Guía rápida de fundamentos en JavaScript

Con más de 200 términos

mouredev pro

mouredev.pro/recursos

Índice

Introducción

Esta es una guía de referencia que hace un recorrido por los principales fundamentos del lenguaje de programación JavaScript.

Puedes utilizarla para conocer rápidamente las características de JavaScript, apoyar tu aprendizaje y obtener información sobre más de 200 términos de manera rápida y sencilla, consultando nombres, definiciones y sintaxis.

Tipos de datos básicos

Los **tipos de datos** son la base de cualquier programa, ya que determinan cómo se almacenan y manipulan los valores. Existen tipos primitivos como números, cadenas de texto, booleanos, null, undefined, símbolos y BigInt, y estructuras de datos más avanzadas como arrays, conjuntos, mapas y objetos. Conocerlos es fundamental para escribir código eficiente y bien estructurado.

Nombre	Palabra reservada	Sintaxis
Primitivos		
Número	`Number`	`42, 3.14, -0.5`
Cadena de texto	`String`	`"Hola"`
Booleano	`Boolean`	`true/false`
Nulo	`null`	`null`
Indefinido	`undefined`	`undefined`
Símbolo	`Symbol`	`Symbol("id")`
BigInt	`BigInt`	`9007199254740993n`
Estructuras		
Array	`Array`	`[1, 2, 3, 4]`
Conjunto	`Set`	`new Set([1, 2, 3])`
Mapa	`Map`	`new Map([["clave", "valor"]])`
Objeto	`Object`	`{clave1: "valor1", clave2: "valor2"}`
Conjunto débil	`WeakSet`	`new WeakSet([obj1, obj2])`
Mapa débil	`WeakMap`	`new WeakMap([[obj1, valor1]])`

Sintaxis básica (parte 1 de 2)

La **sintaxis** de JavaScript, aunque ha evolucionado, siempre se ha mantenido sencilla y cercana al lenguaje humano en sus palabras clave. Aquí encontrarás las estructuras fundamentales del lenguaje, como variables y constantes, condicionales, bucles, funciones y clases, junto con su respectiva sintaxis para escribir código claro y organizado.

Nombre	Palabra reservada	Sintaxis
Variable global	`var`	`var a`
Variable bloque	`let`	`let a`
Constante	`const`	`const a`
Condicional if	`if`	`if (condición) { }`
Condicional else if	`else if`	`if (otra_condición) { }`
Condicional else	`else`	`else { }`
Bucle for	`for`	`for (init; cond; incremento) { }`
Bucle for of	`for…of`	`for (const item of iterable) { }`
Bucle for in	`for…in`	`for (const item in objeto) { }`
Bucle while	`while`	`do (condición) { }`
Bucle do while	`do…while`	`do { } while (condición)`
Sentencia break	`break`	`break`
Sentencia continue	`continue`	`continue`
Funcion	`function`	`function nombre(parámetros) { }`
Retorno función	`return`	`return valor`

Sintaxis básica (parte 2 de 2)

Nombre	Palabra reservada	Sintaxis
Clase	`class`	`class Nombre { }`
Excepciones	`try, catch`	`try { } catch (error) { }`
Bloque finally	`finally`	`try { } catch { } finally { }`
Lanzar excepción	`throw`	`throw new Error`
Comprobar tipo	`typeof`	`typeof valor`
Comprobar instancia	`instanceof`	`valor instanceof tipo_dato`
Módulo import	`import, from`	`import nombre from módulo`
Módulo export	`export`	`export function/class/const…`

Operadores (parte 1 de 3)

Los **operadores** son símbolos que permiten realizar cálculos y comparaciones en JavaScript. Se dividen en varias categorías: aritméticos (suma, resta, multiplicación), de comparación (mayor, menor, igual), lógicos (and, or, not), de asignación (=, +=), bitwise y otros especializados como el operador ternario o de propagación.

