JAVASCRIPT

DISEÑO WEB

Carlos Rojas Sánchez

Licenciatura en Informática

Universidad del Mar

Contenido

Introducción

¿Qué es Javascript?

Javascript es un lenguaje de programación. Es un mecanismo con el que podemos decirle a nuestro navegador que tareas debe realizar. [1]

La consola Javascript

La consola Javascript es una zona del navegador ubicada en las DevTools donde podemos escribir pequeños fragmentos de código y observar los resultados, así como revisar mensajes de información, error u otros detalles similares.

La consola Javascript

El método `console.log()`
Es una función integrada de JavaScript utilizada para imprimir mensajes en la consola del navegador o de un entorno de ejecución como Node.js. Es una herramienta común para depuración y para observar valores durante la ejecución de un programa.

Javascript en línea

Usar la etiqueta <script> para contener las órdenes o líneas de Javascript que le indican al navegador que tiene que hacer.

```
<html>
  <head>
    <title>Título de la página</title>
    <script>
      console.log("¡Hola!");
    </script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

Javascript en externo

Se utiliza la etiqueta $<$ *script* $>$, sólo que en este caso, haremos referencia al archivo Javascript con un atributo src (source),

```
<html>
  <head>
    <title>Título de la página</title>
    <script src="js/index.js"></script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

Ubicación de la etiqueta script

Ubicación	¿Cómo descarga el archivo Javascript?	Estado de la página
En <head>	**ANTES** de empezar a dibujar la página.	Página aún no dibujada.
En <body>	**DURANTE** el dibujado de la página.	Dibujada hasta donde está la etiqueta <script>.
Antes de </body>	**DESPUÉS** de dibujar la página.	Dibujada al 100%.

Javascript

Ed.	Fecha	ECMAScript	Cambios significativos
6	JUN/2015	ES2015	Clases, módulos, hashmaps, sets, for of, proxies...
7	JUN/2016	ES2016	Array includes(), Exponenciación **
8	JUN/2017	ES2017	Async/await
9	JUN/2018	ES2018	Rest/Spread operator, Promise.finally()...
10	JUN/2019	ES2019	flat, flatMap, trimStart(), optional error catch...
11	JUN/2020	ES2020	Dynamic imports, BigInt, Promise.allSettled
12	JUN/2021	ES2021	Promise.any, globalThis, replaceAll(), flat(), flatMap()...
13	JUN/2022	ES2022	at(), top-level await modules, private fields...
14	JUN/2023	ES2023	findLast(), hashbang, toReversed(), toSorted()...
15	JUN/2024	ES2024	groupBy, withResolvers, waitAsync, isWellFormed...

Tipos de datos en Javascript

Tipo de dato	Descripción
Tipos de datos primitivos	
NUMBER Number	Valor numérico (enteros, decimales...)
BIGINT BigInt	Valor numérico muy grande
STRING String	Valor de texto (cadenas de texto, carácteres...)
BOOLEAN Boolean	Valor booleano (valores verdadero o falso)
SYMBOL Symbol	Símbolo (valor único)
UNDEFINED undefined	Valor sin definir (variable sin inicializar)
Tipos de datos no primitivos	
OBJECT Object	Objeto (estructura más compleja)
FUNCTION Function	Función (función guardada en una variable)

Funciones

	Ejemplo	Descripción
1	`function nombre(p1, p2...) { }`	Mediante **declaración** (la más usada por principiantes)
2	`let nombre = function(p1, p2...) { }`	Mediante **expresión** (la más habitual en programadores con experiencia)
3	`new Function(p1, p2..., code);`	Mediante un constructor de **objeto** (no recomendada)

Funciones por declaración

```
function saludar() {
  return "Hola";
}

saludar();      // 'Hola'
```

Funciones por expresión

```
const saludo = function saludar() {
  return "Hola";
};

saludo(); // 'Hola'
```

Funciones como objeto

```
const saludar = new Function("return 'Hola';");

saludar(); // 'Hola'
```

Funciones autoejecutables

```
(function () {
  console.log("Hola!!");
})();
```

```
(function (name) {
  console.log(`¡Hola, ${name}!`);
})("Manz");
```

Funciones Callback

Un callback (llamada hacia atrás) es pasar una función por parámetro a otra función, de modo que esta última función puede ejecutar la función pasada por parámetro de forma genérica desde su propio código, y nosotros podemos definirlas desde fuera de dicha función.

