matriculas[1] = “5678CCC”;

alert(matriculas[1] ) ;
```

2.4 Sentencias de control

Incluyen estructuras condicionales:

- if
- switch

y bucles:

- for
- while
- do … while

Controlan el flujo del programa según condiciones o repeticiones.

2.4 Sentencias de control

Ejemplo: switch

```
var dia = 3;
switch(dia) {
case ( 3 ):
   alert("Entre semana");
   break;
case ( 6 ):
   alert("Fin de semana");
   break;
default: alert("Valor no válido");
}
```

2.4 Sentencias de control

Ejercicio: Completa el switch anterior para que ante cualquier valor nos sepa responder si se trata de un día de entre semana o de fin de semana.

Ejercicio: Completa el programa anterior para que nos pida el día de la semana.

```
var dia = parseInt(prompt("Introduce un día:"));
```

2.5 Funciones

Una función es un bloque de código reutilizable.

Puede ser:

- definida por el usuario o
- ser una función predefinida del entorno JavaScript como puede ser alert() y confirm().

2.5 Funciones

Ejemplo: Función sumar (dos valores)

```
function sumar(A, B) {
return A + B;
}
var num1 = parseFloat(prompt(“Primer nº:"));
var num2 = parseFloat(prompt(“Segundo nº:"));
var resultado = sumar(num1, num2);
console.log("El resultado de la suma es: " +
resultado);
```

2.6 Tratamiento de fecha y hora

JavaScript usa el objeto Date para manejar fechas y horas.

Incluye métodos para obtener, establecer y manipular fechas y horas.

2.6 Tratamiento de fecha y hora

Método	Descripción
`getDate()`	Devuelve el día del mes (entre el 1 y el 31).
`getDay()`	Devuelve el día de la semana (entre el 0 y el 6). Domingo es el 0.
`getMonth()`	Devuelve el mes (entre el 0 y el 11).
`getFullYear()`	Devuelve el año (en formato de 4 dígitos).
`getHours()`	Devuelve la hora (entre el 0 y el 23).
`getMinutes()`	Devuelve los minutos (desde 0 a 59).
`getSeconds()`	Devuelve los segundos (desde 0 a 59).
`getTime()`	Devuelve el número de milisegundos desde el 1 de enero de 1970.
`getTimezoneOffset()`	Devuelve la diferencia de horario en minutos entre la hora local y GMT (meridiano de Greenwich).
`getUTCHours()`	Devuelve la hora UTC (tiempo universal coordinado).

2.6 Tratamiento de fecha y hora

Ejemplo: getDate

```
<script>
var fechaActual = new Date();
var diaDelMes = fechaActual.getDate();