Nombre	Representación	Sintaxis
Aritméticos		
Suma	+	`a + b`
Resta	-	`a - b`
Multiplicación	*	`a * b`
División	/	`a / b`
División entera	//	`a // b`
Módulo (residuo)	%	`a % b`
Exponenciación	**	`a ** b`
Comparación		
Igualdad	==	`a == b`
Igualdad estricta	===	`a === b`
Desigualdad	!=	`a != b`
Desigualdad estricta	!==	`a !== b`
Mayor que	>	`a > b`
Menor que	<	`a < b`
Mayor o igual que	>=	`a >= b`
Menor o igual que	<=	`a <= b`

Operadores (parte 2 de 3)

Nombre	Representación	Sintaxis
Lógicos		
AND	`&&`	`a && b`
OR	`||`	`a || b`
NOT	`!`	`!a`
Asignación		
Asignación	`=`	`x = 5`
A. con suma	`+=`	`x += 3`
A. con resta	`-=`	`x -= 3`
A. con multiplicación	`*=`	`x *= 3`
A. con división	`/=`	`x /= 3`
A. con módulo	`%=`	`x %= 3`
A. con exponenciación	`**=`	`x **= 3`
A. con bitwise A	`&=`	`x &= 3`
A. con bitwise OR	`|=`	`x |= 3`
A. con bitwise XOR	`^=`	`x ^= 3`
A. con desplaz. izq.	`<<=`	`a <<= 3`
A. con desplaz. der.	`>>=`	`a >>= 3`
A. sin signo	`>>>=`	`a >>>= 3`

Operadores (parte 3 de 3)

Nombre	Representación	Sintaxis
Identidad		
Es	`===`	`a === a`
No es	`!==`	`a !== b`
Pertenencia		
Pertenece	`in`	`propiedad in objeto`
Bitwise		
AND bit a bit	`&`	`a & b`
OR bit a bit	`|`	`a | b`
XOR bit a bit	`^`	`a ^ b`
NOT bit a bit	`~`	`~a`
Desplazamiento a la izq.	`<<`	`a << n`
Desplazamiento a la der.	`>>`	`a >> n`
Desp. a la der. sin signo	`>>>`	`a >>> n`
Ternario		
Condicional ternario	`? :`	`cond ? true : false`
Propagación		
Spread	`...`	`...valor`

Funciones (parte 1 de 4)

Las **funciones** permiten encapsular y reutilizar código, haciendo que los programas sean más modulares y organizados. En JavaScript, las funciones son ciudadanos de primera clase: pueden asignarse a variables, pasarse como argumentos y retornarse desde otras funciones. JavaScript ofrece una gran variedad de funciones integradas en diferentes niveles. Aquí tienes las funciones más comunes que facilitan el desarrollo en el lenguaje.

Nombre	Operación
`console.log()`	Imprime mensajes en la consola para depuración
`document.getElementById()`	Obtiene un elemento del DOM por su ID
`document.querySelector()`	Selecciona el primer elemento que coincide con un selector CSS
`document.querySelectorAll()`	Selecciona todos los elementos que coinciden con un selector CSS
`addEventListener()`	Añade un manejador de eventos a un elemento
`removeEventListener()`	Elimina un manejador de eventos de un elemento
`setTimeout()`	Ejecuta una función tras un retraso especificado
`clearTimeout()`	Cancela el timeout
`setInterval()`	Ejecuta una función de forma repetida en intervalos de tiempo
`clearInterval()`	Cancela una ejecución repetida establecida con el intervalo
`JSON.parse()`	Convierte una cadena JSON en un objeto
`JSON.stringify()`	Convierte un objeto en una cadena JSON

Funciones (parte 2 de 4)

Nombre	Operación
`Math.random()`	Genera un número aleatorio entre 0 y 1
`Math.floor()`	Redondea un número hacia abajo
`Math.ceil()`	Redondea un número hacia arriba
`Math.round()`	Redondea un número al entero más cercano
`Math.max()`	Devuelve el valor máximo de entre los argumentos
`Math.min()`	Devuelve el valor mínimo de entre los argumentos
`parseInt()`	Convierte una cadena a un entero
`parseFloat()`	Convierte una cadena a un número decimal
`isNaN()`	Determina si un valor es NaN (no es un número)
`isFinite()`	Determina si un valor es finito
`Number.isInteger()`	Verifica si un valor es un entero
`Date.now()`	Devuelve la marca de tiempo actual en milisegundos
`Date.parse()`	Convierte una cadena de fecha en milisegundos
`String / charAt()`	Devuelve el caracter en una posición específica de una cadena
`String / concat()`	Une dos o más cadenas

Funciones (parte 3 de 4)

Nombre	Operación
`String / includes()`	Comprueba si una cadena contiene otra subcadena
`String / indexOf()`	Devuelve la posición de la primera aparición de una subcadena
`String / replace()`	Reemplaza parte de una cadena por otr
`String / slice()`	Extrae una parte de una cadena
`String / split()`	Divide una cadena en un array de subcadenas
`String / toLowerCase()`	Convierte una cadena a minúsculas
`String / toUpperCase()`	Convierte una cadena a mayúsculas
`Object.keys()`	Devuelve un array con las claves de un objeto
`Object.values()`	Devuelve un array con los valores de un objeto
`Object.entries()`	Devuelve un array de pares [clave, valor] de un objeto
`Object.assign()`	Copia las propiedades de uno o más objetos a un objeto destino
`Object.create()`	Crea un nuevo objeto con el prototipo especificado
`Object.freeze()`	Congela un objeto, impidiendo modificar sus propiedades