Funciones Callback

```
const action = function () {
  console.log("Acción ejecutada.");
};

const mainFunction = function (callback) {
  callback();
};

mainFunction(action);
```

Funciones de orden superior

Las funciones de orden superior o HOF (High Order Functions) son funciones que reciben por parámetro otra función y/o devuelven una función mediante el return.

Funciones de orden superior

```
const action = function () {
  console.log("Acción ejecutada.");
};

const error = function () {
  console.error("Ha ocurrido un error");
};

const doTask = function (callback, callbackError) {
  const isError = Math.random() < 0.5;

  if (!isError) callback();
  else callbackError();
};

doTask(action, error);
```

Funciones Arrow

Las Arrow functions, funciones flecha o «fat arrow» son una forma corta y compacta de escribir las funciones tradicionales de Javascript. A grandes rasgos, se trata de eliminar la palabra function y añadir el texto $=>$ antes de abrir las llaves.

Funciones Arrow

```
const func = function () {
  return "Función tradicional.";
};

const func = () ⇒ {
  return "Función flecha.";
};
```

```
// 0 parámetros: Devuelve "Función flecha"
const func = () ⇒ "Función flecha.";

// 1 parámetro:  Devuelve el valor de e + 1
const func = (e) ⇒ e + 1;

// 2 parámetros: Devuelve el valor de a + b
const func = (a, b) ⇒ a + b;
```

Función Generadora

Una función generadora es una función que se puede pausar y reanudar, y por lo tanto, nos puede devolver múltiples valores. En lugar de llamar a return para devolver un valor, utilizamos yield.

```
function* generator() {
  // code
  yield "One";
  // code
  yield "Two";
  // code
  yield "Three";
}

const values = generator();
console.log(values.next());
console.log(values.next());
console.log(values.next());
console.log(values.next());
```

Operadores

- Aritméticos
- Asignación
- Unarios
- Comparación
  - Igualdad
  - Identidad
- Lógicos

Eventos mediante HTML

Eventos mediante HTML

Un evento Javascript es una característica especial que ha sucedido en nuestra página y a la cuál le asociamos una funcionalidad, de modo que se ejecute cada vez que suceda dicho evento.

Eventos mediante HTML

```
<script>
  function doTask() {
    alert("Hello!");
  }
</script>

<button onClick="doTask()">Saludar</button>
```

Eventos del mouse en HTML

Se activan mediante acciones del ratón sobre un elemento.

- `onclick` – Clic sobre el elemento
- `ondblclick` – Doble clic
- `onmouseover` – El mouse entra al elemento
- `onmouseout` – El mouse sale del elemento
- `onmousemove` – Movimiento del mouse
- `oncontextmenu` – Clic derecho

Eventos de teclado en HTML

Permiten detectar la interacción del usuario con el teclado.

- `onkeydown` – Tecla presionada
- `onkeyup` – Tecla liberada
- `onkeypress` – Tecla presionada (obsoleto)

Uso común: validación y detección de atajos de teclado.

Eventos de formularios

Se utilizan principalmente en `input`, `select` y `form`.

- `onchange` – Cambio de valor
- `oninput` – Entrada de datos en tiempo real
- `onsubmit` – Envío del formulario
- `onfocus` – El elemento recibe foco
- `onblur` – El elemento pierde foco
- `oninvalid` – Validación fallida

Eventos del documento y la ventana

Relacionados con el estado general de la página web.

