



JavaScript

Capítulo 1: Programación Básica

Variables

Declaración de Variables

- `var`: Evitar su uso; tiene alcance global o de función.
- `let`: Para variables que pueden cambiar; alcance de bloque.
- `const`: Para valores constantes; no pueden reasignarse; alcance de bloque.

```
let edad = 25;  
const PI = 3.1416;
```

Reglas de Nomenclatura

- Deben comenzar con una letra, `$` o `_`.
- No pueden ser palabras reservadas.
- Sensibles a mayúsculas y minúsculas (`nombre` \neq `Nombre`).

Tipos de Datos

Primitivos

- **Número**: Enteros y decimales.

```
let entero = 10;  
let decimal = 3.14;
```

- **Cadena de Texto** (`string`): Texto entre comillas simples o dobles.

```
let saludo = "Hola Mundo";
```

- **Caracteres Especiales:**

- `\n`: Nueva línea
- `\t`: Tabulador
- `\'`: Comilla simple
- `\"`: Comilla doble
- `\\`: Barra invertida

- **Booleano:** `true` o `false`.

```
let esVisible = true;
```

- **Undefined:** Variable declarada pero sin asignar.
- **Null:** Intencionalmente sin valor.

Complejos

- **Array:** Lista ordenada de valores.

```
let frutas = ["Manzana", "Banana", "Cereza"];
```

- **Objeto:** Colección de pares clave-valor.

```
let persona = {  
  nombre: "Juan",  
  edad: 30,  
};
```

Operadores

Aritméticos

- **Suma (+)**
- **Resta (-)**
- **Multiplicación (*)**
- **División (/)**
- **Módulo (%):** Resto de la división.

```
let resto = 17 % 5; // resto = 2
```

Asignación

- `=`: Asignación simple.
- `+=`, `=`, `=`, `/=`, `%=`: Operadores de asignación compuesta.

```
let x = 10;  
x += 5; // x = x + 5
```

Comparación

- **Igualdad Débil (`==`)**: Compara valor, realiza coerción de tipo.
- **Igualdad Estricta (`===`)**: Compara valor y tipo.
- **Desigualdad (`!=`, `!==`)**
- **Mayor, Menor, Mayor o Igual, Menor o Igual (`>`, `<`, `>=`, `<=`)**

Lógicos

- **AND (`&&`)**: Verdadero si ambas expresiones son verdaderas.
- **OR (`||`)**: Verdadero si al menos una expresión es verdadera.
- **NOT (`!`)**: Invierte el valor lógico.

```
let esAdulto = true;  
if (!esAdulto) {  
  // Código si no es adulto  
}
```

Incremento y Decremento

- **Incremento (`++`)**
- **Decremento (`-`)**

```
let contador = 0;  
contador++; // contador = 1
```

Estructuras de Control de Flujo

Condicionales

- `if`

```
if (condicion) {  
    // Código si la condición es verdadera  
}
```

- `if...else`

```
if (condicion) {  
    // Código si verdadera  
} else {  
    // Código si falsa  
}
```

- `if...else if...else`

```
if (condicion1) {  
    // Código  
} else if (condicion2) {  
    // Código  
} else {  
    // Código  
}
```

Bucles

- `for`

```
for (let i = 0; i < 10; i++) {  
    // Código que se repite 10 veces  
}
```

- `while`

```
while (condicion) {  
    // Código mientras la condición sea verdadera  
}
```

- **do...while**

```
do {  
    // Código que se ejecuta al menos una vez  
} while (condicion);
```

- **for...of** (para arrays)

```
for (let fruta of frutas) {  
    console.log(fruta);  
}
```

- **for...in** (para objetos)

```
for (let clave in persona) {  
    console.log(clave, persona[clave]);  
}
```

Funciones y Propiedades Básicas

Conversión de Tipos

- **Convertir String a Número**

```
let cadena = "42";  
let numero = Number(cadena); // numero = 42
```

- **Convertir Número a String**

```
let num = 123;  
let texto = String(num); // texto = "123"
```

- **Funciones Alternativas**

- `parseInt(cadena)` : Convierte cadena a número entero.

```
let entero = parseInt("123"); // entero = 123
```

- `parseFloat(cadena)` : Convierte cadena a número decimal.

```
let decimal = parseFloat("3.14"); // decimal = 3.14
```

Funciones para Cadenas de Texto

Los métodos para manipular cadenas de texto nos permiten realizar operaciones comunes y útiles en programación.