Funciones (parte 4 de 4)

Nombre	Operación
`Function / bind()`	Crea una nueva función con un this vinculado
`Function / call()`	Invoca una función con un this y argumentos especificados
`Function / apply()`	Invoca una función con un this y un array de argumentos
`fetch()`	Realiza peticiones HTTP de forma asíncrona
`encodeURI()`	Codifica una URI completa
`decodeURI()`	Decodifica una URI codificada
`encodeURIComponent()`	Codifica un componente de una URI
`decodeURIComponent()`	Decodifica un componente de una URI
`alert()`	Muestra una ventana de alerta con un mensaje
`prompt()`	Muestra un cuadro de diálogo que solicita una entrada al usuario
`confirm()`	Muestra un cuadro de diálogo de confirmación al usuario

Estructuras de datos (parte 1 de 3)

Los arrays, conjuntos, mapas y objetos son **estructuras de datos** clave en JavaScript, y cada una tiene operaciones específicas que permiten manipular la información de manera eficiente. Esta sección cubre métodos esenciales para agregar, eliminar, buscar, ordenar y transformar datos en estas estructuras.

Nombre	Operación	Sintaxis
Arrays		
`length`	Indica la cantidad de elementos	`array.length`
`push()`	Agrega un elemento al final	`array.push(valor)`
`pop()`	Elimina y retorna el último el.	`array.pop()`
`shift()`	Elimina y retorna el primer el.	`array.shift()`
`unshift()`	Agrega un elemento al inicio	`array.unshift(valor)`
`indexOf()`	Busca el índice de la ocurrencia	`array.indexOf(valor)`
`includes()`	True si el valor está presente	`array.includes(valor)`
`slice()`	Crea una porción	`array.slice(inicio, fin)`
`join()`	Une elementos en una cadena	`array.join(conector)`
`reverse()`	Invierte el orden de los el.	`array.reverse()`
`forEach()`	Ejecuta una función callback para cada elemento	`array.forEach(callback)`
`map()`	Crea un array con los elementos del callback	`array.map(callback)`
`filter()`	Crea un array con los elementos del callback que retornan true	`array.filter(callback)`
`find()`	Retorna el primer elemento que cumple la condición	`array.find(callback)`
`reduce()`	Retorna el valor acumulado	`array.reduce(callback)`

Estructuras de datos (parte 2 de 3)

Nombre	Operación	Sintaxis
Conjuntos (set)		
`size`	Retorna en número de elementos	`set.size`
`add()`	Agrega un valor al conjunto	`set.add(valor)`
`has()`	Comprueba si existe un valor	`set.has(valor)`
`delete()`	Elimina un valor	`set.delete(valor)`
`clear()`	Elimina todos los elementos	`set.clear()`
`keys()`	Devuelve iteradores de los valores	`set.keys()`
`values()`	Devuelve iteradores de los valores	`set.values()`
`entries()`	Devuelve iteradores de pares	`set.entries()`
Mapas (map)		
`size`	Retorna en número de pares	`map.size`
`set()`	Asigna un valor a la clave	`map.set(clave, valor)`
`get()`	Retorna el valor de la clave	`map.get(clave)`
`has()`	Comprueba si existe la clave	`map.has(clave)`
`delete()`	Elimina el par de la clave	`map.delete(clave)`
`clear()`	Elimina todos los pares	`map.clear()`
`keys()`	Devuelve iteradores de las claves	`map.keys()`
`values()`	Devuelve iteradores de los valores	`map.values()`
`entries()`	Devuelve iteradores de pares [clave, valor]	`map.entries()`

Estructuras de datos (parte 3 de 3)