- `onload` – Página completamente cargada
- `onunload` – Salida de la página
- `onresize` – Cambio de tamaño de la ventana
- `onscroll` – Desplazamiento vertical u horizontal

Eventos multimedia

Aplican a elementos como `audio` y `video`.

- `onplay` – Inicia la reproducción
- `onpause` – Pausa del contenido
- `onended` – Finaliza el contenido
- `onvolumechange` – Cambio de volumen
- `ontimeupdate` – Avance del tiempo de reproducción

Eventos en JavaScript

Eventos en JavaScript

Los eventos permiten ejecutar código cuando ocurre una acción del usuario o del navegador.

- Interacciones del usuario (clics, teclado, mouse)
- Cambios en formularios
- Estados del documento o multimedia

JavaScript moderno: `addEventListener()`

Ejemplo básico de evento en JavaScript

Separación de HTML y JavaScript

- HTML define la estructura
- JavaScript maneja los eventos

```
button.addEventListener('click', funcion);
```

Eventos del mouse en JavaScript

Se registran sobre elementos del DOM.

- `click`
- `dblclick`
- `mouseover` / `mouseout`
- `mouseenter` / `mouseleave`
- `mousemove`
- `contextmenu`

Eventos de teclado en JavaScript

Se detectan normalmente en `document` o campos de texto.

- `keydown` – Tecla presionada
- `keyup` – Tecla liberada

Uso típico: atajos de teclado y validaciones.

Eventos de formularios en JavaScript

Permiten controlar la entrada de datos del usuario.

- `submit`
- `change`
- `input`
- `focus`
- `blur`
- `invalid`

Eventos del documento y la ventana

Se usan para detectar cambios globales del navegador.

- `DOMContentLoaded`
- `load`
- `resize`
- `scroll`
- `beforeunload`

Eventos multimedia en JavaScript

Aplican a elementos `audio` y `video`.

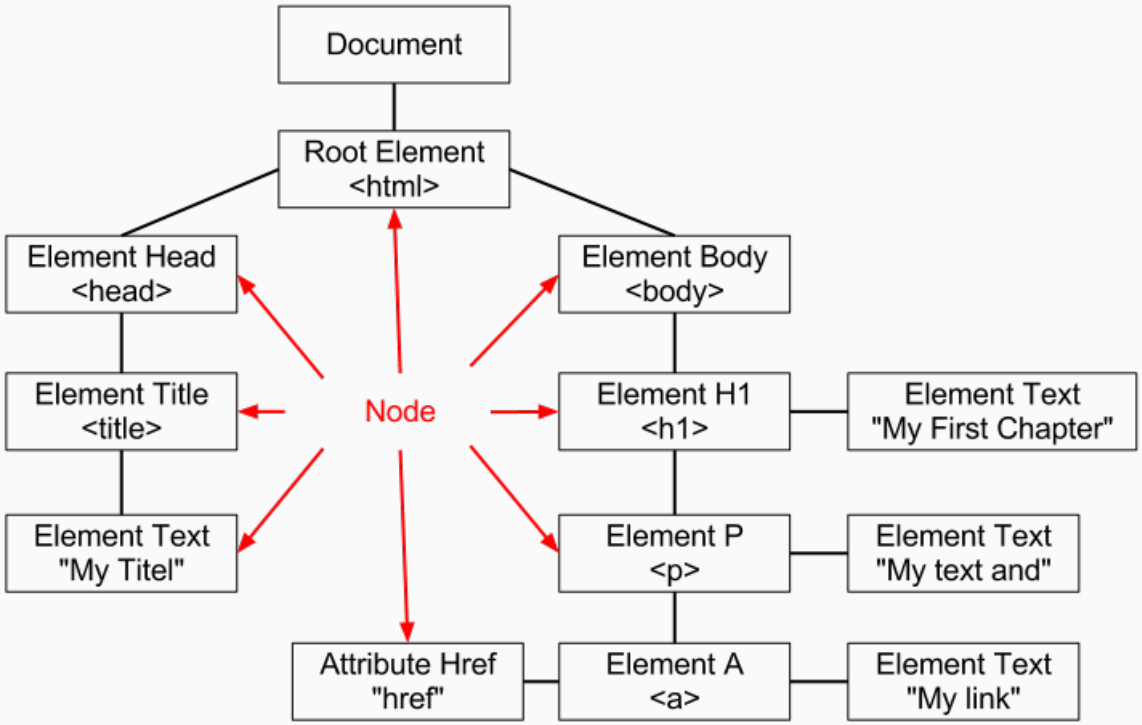
- `play`
- `pause`
- `ended`
- `timeupdate`
- `volumechange`