console.log("El día del mes actual es: " +
diaDelMes);
</script>
```

3. Utilización de objetos en JavaScript: DOM

En el primer apartado se ha indicado que JavaScript es un lenguaje orientado a objetos.

A partir de ahora se va a trabajar con los elementos de la página web los cuales pueden ser tratados como objetos por JavaScript.

3.1 Introducción al DOM

El DOM (Document Object Model) es una interfaz estándar del W3C que organiza los documentos HTML y XML como un árbol jerárquico de nodos.

Cada nodo representa una parte del documento (etiquetas, texto, atributos, etc.) y permite su manipulación dinámica a través de lenguajes como JavaScript.

3.1 Introducción al DOM

El DOM es esencial para conectar el contenido estático de HTML con la programación dinámica, permitiendo a los desarrolladores crear experiencias interactivas y personalizadas.

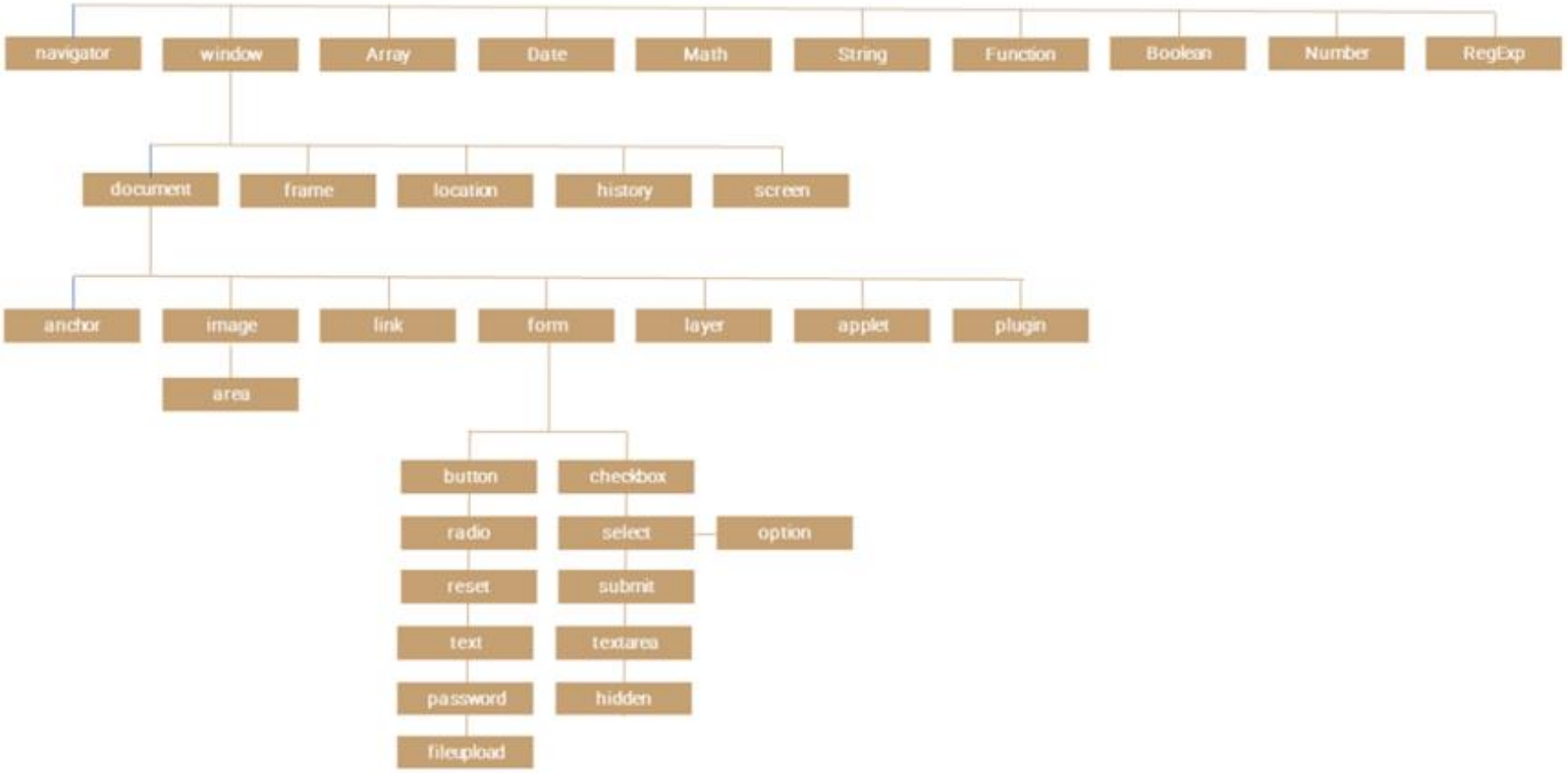
Se utiliza para modificar contenido, estilos y atributos en tiempo real, ajustándose a las interacciones del usuario.

3.1 Introducción al DOM

La jerarquía del DOM permite navegar fácilmente por su estructura mediante relaciones como nodos padre, nodos hijos y nodos hermanos.

Esto facilita la selección y manipulación de elementos específicos.

3.1 Introducción al DOM



3.1 Introducción al DOM

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>Ejemplo: Información sobre el navegador</title>
<script>
function navegador() {
document.write("Propiedades del objeto navigator(): "+ "<br><br><br>");
document.write("appCodeName: " +navigator.appCodeName+ "<br><br>" );
document.write("appName: " +navigator.appName+ "<br><br>" );
document.write("appVersion: " +navigator.appVersion+ "<br><br>" );
document.write("cookieEnabled: " +navigator.cookieEnabled+ "<br><br>" );
document.write("platform: " +navigator.platform+ "<br><br>" );
document.write("userAgent: " +navigator.userAgent+ "<br><br>" );
}
</script>
</head>
<body onload="navegador();"></body>
</html>
```