- **Sintaxis del Objeto String**

Aunque en JavaScript las cadenas se pueden manipular directamente, también es posible crear un objeto String:

```
let texto = new String("Hola Mundo");
```

- `length` : Devuelve la longitud de la cadena.

```
let saludo = "Hola Mundo";  
let longitud = saludo.length; // 10
```

- `charAt(indice)` : Devuelve el carácter en la posición indicada.

```
let letra = saludo.charAt(0); // "H"
```

- `substring(ind1, ind2)` : Devuelve el texto comprendido entre los índices.

```
let subcadena = saludo.substring(0, 4); // "Hola"
```

- `indexOf(cadena)` : Devuelve el índice de la primera aparición de la subcadena.

```
let indice = saludo.indexOf("Mundo"); // 5
```

- `replace(cadenaVieja, cadenaNueva)` : Reemplaza una subcadena por otra.

```
let nuevoSaludo = saludo.replace("Mundo", "JavaScript");  
// "Hola JavaScript"
```

- `toLowerCase()` : Transforma la cadena a minúsculas.

```
let minusculas = saludo.toLowerCase(); // "hola mundo"
```

- `toUpperCase()` : Transforma la cadena a mayúsculas.

```
let mayusculas = saludo.toUpperCase(); // "HOLA MUNDO"
```

- `split(separador)` : Convierte la cadena en un array, separando por el delimitador indicado.

```
let palabras = saludo.split(" "); // ["Hola", "Mundo"]
```

Funciones Matemáticas

JavaScript proporciona una serie de métodos matemáticos a través del objeto `Math`, que permiten realizar operaciones complejas.

- **Métodos Matemáticos Comunes:**

Método	Descripción	Ejemplo
<code>Math.sin(x)</code>	Devuelve el seno de un ángulo (en radianes).	<code>Math.sin(0)</code> // 0
<code>Math.cos(x)</code>	Devuelve el coseno de un ángulo (en radianes).	<code>Math.cos(0)</code> // 1
<code>Math.tan(x)</code>	Devuelve la tangente de un ángulo (en radianes).	<code>Math.tan(0)</code> // 0
<code>Math.abs(x)</code>	Devuelve el valor absoluto de un número.	<code>Math.abs(-5)</code> // 5
<code>Math.log(x)</code>	Devuelve el logaritmo natural de un número.	<code>Math.log(1)</code> // 0
<code>Math.max(x, y, ...)</code>	Devuelve el número mayor entre los proporcionados.	<code>Math.max(5, 10)</code> // 10
<code>Math.min(x, y, ...)</code>	Devuelve el número menor entre los proporcionados.	<code>Math.min(5, 10)</code> // 5

<code>Math.pow(base, exponente)</code>	Calcula la potencia de un número.	<code>Math.pow(2, 3) // 8</code>
<code>Math.sqrt(x)</code>	Calcula la raíz cuadrada de un número.	<code>Math.sqrt(16) // 4</code>
<code>Math.round(x)</code>	Redondea un número al entero más cercano.	<code>Math.round(3.6) // 4</code>
<code>Math.random()</code>	Genera un número aleatorio entre 0 y 1.	<code>Math.random()</code> // Ejemplo: 0.823432
<code>Math.floor(x)</code>	Redondea hacia abajo al entero más cercano.	<code>Math.floor(3.9) // 3</code>
<code>Math.ceil(x)</code>	Redondea hacia arriba al entero más cercano.	<code>Math.ceil(3.1) // 4</code>

- **Generar Número Aleatorio en un Rango:**

```
// Número aleatorio entre 0 y 10
let num = Math.random() * 10;

// Número aleatorio entero entre 1 y 100
let enteroAleatorio = Math.floor(Math.random() * 100) + 1;
```

- **Funciones Trigonométricas Inversas:**

```
Math.asin(x); // Arco seno de x
Math.acos(x); // Arco coseno de x
Math.atan(x); // Arco tangente de x
```

Función `eval()`

- `eval(cadena)`: Evalúa una cadena de texto como código JavaScript.

```
let expresion = "2 + 2";
let resultado = eval(expresion); // resultado = 4
```

Nota: El uso de `eval()` es generalmente desaconsejado por razones de seguridad y rendimiento. Úsalo con

| precaución.