Nombre	Operación	Sintaxis
Objetos		
`Acceder (punto)`	Accede al valor	`obj.propiedad`
`Acceder (corchetes)`	Accede al valor	`obj[propiedad]`
`Agregar`	Añade una propiedad	`obj.propiedad = valor`
`Modificar`	Modifica una propiedad	`obj.propiedad = valor`
`delete`	Elimina la propiedad	`delete obj.propiedad`
`in`	Comprueba existencia	`propiedad in obj`
`hasOwnPropery()`	Comprueba existencia	`obj.hasOwnPropery(prop)`
`keys()`	Devuelve iteradores de las claves	`obj.keys()`
`values()`	Devuelve iteradores de los valores	`obj.values()`
`entries()`	Devuelve iteradores de pares [clave, valor]	`obj.entries()`
`assign`	Copia propiedades	`Object.assign(destino, obj_fuente)`
`Propagación`	Copia propiedades	`{...obj1, ...obj2}`

Manejo de archivos (parte 1 de 3)

El **manejo de archivos** en JavaScript del lado del navegador es diferente a otros lenguajes de programación, ya que por razones de seguridad no se permite acceder libremente al sistema de archivos del cliente. Fuera del navegador (por ejemplo, desde Node.js) podemos leer, escribir, modificar y eliminar archivos de manera sencilla, así como manipular directorios o trabajar con formatos específicos propios de sistemas operativos de escritorio.

Nombre	Operación	Sintaxis
Navegador (File API)		
`File`	Input en HTML	`<input type="file">`

`FileReader`	Lee el contenido del archivo	`readAsText(fichero, encoding)`
		`readAsDataURL(fichero)`
		`readAsArrayBuffer(fichero)`

`Blob`	Datos binarios	`new Blob`
Navegador (Web Storage)		

`localStorage`	Persistente en el navegador	`setItem(clave, valor)`
		`getItem(clave)`
		`removeItem(clave)`
		`clear()`

`sessionStorage`	Sesión del navegador (temporal)	`setItem(clave, valor)`
		`getItem(clave)`
		`removeItem(clave)`
		`clear()`

Manejo de archivos (parte 2 de 3)

El **manejo de archivos** desde fuera del navegador puede requerir de algún módulo externo. En este caso, desde Node.js se utiliza fs (File System):
`const fs = require("fs")`

Modos de apertura en `open()`:

- `"r"` → Lectura (por defecto)
- `"w"` → Escritura (borra contenido previo)
- `"a"` → Agregar contenido al final
- `"r+"` → Lectura y escritura

Nombre	Representación	Sintaxis
Node.js (File System)		
`readFile()`	Lee un archivo	`fs.readFile("a.txt", "utf8", (err, data) => { })`
`readFileSync()`	Lee un archivo de forma síncrona	`const datos = fs.readFileSync("a.txt", "utf8")`
`writeFile()`	Escribe un archivo	`fs.writeFile("a.txt", data, "utf8", (err) => { })`
`writeFileSync()`	Escribe un archivo de forma síncrona	`fs.writeFileSync("a.txt", data, "utf8")`
`appendFile()`	Añade contenido	`fs.appendFile("a.txt", data, "utf8", (err) => { })`
`appendFileSync()`	Añade contenido de forma síncrona	`fs.appendFileSync("a.txt", data, "utf8")`
`rename()`	Renombra un archivo	`fs.rename("old.txt", "new.txt", (err) => { })`
`renameSync()`	Renombra un archivo de forma síncrona	`fs.renameSync("old.txt", "new.txt")`

Manejo de archivos (parte 3 de 3)

Nombre	Representación	Sintaxis
`unlink()`	Elimina un archivo	`fs.unlink("a.txt", (err) => { })`
`unlinkSync()`	Elimina un archivo de forma síncrona	`fs.unlinkSync("a.txt")`
`copyFile()`	Copia un archivo	`fs.copyFile("origen.txt", "destino.txt", (err) => { /* ... */ })`
`readdir()`	Lista el contenido de un directorio	`fs.readdir("./", (err, files) => { })`
`mkdir()`	Crea un directorio	`fs.mkdir("nuevoDir", { recursive: true }, (err) => { })`
`mkdirSync()`	Crea un directorio de forma síncrona	`fs.mkdirSync("nuevoDir", { recursive: true })`
`stat()`	Estadísticas de un archivo	`fs.stat("a.txt", (err, stats) => { })`
`statSync()`	Estadísticas de un archivo síncrono	`const estadísticas = fs.statSync("a.txt")`
`createReadStream()`	Crea un stream de lectura	`const stream = fs.createReadStream( "a.txt", { encoding: "utf8" })`
`createWriteStream()`	Crea un stream de escritura	`const stream = fs.createWriteStream( "a.txt", { encoding: "utf8" })`
`fs/promises API`	Operaciones con promesas y async/await	`const fs = require("fs/promises") await fs.readFile( "a.txt", "utf8")`

Módulos estándar

JavaScript cuenta con una **biblioteca estándar** que incluye **objetos**, **clases** y **módulos** integrados listos para ser usados sin instalación adicional. Estos módulos facilitan tareas como manipulación de archivos (`fs`), operaciones matemáticas (`Math`), manejo de fechas (`Date`), y muchas más. Conocer estos módulos ayuda a optimizar el desarrollo sin necesidad de librerías externas.