3.1 Introducción al DOM

Además, el DOM soporta eventos, lo que permite reaccionar a acciones como clics, movimientos del ratón y teclas presionadas.

Esto lo convierte en una herramienta indispensable para el desarrollo de aplicaciones modernas.

3.2 Acceso a la página web: DOM

El acceso al DOM es clave para interactuar con elementos específicos de una página web.

JavaScript proporciona métodos estándar como:

- getElementById,
- querySelector y
- getElementsByClassName

para localizar y seleccionar nodos específicos.

3.2 Acceso a la página web: DOM

Estos métodos permiten modificar atributos, contenido y estilos de los elementos seleccionados, facilitando la creación de interfaces dinámicas.

Por ejemplo, se pueden cambiar los colores, agregar clases CSS o actualizar textos.

3.2 Acceso a la página web: DOM

El DOM también permite:

- la creación de nuevos nodos dinámicos mediante createElement y
- la eliminación de nodos existentes con removeChild.

Esto es útil para añadir o quitar contenido según las necesidades del usuario.

3.2 Acceso a la página web: DOM

Propiedades como

- `parentNode`,
- `childNodes` y
- `nextSibling`

ayudan a navegar por la estructura jerárquica del DOM, permitiendo moverse eficientemente entre los nodos relacionados.

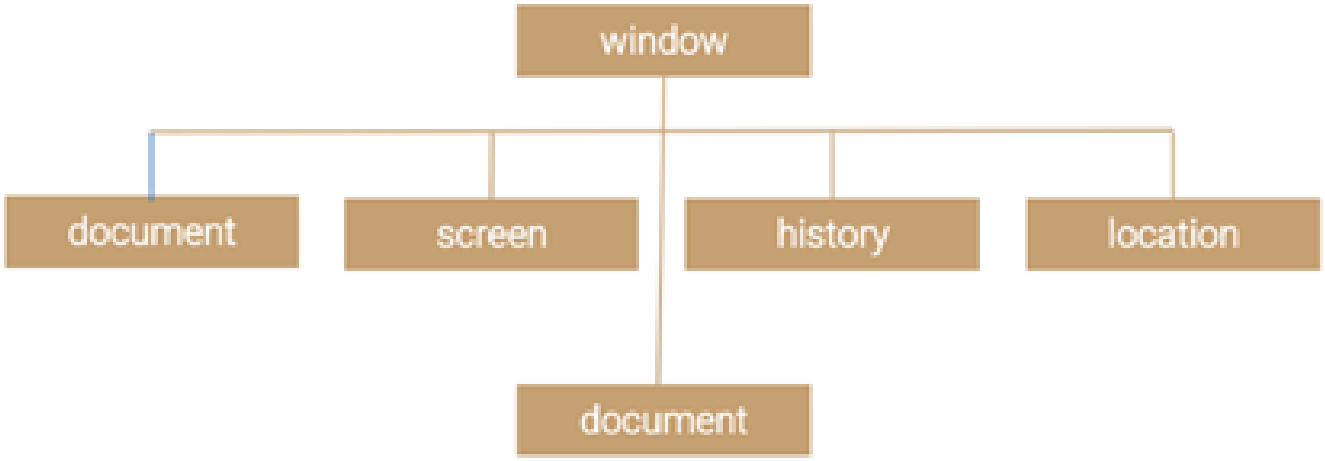
3.2 Acceso a la página web: DOM

El acceso al DOM es esencial para desarrollar aplicaciones web modernas, ya que permite actualizar el contenido de forma dinámica sin necesidad de recargar la página.