Modelo de Objetos del Documento (DOM)

¿Qué es el DOM?

El modelo de objeto de documento (DOM) es una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su estructura, estilo y contenido. El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. Esencialmente, conecta las páginas web a scripts o lenguajes de programación.

DOM


Document
Root Element
<html>
Element Head
<head>
Element Body
<body>
Element Title
<title>
Node
Element H1
<h1>
Element Text
"My First Chapter"
Element Text
"My Titel"
Element P
<p>
Element Text
"My text and"
Attribute Href
"href"
Element A
<a>
Element Text
"My link"

DOM

El DOM es una representación completamente orientada al objeto de la página web y puede ser modificado con un lenguaje de script como JavaScript. El W3C DOM estándar forma la base del funcionamiento del DOM en muchos navegadores modernos.

DOM y JavaScript

El DOM no es un lenguaje de programación pero sin él, el lenguaje JavaScript no tiene ningún modelo o noción de las páginas web, de la páginas XML ni de los elementos con los cuales es usualmente relacionado. Cada elemento -"el documento íntegro, el título, las tablas dentro del documento, los títulos de las tablas, el texto dentro de las celdas de las tablas"- es parte del modelo de objeto del documento para cada documento, así se puede acceder y manipularlos utilizando el DOM y un lenguaje de escritura, como JavaScript.

El objeto document

Desde el navegador, la forma de acceder al DOM es a través de un objeto Javascript llamado document, que representa el árbol DOM de la página o, más concretamente, la página de la pestaña del navegador donde nos encontramos. En su interior pueden existir varios tipos de elementos, pero principalmente serán objetos de tipo ELEMENT o NODE:

Métodos tradicionales de búsqueda

Métodos de búsqueda	Descripción	Si no lo encuentra...
ELEMENT `.getElementById(id)`	Busca el elemento HTML por su `id`.	Devuelve `NULL`.
ARRAY `.getElementsByClassName(class)`	Busca elementos con la clase `class`.	Devuelve `[]`.
ARRAY `.getElementsByName(value)`	Busca elementos con el atributo `name` a `value`.	Devuelve `[]`.
ARRAY `.getElementsByTagName(tag)`	Busca etiquetas HTML `tag`.	Devuelve `[]`.

Modificación de contenido

Permite cambiar texto y contenido HTML.

- `element.textContent`
- `element.innerText`
- `element.innerHTML`

Nota: `innerHTML` puede ser riesgoso si se usa con datos del usuario.

Modificación de contenido

Cambiar texto de un elemento

```
const titulo = document.getElementById("titulo");
titulo.textContent = "Nuevo texto";
```

Opciones comunes:

- `textContent`
- `innerText`
- `innerHTML`

Atributos y estilos del DOM

- `element.getAttribute()`
- `element.setAttribute()`
- `element.removeAttribute()`
- `element.style.propiedad`
- `element.classList`

Manipulación de atributos y estilos

Modificar atributos HTML

```
const img = document.querySelector("img");
img.setAttribute("alt", "Imagen ejemplo");
```

Modificar estilos CSS

```
img.style.border = "2px solid blue";
```

Manipulación de clases CSS

La forma recomendada de manejar clases.