Nombre	Descripción
Estándar ECMAScript	
`Math`	Funciones y constantes matemáticas
`Date`	Manejo y formateo de fechas y horas
`RegExp`	Definición y uso de expresiones regulares
`Promise`	Manejo de operaciones asíncronas
`JSON`	Parseo y generación de datos en formato JSON
`Intl`	Internacionalización (formatos numéricos, fechas, etc)
Módulos Node.js (requieren importación)	
`os`	Información y utilidades del sistema operativo
`fs`	Manejo del sistema de archivos
`path`	Utilidad para trabajar con rutas de archivos y directorios
`http/https`	Creación de servidores y clientes HTTP/HTTPS
`events`	Implementa un emisor para manejar eventos
`util`	Funciones con diferentes utilidades
`stream`	Manejo de streams de datos
`url`	Parseo y manipulación de URLs
`crypto`	Funciones de criptografía y generación de hash

Módulos externos (parte 1 de 2)

Además de la biblioteca estándar, JavaScript permite instalar **módulos externos** para ampliar sus capacidades. Algunas de las bibliotecas y frameworks más populares incluyen `Express` para peticiones web, `React` para crear interfaces gráficas, `Three.js` para gráficos, o `Jest` para testing. Estas son algunas de las muchísimas librerías de terceros más utilizadas.

Nombre	Descripción
`React`	Biblioteca para construir interfaces de usuario de forma declarativa y con componentes reutilizables
`Angular`	Framework robusto para desarrollar aplicaciones web de gran escala
`Vue`	Framework progresivo para crear interfaces y aplicaciones web reactivas
`Express`	Framework minimalista para crear APIs y aplicaciones web en Node.js
`Node`	Entorno de ejecución de JavaScript del lado del servidor basado en el motor V8 de Chrome
`Next`	Framework basado en React con renderizado del lado del servidor y generación de sitios estáticos
`Nuxt`	Framework basado en Vue.js para aplicaciones universales y sitios estáticos
`Svelte`	Framework que compila componentes en código altamente optimizado para interfaces reactivas
`jQuery`	Biblioteca clásica que simplifica la manipulación del DOM y eventos en el navegador
`Redux`	Solución para la gestión centralizada y predecible del estado en aplicaciones, especialmente en React
`Astro`	Framework para sitios centrados en contenido con renderizado parcial en el cliente

Módulos externos (parte 2 de 2)

Nombre	Descripción
`Lodash`	Conjunto de utilidades para trabajar con arrays, objetos y funciones de forma sencilla
`Moment`	Herramienta para el manejo y formateo de fechas y horas
`RxJS`	Biblioteca para programación reactiva mediante observables
`Chart`	Biblioteca sencilla para generar grÃ¡ficos y visualizaciones interactivas
`Three`	Biblioteca para crear gráficos 3D en el navegador utilizando WebGL
`Electron`	Framework para crear aplicaciones de escritorio multiplataforma con tecnologías web
`Jest`	Framework de pruebas con enfoque en simplicidad que incluye assertions, mocks y cobertura de código
`Mocha`	Framework de testing flexible para ejecutar pruebas unitarias y de integración
`Webpack`	Empaquetador de módulos que optimiza y organiza recursos en proyectos
`Deno / Bun`	Entornos de ejecución
`Socket.IO`	Biblioteca para aplicaciones web en tiempo real con WebSockets

Es recomendable utilizar un gestor de dependencias como **npm** para su instalación. Todos estos módulos pueden instalarse con `npm install nombre_del_modulo`.

¿Quieres aprender más sobre JavaScript?

Aquí tienes mis cursos para aprender JavaScript desde cero.

Curso gratis en YouTube:
https://mouredev.link/javascript

Curso con extras (lecciones por tema, ejercicios, soporte, comunidad, test y certificado) en el campus de estudiantes mouredev pro:
https://mouredev.pro/cursos/javascript-desde-cero

(Utiliza el cupón "PRO" para acceder con un 10% de descuento a todas las suscripciones y cursos del campus).

mouredev pro

Estudia programación y desarrollo de software de manera diferente

mouredev.pro