3.3 Objeto window

El objeto window es el contexto global en el navegador y contiene todas las funciones y propiedades relacionadas con el entorno de la ventana.

Es el punto de entrada para acceder al DOM y otros objetos importantes del navegador.



3.3 Objeto window

Algunas de las propiedades más utilizadas incluyen:

- `innerWidth` y `innerHeight`: Proporcionan las dimensiones de la ventana del navegador.
- `location`: Proporciona información sobre la URL actual.
- `navigator`: Brinda detalles sobre el navegador y el sistema operativo del usuario.

3.3 Objeto window

Algunos métodos del objeto window:

- prompt, muestra una ventana de diálogo para introducir datos.
- alert, para mostrar mensajes
- confirm, muestra una ventana emergente con un mensaje, un botón de aceptar y un botón de cancelar.
- open, para abrir nuevas ventanas y
- setTimeout, para configurar temporizadores.

Estos métodos son fundamentales para crear interacciones dinámicas con el usuario.

3.3 Objeto window

El objeto window también gestiona eventos globales como:

- `resize` (cambio de tamaño de la ventana) y
- `beforeUnload` (antes de cerrar la página).

Esto lo hace indispensable para controlar el comportamiento de las aplicaciones web.

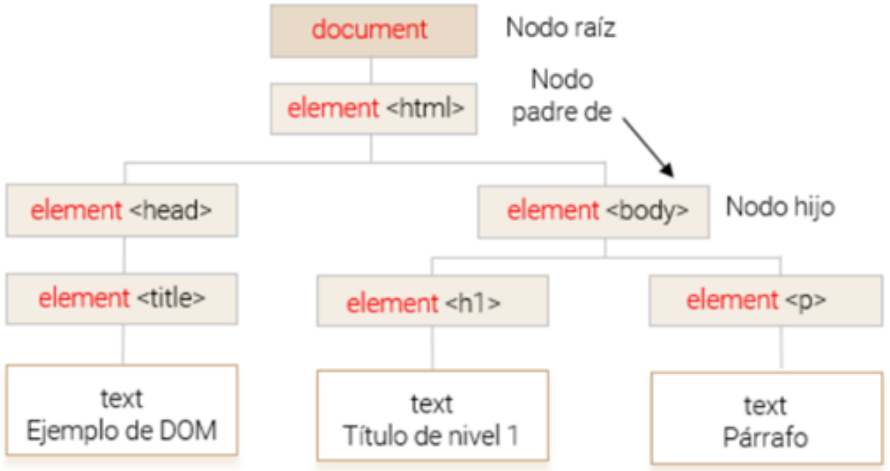
3.4 Objeto document

El objeto document representa el contenido completo de la página web cargada en el navegador y es el nodo raíz del DOM.

Es la puerta de entrada para interactuar con los elementos de la página web.

3.4 Objeto document

```
<html lang="es">
<head>
<meta charset=utf-8" />
<title>Ejemplo de DOM</title>
</head>
<body>
<h1>Título de nivel l</h1>
<p>Párrafo</p>
</body>
<html>
```



3.4 Objeto document

El DOM define 12 tipos de nodos, pero las páginas HTML normalmente manejan los siguientes:

- document: es la raíz de la estructura en árbol.
- element: nodo hijo del raíz, identifica la etiqueta <html> y a su vez es padre de otros nodos element que contienen etiquetas HTML.
- attr: identifica los atributos de las etiquetas que contienen "atributo=valor".
- text: identifica el texto encerrado por una etiqueta.
- comment: identifica los comentarios incluidos en la página HTML.

3.4 Objeto document

Proporciona métodos esenciales como:

- getElementById, para seleccionar elementos específicos,
- createElement, para generar nuevos nodos y
- appendChild, para añadir elementos al DOM.