- `classList.add()`
- `classList.remove()`
- `classList.toggle()`
- `classList.contains()`

Manipulación de clases CSS

Uso de classList

```
const caja = document.querySelector(".caja");
caja.classList.toggle("activo");
```

Métodos principales:

- `add()`
- `remove()`
- `toggle()`

Crear y eliminar elementos del DOM

- `document.createElement()`
- `element.appendChild()`
- `element.remove()`
- `element.replaceWith()`

Crear y eliminar elementos

Crear un nuevo nodo

```
const p = document.createElement("p");
p.textContent = "Texto dinámico";
document.body.appendChild(p);
```

Eliminar un nodo

```
p.remove();
```

Recorrido del DOM

Permite navegar entre nodos.

- `parentElement`
- `children`
- `firstElementChild`
- `lastElementChild`
- `nextElementSibling`

Recorrido del DOM

Navegar entre elementos

```
const lista = document.querySelector("ul");
const primerItem = lista.firstElementChild;
```

Propiedades comunes:

- `parentElement`
- `children`
- `nextElementSibling`

Relación entre DOM y eventos

- Se selecciona un elemento del DOM
- Se registra un evento
- Se ejecuta una acción sobre el DOM

Base del desarrollo web interactivo

Importación y exportación de funciones JavaScript

Importación y exportación en JavaScript

La importación y exportación permiten dividir el código en archivos independientes llamados **módulos**.

- Mejor organización del código
- Reutilización de funciones
- Mantenimiento más sencillo

Módulos en JavaScript

Un módulo es un archivo JavaScript que:

- Contiene funciones, variables o clases
- Puede exportar elementos
- Puede importar elementos de otros módulos

Extensión común: `.js`

Exportación nombrada

Permite exportar múltiples elementos desde un módulo.

```
export function suma(a, b) {
    return a + b;
}

export function resta(a, b) {
    return a - b;
}
```

Importación nombrada

Se deben usar los mismos nombres exportados.

```
import { suma, resta } from "./operaciones.js";

console.log(suma(5, 3));
```

Exportación por defecto

Solo puede existir una exportación por defecto por archivo.

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

Importación por defecto

El nombre puede ser cualquiera.

```
import multi from "./multiplicar.js";

console.log(multi(4, 2));
```

Importar todo desde un módulo

Agrupa todas las exportaciones en un solo objeto.

```
import * as ops from "./operaciones.js";

ops.suma(3, 2);
ops.resta(5, 1);
```

Módulos en el navegador

Es necesario indicar que el script es un módulo.

```
<script type="module" src="app.js"></script>
```

Nota: Los módulos usan `strict mode` por defecto.

Módulos en Node.js

JavaScript soporta dos sistemas:

- CommonJS: `require()`
- ES Modules: `import / export`

Módulos en Node.js

JavaScript soporta dos sistemas:

- CommonJS: `require()`
- ES Modules: `import / export`

Recomendado: ES Modules modernos

Buenas prácticas

- Un módulo por responsabilidad
- Usar nombres descriptivos
- Evitar dependencias circulares
- Mantener rutas claras

Consumo de APIs con JavaScript

¿Qué es una API?

Una **API (Application Programming Interface)** permite la comunicación entre aplicaciones.

- Provee datos o servicios
- Usa formatos estándar
- Facilita la integración de sistemas

APIs en el desarrollo web

Las APIs permiten:

- Obtener datos externos
- Enviar información a servidores
- Construir aplicaciones dinámicas

Ejemplos: clima, mapas, autenticación, pagos.

API REST

REST es un estilo de arquitectura basado en HTTP.