Capítulo 2: Programación Avanzada

Funciones

Declaración de Funciones

- **Función Declarada**

```
function nombreFuncion(parametros) {  
  // Código  
  return resultado;  
}
```

- **Función Expresada (Anónima)**

```
let sumar = function (a, b) {  
  return a + b;  
};
```

- **Funciones Flecha (ES6+)**

```
let restar = (a, b) => a - b;
```

Parámetros y Argumentos

- **Parámetros:** Variables en la definición.
- **Argumentos:** Valores al invocar la función.

```
function saludar(nombre) {  
  return "Hola, " + nombre;  
}  
saludar("Ana"); // "Hola, Ana"
```

Valores de Retorno

- **return:** Finaliza la función y devuelve un valor.

```
function multiplicar(a, b) {  
  return a * b;  
}
```

Ámbito de Variables (Scope)

- **Local:** Dentro de una función o bloque.
- **Global:** Fuera de funciones y bloques.

```
let global = "Variable global";  
  
function ejemplo() {  
  let local = "Variable local";  
  console.log(global); // Accede a variable global  
}  
  
console.log(local); // Error: local no está definida
```

Sentencias `break` y `continue`

- `break`: Sale del bucle actual.

```
for (let i = 0; i < 10; i++) {  
  if (i === 5) break;  
  console.log(i); // 0 a 4  
}
```

- `continue`: Salta a la siguiente iteración.

```
for (let i = 0; i < 10; i++) {  
  if (i % 2 === 0) continue;  
  console.log(i); // Números impares  
}
```

Estructuras de Control Adicionales

`switch`

- Estructura condicional múltiple.

```
switch (expresion) {  
  case valor1:  
    // Código  
    break;  
  case valor2:  
    // Código  
    break;  
  default:  
    // Código por defecto  
}
```

Operador Ternario

- Sintaxis concisa para condiciones simples.

```
let acceso = edad >= 18 ? "Permitido" : "Denegado";
```

Temporizadores

setTimeout()

- Ejecuta una función después de un retraso específico (en milisegundos).

```
setTimeout(functionLlamada, tiempo);
```

- **Ejemplo:**

```
function mostrarMensaje() {  
  alert("¡Hola después de 2 segundos!");  
}  
  
setTimeout("mostrarMensaje()", 2000);
```

Nota: Es preferible pasar la función directamente sin comillas:

```
setTimeout(mostrarMensaje, 2000);
```

Capítulo 3: Document Object Model (DOM)

Introducción al DOM

- **Definición:** Representación en forma de árbol de todos los elementos de una página web.
- **Nodos:**
 - **Document:** Nodo raíz que representa todo el documento.
 - **Element:** Cada etiqueta HTML.
 - **Text:** El contenido de texto dentro de un elemento.
 - **Attribute:** Atributos de los elementos.
 - **Comment:** Comentarios en el código HTML.

Selección de Elementos

Métodos Clásicos

- `document.getElementById(id)`

```
let titulo = document.getElementById("tituloPrincipal");
```

- `document.getElementsByTagName(etiqueta)`

```
let parrafos = document.getElementsByTagName("p");
```

- `document.getElementsByClassName(clase)`

```
let elementosDestacados = document.getElementsByClassName("destacado");
```

- `document.getElementsByName(name)`

```
let camposFormulario = document.getElementsByName("usuar
```

```
io");
```

Métodos Modernos

- `document.querySelector(selector)`

- Selecciona el primer elemento que coincide con el selector CSS.

```
let primerElemento = document.querySelector(".clase .  
subclase");
```

- `document.querySelectorAll(selector)`

- Selecciona todos los elementos que coinciden.

```
let todosLosEnlaces = document.querySelectorAll("a");
```