3.4 Objeto document

Ejemplo 1: Se obtiene una referencia al nodo div

```
<div id="titulo">
    <h1>Ejemplo: Uso de getElementById()</h1>
</div>
<script>
    var nododiv = document.getElementById("titulo");
</script>
```

Ejemplo 2: Se añade 'nuevoElemento' como último hijo del elemento body

```
var nuevoElemento = document.createElement("h2");
var nuevoNodoTexto = document.createTextNode("Nuevo nodo texto");
nuevoElemento.appendChild(nuevoNodoTexto);
 document.body.appendChild(nuevoElemento);
```

3.4 Objeto document

Acceder a los nodos	Añadir nodos al DOM	Eliminar nodos
• getElementById() • getElementByTagName() • getElementByClassName() • getElementByName() • parentNode • childNodes • firstChild	Crear nodos • createElement() • createTextNode() Añadir nodos • appendChild() • insertBefore()	• removeChild() • replaceChild()
Modificar contenido de nodos	**Modificar atributo style**	**Modificar atributo class**

• Contenido elementos
  • innerHTML
  • textContent
• Atributo elementos
  • setAttribute()
  • getAttribute()
  • removeAttribute()
  • hasAttribute()
	• Acceder al objeto style
	• Dar propiedades style
	• Crear un objeto style
		• className
		• classList
		  • add()
		  • remove()
		  • toggle()
		  • contains()
		  • items()

3.4 Objeto document

El objeto document también permite manejar eventos como:

- DOMContentLoaded

que asegura que el DOM esté completamente cargado antes de ejecutar scripts.

Esto mejora la experiencia del usuario y optimiza el rendimiento.

3.4 Objeto document

Es posible modificar dinámicamente la estructura de la página web mediante la adición o eliminación de elementos, así como ajustar sus atributos y estilos.

3.4 Objeto document

El objeto document es esencial para cualquier desarrollo web dinámico, ya que permite actualizar el contenido de forma eficiente y responder a las interacciones del usuario en tiempo real.

4. Eventos en JavaScript

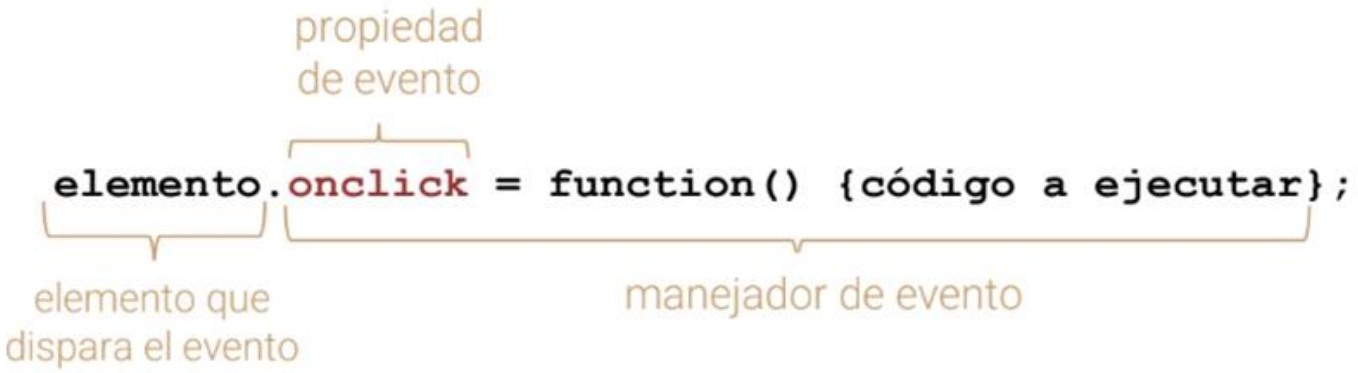
Los eventos en JavaScript son interacciones o sucesos que ocurren en una página web del navegador web, como clics, movimientos del ratón, o pulsaciones de teclas.