- Cliente–Servidor
- Sin estado (stateless)
- Uso de métodos HTTP

Métodos HTTP

- GET – Obtener datos
- POST – Enviar datos
- PUT – Actualizar datos
- DELETE – Eliminar datos

Formato JSON

JSON es el formato más usado en APIs web.

- Ligero y legible
- Basado en pares clave–valor
- Fácil de convertir en objetos JavaScript

JSON y JavaScript

- `JSON.parse()` – JSON a objeto
- `JSON.stringify()` – Objeto a JSON

Programación asíncrona

El consumo de APIs es asíncrono.

- No bloquea la ejecución
- Usa promesas o `async/await`

Promesas en JavaScript

Una promesa representa un valor futuro.

- Pendiente
- Resuelta
- Rechazada

fetch()

`fetch()` es la forma moderna de consumir APIs.

- Basado en promesas
- Nativo en navegadores modernos

fetch() con GET

```
fetch("https://api.ejemplo.com/datos")
  .then(res => res.json())
  .then(data => console.log(data));
```

fetch() con POST

```
fetch(url, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify({ nombre: "Ana" })
});
```

async / await

Sintaxis más clara para código asíncrono.

```
async function obtenerDatos() {
  const res = await fetch(url);
  const data = await res.json();
}
```

Manejo de errores

- Errores de red
- Respuestas no válidas
- Estados HTTP (404, 500)

Códigos de estado HTTP

- 200 – OK
- 201 – Created
- 400 – Bad Request
- 401 – Unauthorized
- 404 – Not Found
- 500 – Server Error

Autenticación en APIs

Algunas APIs requieren credenciales.

- API Keys
- Tokens (JWT)
- Headers de autorización

Headers HTTP

Los headers envían información adicional.

- Content-Type
- Authorization
- Accept

CORS

CORS controla el acceso entre dominios.

- Política de seguridad del navegador
- Configuración del servidor

Integración API y DOM

- Obtener datos de la API
- Procesarlos en JavaScript
- Mostrar resultados en el DOM

Errores comunes

- No usar `await`
- Olvidar convertir a JSON
- No manejar errores

Buenas prácticas al consumir APIs

- Separar lógica de red y DOM
- Manejar errores siempre
- No exponer claves privadas
- Documentar endpoints

Principales librerías de JavaScript

¿Qué es una librería JavaScript?

Una librería es un conjunto de funciones reutilizables que facilitan tareas comunes.

- No impone estructura
- Se usa solo lo necesario
- Reduce código repetitivo

Librerías vs Frameworks

- **Librería:** tú controlas el flujo del programa
- **Framework:** el framework controla el flujo

Ejemplo: jQuery (librería) vs Angular (framework)

jQuery

Librería clásica para manipulación del DOM.

- Simplifica selección y eventos
- Compatible con navegadores antiguos
- Uso en proyectos legacy (o proyecto heredado)

Axios

Librería para consumo de APIs HTTP.

- Basada en promesas
- Manejo automático de JSON
- Soporte para interceptores

Lodash

Librería de utilidades para JavaScript.

- Manipulación de arreglos y objetos
- Funciones funcionales
- Optimización de código

Chart.js

Librería para visualización de datos.

- Gráficas simples y atractivas
- Fácil integración con canvas
- Ideal para dashboards

Moment.js / Day.js

Librerías para manejo de fechas y horas.

- Formato de fechas
- Operaciones de tiempo
- Day.js como alternativa ligera

Anime.js

Librería para animaciones.

- Animaciones fluidas
- Control preciso del tiempo
- Soporte CSS, SVG y JS

Jest

Librería para pruebas en JavaScript.

- Pruebas unitarias
- Rápida configuración
- Uso común con React y Node.js

¿Cuándo usar librerías?

- Para acelerar el desarrollo
- Cuando resuelven un problema específico
- Evitar reinventar la rueda

Referencias i

J. Román.

Lenguaje javascript.

`https://lenguajejs.com/`, 2008.

[Online; accessed 5-October-2024].