Manipulación del DOM

Creación de Elementos

- **Crear un nuevo elemento**

```
let nuevoDiv = document.createElement("div");
```

- **Crear un nodo de texto**

```
let texto = document.createTextNode("Hola Mundo");
```

- **Añadir contenido al elemento**

```
nuevoDiv.appendChild(texto);
```

Inserción en el DOM

- **Añadir al final del cuerpo**

```
document.body.appendChild(nuevoDiv);
```

- **Insertar antes de un elemento existente usando `insertBefore()`**

```
let referencia = document.getElementById("referencia");
let padre = referencia.parentNode;
padre.insertBefore(nuevoDiv, referencia);
```

- **Explicación:** `insertBefore()` inserta un nodo antes de otro nodo hijo específico. Se aplica al nodo padre.

```
padre.insertBefore(nuevoNodo, nodoReferencia);
```

- **Insertar en una posición específica**

```
let lista = document.getElementById("miLista");
let nuevoElemento = document.createElement("li");
nuevoElemento.textContent = "Nuevo ítem";

let primerItem = lista.firstChild;
lista.insertBefore(nuevoElemento, primerItem); // Insert
a al inicio de la lista
```

Clonación de Elementos usando `cloneNode()`

- **Clonar un nodo existente**

```
let elementoOriginal = document.getElementById("elemento
Original");
let copiaElemento = elementoOriginal.cloneNode(true);
```

- **Parámetro `true`:** Clona el nodo y todos sus descendientes (clonación profunda).
- **Parámetro `false`:** Clona solo el nodo sin sus hijos (clonación superficial).

- **Insertar el nodo clonado en el DOM**

```
document.body.appendChild(copiaElemento);
```

- **Ejemplo Práctico**

```
// Clonar un elemento de lista y añadirlo al final
let itemOriginal = document.querySelector(".lista .item");
let copiaItem = itemOriginal.cloneNode(true);

let lista = document.querySelector(".lista");
lista.appendChild(copiaItem);
```

Reemplazo de Elementos usando `replaceChild()`

- Reemplazar un elemento por otro

```
let nuevoElemento = document.createElement("section");
nuevoElemento.textContent = "Contenido nuevo";

let elementoViejo = document.getElementById("viejo");
let padre = elementoViejo.parentNode;
padre.replaceChild(nuevoElemento, elementoViejo);
```

- **Explicación:** `replaceChild()` reemplaza un nodo hijo por otro. Se aplica al nodo padre.

```
padre.replaceChild(nuevoNodo, nodoExistente);
```

Eliminación de Elementos

- Eliminar un elemento existente

```
let elementoAEliminar = document.getElementById("elemento");
elementoAEliminar.parentNode.removeChild(elementoAEliminar);
```

Atributos y Propiedades

Manipulación de Atributos

- Obtener un atributo

```
let enlace = document.getElementById("enlace");  
let url = enlace.getAttribute("href");
```

- **Establecer un atributo**

```
enlace.setAttribute("href", "https://nueva-url.com");
```

- **Eliminar un atributo**

```
enlace.removeAttribute("target");
```

Manipulación de Clases

- **Añadir una clase**

```
elemento.classList.add("nueva-clase");
```

- **Eliminar una clase**

```
elemento.classList.remove("clase-antigua");
```

- **Comprobar si tiene una clase**

```
elemento.classList.contains("activo");
```

- **Alternar una clase**

```
elemento.classList.toggle("visible");
```

Estilos en Línea

- **Modificar estilos directamente**

```
elemento.style.color = "red";  
elemento.style.backgroundColor = "#f0f0f0";
```


Nota: Las propiedades CSS con guiones se convierten a camelCase en JavaScript.

Trabajando con Contenido de Texto

- **Modificar el contenido de texto de un elemento**

```
elemento.textContent = "Nuevo contenido de texto";
```

- **Obtener el contenido de texto**

```
let texto = elemento.textContent;
```

Trabajando con Formularios

- **Obtener el valor de un input**

```
let input = document.getElementById("miInput");  
let valor = input.value;
```

- **Establecer el valor de un input**

```
input.value = "Nuevo valor";
```

Objetos del Navegador

Además del DOM, JavaScript proporciona objetos que permiten interactuar con diferentes aspectos del navegador y la pantalla del usuario. Los más comunes son `window`, `document`, `navigator` y `screen`.