4.1 Introducción

Los eventos permiten que las páginas sean interactivas al reaccionar a las acciones del usuario en tiempo real.

JavaScript captura estos eventos utilizando manejadores de eventos, que son funciones asociadas a elementos específicos para responder a las acciones del usuario.

4.1 Introducción



Ejemplo: Definimos el evento mediante el atributo de evento

```
<input type="button" onclick="alert('¡Holal');" value="saluda"/>
```

Ejemplo: Definimos el evento mediante el atributo de evento

```
<input type="button" onclick="saludar()" value="saluda"/>
<script>
function saludar() {
        alert("¡Hola!");
};
</script>
```

4.1 Introducción

Ejemplo: Definimos el evento mediante una propiedad de evento

```
<script>
   document.getElementById("Boton").onclick = function Color() {
            document.formulario.caja_texto.style.backgroundColor='#f88067’;
   }
</script>
```

Ejemplo: Definimos el evento mediante un manejador de evento

```
<script>
function alerta() {
     alert('¡Prueba superada!');
}
window.onload = function () {
        document.querySelector("button").addEventListener('click',alerta,false);
}
</script>
```

4.1 Introducción

Ejemplos comunes de eventos incluyen:

- onclick,
- onmouseover,
- onkeydown, y
- onsubmit

que se utilizan para interactuar con botones, formularios y otros elementos web.

4.1 Introducción

La gestión de eventos es fundamental para crear interfaces dinámicas y mejorar la experiencia del usuario en las aplicaciones web.

4.2 Gestión de eventos

La gestión de eventos en JavaScript permite asociar funciones a eventos específicos en elementos de la página web.

4.2 Gestión de eventos

El método más utilizado para gestionar eventos es `addEventListener`, que permite asignar múltiples manejadores a un mismo elemento, de manera flexible.

4.2 Gestión de eventos

Los manejadores de eventos también pueden configurarse directamente en el propio HTML mediante atributos como onclick, aunque este enfoque es menos recomendado en proyectos modernos.

4.2 Gestión de eventos

Los eventos capturados pueden utilizarse para ejecutar funciones específicas, como mostrar un mensaje, cambiar estilos o validar formularios.

4.2 Gestión de eventos

Una buena gestión de eventos mejora la funcionalidad, la experiencia del usuario y el mantenimiento del código.

4.3 Clasificación de eventos

Los eventos en JavaScript pueden clasificarse según su origen:

- Eventos de ratón:
  - clics (`click`),
  - desplazamientos (`mousemove`).
- Eventos de teclado:
  - teclas presionadas (`keydown`, `keyup`).
- Eventos de formulario:
  - envío (`submit`),
  - cambios (`change`).

4.3 Clasificación de eventos

Los eventos globales, como el redimensionamiento de la ventana (`resize`), afectan a todo el entorno de la página web.

4.3 Clasificación de eventos

Cada categoría de eventos tiene propiedades específicas que los desarrolladores pueden utilizar para manejar el comportamiento esperado de los elementos.

4.3 Clasificación de eventos

La clasificación de eventos permite a los desarrolladores elegir los eventos adecuados para cada funcionalidad, optimizando la interacción con los usuarios.

4.4 Ejecución de JavaScript desde enlaces

Los enlaces HTML pueden ejecutar código JavaScript directamente mediante el uso de atributos como `onclick`.

Esto permite asociar acciones específicas a clics en los enlaces.

Ejemplo:

```
<a href="javascript: funcionJS()">Texto del enlace</a>
```

4.4 Ejecución de JavaScript desde enlaces

Aunque esta técnica es funcional, el enfoque moderno sugiere mantener la separación de estructura (HTML) y comportamiento (JavaScript) utilizando `addEventListener`.

4.4 Ejecución de JavaScript desde enlaces

La ejecución de JavaScript desde enlaces es útil para manejar acciones como navegación personalizada, apertura de ventanas o validación previa al envío de formularios.

Sin embargo, es importante evitar abusar de esta práctica para mantener el código organizado y más fácil de mantener.

