99 TIPS

Para Web Development

Freddy Montes, Junio 2021

99 tips para Web Development

HTML, CSS y JavaScript

Freddy Montes (@fmontes)

Junio 2021

Contenido

Introducción

Con los cientos de frameworks de JavaScript que existen, y los que faltan, he observado que se están implementando soluciones muy complicadas para problemas del día a día que se pueden resolver con HTML, CSS o JavaScript puro aprovechando bien las capacidades nativas de los navegadores

Este libro **recopila 99 soluciones a problemas de desarrollo web** que se pueden resolver de manera nativa, fácil, pero más importante, eficientemente.

Para más contenido sobre diseño y desarrollo web o sugerencias estoy en Twitter: https://twitter.com/fmontes y en Instagram https://instagram.com/fmontes.

HTML

Asociar labels e inputs programáticamente

El atributo `for` en el `<label>` te permite asociar programáticamente el label con su respectivo elemento `input`.

```
<input type="checkbox" name="ps5" id="ps5">
<label for="ps5">¿Quieres un PS5?</label>
```

Beneficios

1. El usuario puede hacer click en el label para "activar" el `input` esto es bueno porque haces el "hit area" más grande.
2. En accesibilidad va a permitir que el screen reader lea el texto del `label` cuando el `input` esté focus o seleccionado.

Enfocar un elemento cuando carga la página

Es un atributo global que te permite indicarle al navegador cuál elemento debe estar enfocado cuando la página carga.

```
<input type="text" autofocus />
```

Beneficios

1. Indicar al usuario donde comenzar a realizar la tarea en la página. Por ejemplo en una página de login puedes hacer "focus" al `input` del username.
2. Puedes ahorrarle tiempo al usuario o salvar un click.

Pedir al navegador que descargue un archivo

Se usa en la etiqueta `<a>` y "pide" al usuario que guarde el archivo de la URL en lugar de navegar hasta ella.

```
<a href="file.pdf" download>Descargar</a>
```

Beneficios

1. Los navegadores modernos pueden leer PDF por lo que al usar el atributo download fuerzas al usuario que descargue el archivo en lugar de abrirlo en el navegador.
2. Permite controlar lo que quieres hacer con el archivo del link.

¿Cómo hacer tu página web editable?

Con el atributo `contenteditable` puedes hacer cualquier elemento de tu página editable.

Y puedes escuchar el evento `input` obtener el valor.

```
<p contenteditable="true">
    Este contenido es editable
</p>

<script type="application/javascript">
    document.querySelector('p')
        .addEventListener('input', (e) => {
            console.log(e)
        })
</script>
```

Si necesitas hacer una página completa editable o incluso un elemento en específico lo puedes hacer con este atributo y con esto crear una herramienta #nocode

¿Cómo escribir direcciones correctamente?

La etiqueta `<address>` indica que el HTML dentro de ella proporciona información de contacto para una persona u organización.

```
<address>
    <a href="mailto:hi@midominio.com">
        hi@midominio.com
    </a>
    <br />
  <a href="tel:+1234567890">
        (123) 456-7890
    </a>
</address>
```

Semántica, es la etiqueta correcta para mostrar una dirección.

La especificación fue actualizada y se puede usar para mostrar una dirección arbitraria y no solo la del autor de la página.

Un elemento para mostrar fecha y hora

La manera correcta de mostrar "hora" o "tiempo" en una página es usando el elemento `<time>`.

```
<p>
    Oasis va a celebrar su 25 aniversario
    <time datetime="2021-07-07">el 7 de julio</time>
    en el Hyde Park de Londres.
</p>

<p>
    El concierto comienza a las
    <time datetime="20:00">08:00pm</time>
</p>
```

Debes incluir el atributo `datetime` para traducir la hora a un formato **legible por la máquina**, lo que permite obtener mejores resultados en los motores de búsqueda o funciones personalizadas como los recordatorios.

Crear links para hacer llamadas o enviar SMS o correos

Si queremos que un usuario nos contacte tenemos que ponérsela fácil, para eso podemos crear links para los diferentes métodos de contacto.

```
<a href="mailto:email@empresa.com?subject=titulo">
  Enviar un email
</a>

<a href="tel:+50612312312">
  Llamanos
</a>

<a href="sms:+50612312312?body=contenido">
  Send us a message
</a>
```

Incluso puedes agregar contenido al correo o al sms como el título o el cuerpo.

Agregar “citas” a un documento

Para indicar que un texto pertenece a otro autor se hace una cita con el elemento con `<blockquote>`.

Puedes usar el atributo “cite” para pasar el link de la fuente.

```
<blockquote cite="https://pagina.com/articulo.html">
  <p>Este texto es algo que dijo alguien más.</p>
</blockquote>
```

Indicar semánticamente correcta las citas en un documento.

Crear un acordeón con solo HTML

Crea un component tipo acordeón en el que la información completa es visible solo cuando le dan click y se cambia a un estado "expandido". Puedes pasar el texto al estado "no expandido" mediante el elemento `<summary>`.

```
<details>
    <summary>Título</summary>
    Este es el contenido que se muestra
</details>
```

1. Cero JavaScript.
2. Fácil de hacer los estilos.

Especificar el url base de tu página web

En la etiqueta `<base>` con el atributo href indicas cuál es el url base para todos los url relativos de tu documento.

```
<head>
    <base href="https://ejemplo.com/">
</heead>

<body>
    <a href="prueba.html">Click aqui</a>
</body>
<!--
Este resuelve a https://ejemplo.com/prueba.html
-->
```

Puedes controlar al detalle los URL relativos.

¿Cómo deshabilitar el botón derecho?