Objeto `window`

El objeto `window` representa la ventana del navegador y es el objeto global en el contexto del navegador. Permite controlar aspectos de la ventana y abrir nuevas ventanas.

Métodos y Propiedades del Objeto `window`:

Método/Propiedad	Descripción	Sintaxis
<code>open()</code>	Abre una nueva ventana.	<pre>var nuevaVentana = window.open(url, nombre,</pre>

		<code>atributos);</code>
<code>close()</code>	Cierra la ventana actual o la especificada.	<code>nuevaVentana.close();</code>
<code>opener</code>	Referencia a la ventana que abrió la ventana actual.	<code>var openerVentana = nuevaVentana.opener;</code>
<code>closed</code>	Indica si la ventana ha sido cerrada.	<code>var estaCerrada = nuevaVentana.closed;</code>
<code>location</code>	Obtiene o establece la URL de la ventana.	<code>window.location = "https://www.ejemplo.com";</code>
<code>print()</code>	Abre el diálogo de impresión para la ventana actual.	<code>window.print();</code>
<code>alert()</code>	Muestra un cuadro de diálogo de alerta.	<code>window.alert("Mensaje");</code>
<code>confirm()</code>	Muestra un cuadro de diálogo de confirmación.	<code>var respuesta = window.confirm("¿Estás seguro?");</code>
<code>prompt()</code>	Muestra un cuadro de diálogo para ingresar texto.	<code>var entrada = window.prompt("Ingrese su nombre:", "Valor por defecto");</code>
<code>status</code>	Establece el texto en la barra de estado del navegador (no soportado en todos los navegadores).	<code>window.status = "Cargando...";</code>
<code>setTimeout()</code>	Ejecuta una función después de un retraso especificado.	<code>setTimeout(funcion, tiempo);</code>
<code>showModalDialog()</code>	Crea una ventana modal (obsoleto en navegadores modernos).	<code>var retorno = window.showModalDialog(url, argumentos, atributos);</code>

Objeto `document`

El objeto `document` representa el documento HTML cargado en la ventana. Proporciona métodos y propiedades para interactuar con el contenido de la página.

Métodos y Propiedades del Objeto `document` :

Método/Propiedad	Descripción	Sintaxis
------------------	-------------	----------

<code>write()</code>	Escribe contenido directamente en el documento.	<code>document.write("Texto");</code>
<code>writeln()</code>	Escribe contenido y añade un salto de línea.	<code>document.writeln("Texto");</code>
<code>bgColor</code>	Establece el color de fondo de la página.	<code>document.bgColor = "#FFFFFF";</code>
<code>fgColor</code>	Establece el color del texto de la página.	<code>document.fgColor = "#000000";</code>
<code>linkColor</code>	Establece el color de los enlaces no visitados.	<code>document.linkColor = "#0000FF";</code>
<code>vlinkColor</code>	Establece el color de los enlaces visitados.	<code>document.vlinkColor = "#800080";</code>
<code>location</code>	Obtiene la URL actual del documento.	<code>var urlActual = document.location;</code>
<code>lastModified</code>	Obtiene la fecha de última modificación del documento.	<code>var fechaModificacion = document.lastModified;</code>
<code>referrer</code>	Obtiene la URL de la página que enlazó al documento actual.	<code>var urlReferrer = document.referrer;</code>

Objeto `navigator`

El objeto `navigator` contiene información sobre el navegador del usuario.

Propiedades del Objeto `navigator`:

Propiedad	Descripción	Sintaxis
<code>appName</code>	Nombre del navegador.	<code>var nombreNavegador = navigator.appName;</code>
<code>appVersion</code>	Versión del navegador.	<code>var versionNavegador = navigator.appVersion;</code>
<code>userAgent</code>	Información del agente de usuario.	<code>var agenteUsuario = navigator.userAgent;</code>
<code>platform</code>	Plataforma del sistema operativo.	<code>var plataforma = navigator.platform;</code>
<code>language</code>	Idioma del navegador.	<code>var idioma = navigator.language;</code>

Objeto `screen`

El objeto `screen` proporciona información sobre la configuración de la pantalla del usuario.