5. Formularios

Los formularios son componentes esenciales en las páginas web, diseñados para recopilar datos del usuario a través de campos de entrada, botones y otros controles.

5.1 Introducción

HTML proporciona una variedad de elementos de formulario, como:

- <input>
- <textarea>
- <select>
- botones de envío (<button> o <input type='submit'>).

5.1 Introducción a formularios

Los formularios permiten al usuario interactuar con la página y enviar datos al servidor para su procesamiento, como información de registro o búsqueda.

5.1 Introducción a formularios

Además, HTML5 ha mejorado los formularios al incluir:

- tipos específicos de entrada (email, number, date) y
- atributos como `required` para validar datos directamente en el navegador.

5.1 Introducción a formularios

La comprensión y el uso efectivo de los formularios son fundamentales para crear aplicaciones web interactivas y funcionales.

5.2 Acceso a los formularios

El acceso a los formularios en JavaScript se realiza mediante el objeto `document`, utilizando propiedades como `forms`, que devuelve una colección de formularios en la página.

5.2 Acceso a los formularios

Cada formulario puede identificarse mediante su índice o su atributo `name` o `id`, lo que permite manipularlo de forma precisa.

5.2 Acceso a los formularios

Los elementos individuales de un formulario (como campos de texto o botones) pueden accederse mediante propiedades como:

- `elements` o
- selectores específicos (`getElementById`).

Este acceso permite validar entradas, ajustar valores predeterminados o responder a interacciones del usuario en tiempo real.

5.2 Acceso a los formularios

El acceso dinámico a los formularios es esencial para personalizar la experiencia del usuario y garantizar la funcionalidad de las aplicaciones web.

5.3 Propiedades, métodos y eventos de los formularios

Los formularios tienen propiedades clave como:

- action: URL donde se enviarán los datos.
- method: Método HTTP utilizado (GET o POST).
- elements: Colección de campos en el formulario.

5.3 Propiedades, métodos y eventos de los formularios

Los métodos más utilizados incluyen:

- submit(): Envía el formulario programáticamente.
- reset(): Restablece los valores de los campos a sus valores predeterminados.

5.3 Propiedades, métodos y eventos de los formularios

Los eventos más comunes de los formularios son:

- onsubmit: Se dispara cuando se envía el formulario.
- onchange: Detecta cambios en un campo.

5.3 Propiedades, métodos y eventos de los formularios

Estos elementos permiten gestionar y controlar completamente el comportamiento de los formularios, mejorando la interactividad y validación.

5.4 Envío y validación de formularios

La validación de formularios asegura que los datos introducidos por los usuarios cumplan con criterios específicos antes de enviarse al servidor.

5.4 Envío y validación de formularios

Para realizar validaciones básicas directamente en el navegador HTML5 proporciona atributos como:

- `required`,
- `pattern` y
- `min`/ `max`

5.4 Envío y validación de formularios

JavaScript permite crear validaciones personalizadas, como verificar formatos de correo electrónico, números en un rango o campos obligatorios.

5.4 Envío y validación de formularios

El evento `onsubmit` se utiliza para interceptar el envío del formulario y realizar validaciones adicionales antes de procesar los datos.

Una validación eficiente mejora la calidad de los datos enviados y garantiza una experiencia de usuario más segura y fluida.

5.4 Envío y validación de formularios

```
<form action="" method="" id="" name="" onsubmit="return validar()">
/* campos del formulario */
</form>

function validar() {
   if (condición para el primer campo) {
      // Si no se cumple...
      console.log('[ERROR] Mensaje al usuario con indicaciones...');
      return false;
   } else if (condición para el segundo campo) {
      // Si no se cumple...
      console.log('[ERROR] Mensaje al usuario con indicaciones...');
      return false;
   } else if (condición para el último campo) {
      } // Si no se cumple...
      console.log('[ERROR] Mensaje al usuario con indicaciones...');
      return false;

   // Llegados a este punto, todas las condiciones se han cumplido
   return true;
}
```