Propiedades del Objeto `screen`:

Propiedad	Descripción	Sintaxis
<code>width</code>	Ancho total de la pantalla en píxeles.	<pre>var anchoPantalla = screen.width;</pre>
<code>height</code>	Alto total de la pantalla en píxeles.	<pre>var altoPantalla = screen.height;</pre>
<code>colorDepth</code>	Profundidad de color de la pantalla (bits por píxel).	<pre>var profundidadColor = screen.colorDepth;</pre>

Capítulo 4: Eventos

Conceptos Básicos de Eventos

- **Eventos:** Acciones que ocurren en la página (clics, teclas presionadas, carga de la página, etc.).
- **Programación Basada en Eventos:** El código se ejecuta en respuesta a acciones del usuario o del navegador.
- **Nomenclatura de Eventos:** Los nombres de los eventos comienzan con el prefijo `on` seguido del nombre en inglés de la acción asociada. Por ejemplo, `onclick`, `onmouseover`.

Tipos de Eventos

En JavaScript, cada elemento HTML puede tener asociados diferentes tipos de eventos. Un mismo evento puede aplicarse a múltiples elementos, y un elemento puede tener múltiples eventos.

Eventos Comunes

Evento	Descripción	Elementos Aplicables	Ejemplo
<code>onblur</code>	Cuando un elemento pierde el foco	<pre><input> , <select> , <textarea> , <button></pre>	<pre>elemento.onblur = funcion;</pre>

<code>onchange</code>	Cuando el valor de un elemento cambia	<code><input></code> , <code><select></code> , <code><textarea></code>	<code>elemento.onchange = funcion;</code>
<code>onclick</code>	Cuando se hace clic en un elemento	Todos los elementos	<code>elemento.onclick = funcion;</code>
<code>ondblclick</code>	Doble clic sobre un elemento	Todos los elementos	<code>elemento.ondblclick = funcion;</code>
<code>onfocus</code>	Cuando un elemento recibe el foco	<code><input></code> , <code><select></code> , <code><textarea></code> , <code><button></code>	<code>elemento.onfocus = funcion;</code>
<code>onkeydown</code>	Al presionar una tecla (sin soltar)	Elementos de formulario y <code><body></code>	<code>elemento.onkeydown = funcion;</code>
<code>onkeypress</code>	Al presionar una tecla	Elementos de formulario y <code><body></code>	<code>elemento.onkeypress = funcion;</code>
<code>onkeyup</code>	Al soltar una tecla	Elementos de formulario y <code><body></code>	<code>elemento.onkeyup = funcion;</code>
<code>onload</code>	Cuando la página ha cargado completamente	<code><body></code>	<code>window.onload = funcion;</code>
<code>onmousedown</code>	Al presionar un botón del ratón	Todos los elementos	<code>elemento.onmousedown = funcion;</code>
<code>onmousemove</code>	Al mover el ratón sobre un elemento	Todos los elementos	<code>elemento.onmousemove = funcion;</code>
<code>onmouseout</code>	Cuando el ratón sale de un elemento	Todos los elementos	<code>elemento.onmouseout = funcion;</code>
<code>onmouseover</code>	Cuando el ratón entra en un elemento	Todos los elementos	<code>elemento.onmouseover = funcion;</code>
<code>onmouseup</code>	Al soltar un botón del ratón	Todos los elementos	<code>elemento.onmouseup = funcion;</code>
<code>onreset</code>	Al reiniciar un formulario	<code><form></code>	<code>formulario.onreset = funcion;</code>
<code>onresize</code>	Al cambiar el tamaño de la ventana	<code><body></code>	<code>window.onresize = funcion;</code>

<code>onselect</code>	Cuando se selecciona un texto	<code><input></code> , <code><textarea></code>	<code>elemento.onselect = funcion;</code>
<code>onsubmit</code>	Al enviar un formulario	<code><form></code>	<code><form onsubmit="return funcion();"></code>
<code>onunload</code>	Al abandonar la página	<code><body></code>	<code>window.onunload = funcion;</code>

Manejadores de Eventos

Asignación Mediante HTML (No Recomendado)

```
<button onclick="alert('Hola')">Haz clic</button>
```

Asignación Mediante Propiedades DOM

```
elemento.onclick = funcion;
```

- **Ejemplo:**

```
let boton = document.getElementById("miBoton");
boton.onclick = function () {
    alert("¡Botón presionado!");
};
```

Uso de `addEventListener` (Recomendado)

- **Sintaxis**

```
elemento.addEventListener("evento", funcion, usoCaptura);
```

- `evento`: Tipo de evento (e.g., `"click"`, `"mouseover"`).
- `funcion`: Función que maneja el evento.
- `usoCaptura`: Booleano que indica la fase de propagación (por defecto, `false` para burbuja).

- **Ejemplo**

```
boton.addEventListener("click", funcion);
```

Obteniendo Información del Evento

Para acceder al objeto `event` y obtener información sobre el evento ocurrido, se puede definir una función manejadora que reciba el evento como parámetro.

```
function manejadorEventos(elEvento) {  
    var evento = elEvento || window.event;  
    // Código que utiliza el objeto evento  
}
```

Ejemplo Práctico

```
function resalta(elEvento) {  
    var evento = elEvento || window.event;  
    switch (evento.type) {  
        case "mouseover":  
            this.style.borderColor = "black";  
            break;  
        case "mouseout":  
            this.style.borderColor = "silver";  
            break;  
    }  
}  
  
window.onload = function () {  
    var seccion = document.getElementById("seccion");  
    seccion.onmouseover = resalta;  
    seccion.onmouseout = resalta;  
};
```

HTML Asociado:

```
<div  
    id="seccion"  
    style="width:150px; height:60px; border:thin solid silve
```

```
r"
>
  Sección de contenidos...
</div>
```

- **Explicación:** La función `resalta` cambia el color del borde del elemento según el tipo de evento (`mouseover` o `mouseout`). Utiliza `this` para referirse al elemento que generó el evento.

Objeto `event`

- **Acceso al Objeto** `event`

```
elemento.onclick = function (event) {
  // Código que utiliza el objeto event
};
```

Propiedades Comunes

- `event.type` : Tipo de evento.
- `event.target` : Elemento que originó el evento.
- `event.currentTarget` : Elemento al que se le asignó el manejador.
- `event.preventDefault()` : Evita la acción por defecto del evento.
- `event.stopPropagation()` : Detiene la propagación del evento.

Propagación y Delegación de Eventos

Fases de Propagación

1. **Captura:** El evento se propaga desde el elemento más externo hasta el objetivo.
2. **Objetivo:** El evento alcanza el elemento objetivo.
3. **Burbuja:** El evento se propaga desde el objetivo hacia los elementos padres.

Detener la Propagación

- `event.stopPropagation()`


```
elemento.addEventListener("click", function (event) {  
    event.stopPropagation();  
    // Código específico  
});
```

Delegación de Eventos

- Asignar un evento a un elemento padre para manejar eventos en elementos hijos.

```
contenedor.addEventListener("click", function (event) {  
    if (event.target.matches(".boton")) {  
        // Acción para elementos con clase 'boton'  
    }  
});
```

Prevención de Comportamientos por Defecto

- `event.preventDefault()`
 - Evita acciones como seguir un enlace o enviar un formulario.

```
enlace.addEventListener("click", function (event) {  
    event.preventDefault();  
    // Código alternativo  
});
```

Notas Adicionales y Consejos

Carga del Documento

`window.onload`

- Se ejecuta cuando todos los recursos (imágenes, scripts, etc.) han cargado.

```
window.onload = function () {  
    // Código a ejecutar
```

```
};
```

DOMContentLoaded

- Se ejecuta cuando el DOM ha sido completamente cargado, sin esperar a recursos externos.

```
document.addEventListener("DOMContentLoaded", function  
( ) {  
    // Código a ejecutar  
});
```