Esto es una malísima práctica de accesibilidad y la experiencia de usuario.

```
<body oncontextmenu="return false">
```

Si estás desarrollando una aplicación donde necesitas un comportamiento personalizado para el botón derecho puedes usar esta solución.

Usar patrones específicos de validación en elementos `<input>`

El atributo de “pattern” especifica una expresión regular con la que debe coincidir el valor del input.

Por ejemplo: el siguiente input solo puede tener una cadena de texto de 4 a 8 caracteres todos en minúscula para ser válido

```
<input type="text" pattern="[a-z]{4,8}" required />
```

No usar JavaScript para validar la entrada de datos de un usuario

Invertir los números en una lista

Las listas ordenadas `<ol>` por defecto se ordenan de manera descendente, pero puedes cambiar ese orden con el atributo `reversed`.

```
<ol reversed>
    <li>Primero</li>
    <li>Segunda</li>
    <li>Tercero</li>
</ol>
```

Puedes hacer un conteo regresivo fácilmente.

Crear un selector con autocomplete con puro HTML

Él autocomplete es un patrón de diseño que le facilita al usuario filtrar elementos usando un string de texto para facilitar la selección.

```
<label>
    ¿Qué quieres comer?
    <input list="elementos" />
</label>
<datalist id="elementos">
    <option>Hamburguesa</option>
    <option>Pizza</option>
    <option>Tacos</option>
    <option>Arepas</option>
</datalist>
```

Mejora la experiencia de usuario cuando tienes selectores con muchísimas opciones, ya que le permites filtrar para seleccionar opciones.

¿Cómo usar un selector de colores?

El elemento `input` es mucho más que solo un campo de texto, puedes tener un selector de colores.

```
<input type="color" />
```

Esto funciona como un `input` normal y puedes acceder al valor y escuchar eventos.

Cambiando el color del tema del navegador

Para cambiar los colores del navegador se utiliza la etiqueta `<meta>` con el atributo `theme-color` de la siguiente manera:

```
<meta name="theme-color" content="#4285f4">
```

Esto cambia los colores principales de la interfaz del navegador en dispositivos móviles.

Para cuando escribí este libro junio 2021 solo funcionaba en Android, pero ya estaba anunciado el soporte para iOS 15.

Cargar imágenes con “lazy load”

Lazy load es una estrategia que se refiere a cargar recursos cuando sea requerido por el usuario, esto nos permite ahorrar recursos.

Las imágenes son muy comunes en la web, pero son también responsable de problemas de performance, es por esto que es recomendado cargarlas de manera lazy y para eso se utiliza el atributo `loading`.

```
<img src="path/a/la/imagen.jpg" loading="lazy" />
```

Este atributo le indica al navegador que solo cargue la imagen cuando esté cerca de mostrarse en el viewport.

Cargando imágenes para cada media query

En la web moderna mostrar una imagen ya no es tan simple como usar un `img` tag con las diferentes resoluciones y pantallas que hay que tomar en cuenta.

Para eso se creó el elemento `<picture>` que junto a `<source>` te permite cargar diferentes imágenes dependiendo de un media query:

```
<picture>
    <source
            srcset="/path/a/la/imagen-800.jpg"
            media="(min-width: 800px)">
    <img src="/path/a/la/imagen-320.jpg" />
</picture>
```

`picture` puede contener cero o varios `source` y es aquí donde le pasamos la imagen en el atributo `srcset` y él media query donde queremos que esta imagen se muestre. Al final le pasamos un `img` element normal para cuando ningún media query haga match.

Esto te va a permitir tener imágenes de diferentes resoluciones y tamaño para cada tipo de pantalla.

Hacer una barra de progreso

Con el elemento `<progress>` puedes mostrar **un indicador del progreso de una tarea**, normalmente en forma de barra de progreso.

```
<progress max="100" value="70">70%</progress>
```

Con los atributos `max` defines el máximo valor que puede tener la barra y con el value cuál es el valor actual.

¿Cómo usar el elemento `<dialog>`?

Representa un cuadro de diálogo u otro componente interactivo, como una alerta, un inspector o una subventana. Normalmente este elemento se creaba con etiquetas tradicionales de HTML como `div` y `span`, pero al ser tan popular se creó el elemento.

```
<dialog open>
    <h3>Titulo del dialog</h3>
    <p>Contenido del dialog</p>
</dialog>
```

Al pasarle el atributo `open` el dialog aparecería "abierto" pero si necesitas abrirse dinámicamente hay que usar JavaScript:

```
const dialog = document.querySelector('dialog');
dialog.showModal();
```

El método `showModal` muestra el dialog en la pagina.

Indicarle al navegador el lenguaje de tu documento

Con el atributo `lang` se indica en que idioma está el contenido de un element HTML. Para ponerlo a todo el documento se agrega a la etiqueta `html` al inicio de la página.

```
<html lang="en">
 ...
</html>
```

Es superimportante indicar que lenguaje es tu página web, ya que esto permite a los buscadores mostrar tu página en los resultados de las búsquedas del lenguaje correcto, pero además a los lectores de pantallas de leer el documento en el idioma correcto.

Indicar lenguaje en una sección especifica del documento

El atributo `lang` al ser un atributo global se le puede pasar a cualquier elemento en la página.

```
<html lang="en">
    <body>
        <p>This is english</p>
        <p lang="es">Esto es español</p>
    </body>
</html>
```

Al pasar el atributo a un elemento en particular puedes sobreescribir el lenguaje global e indicar al navegador que el contenido de esa etiqueta es otro lenguaje.

Agregar un corrector ortográfico a un elemento

Existe un atributo global llamado `spellcheck` que define si el elemento puede ser revisado por errores ortográficos.

```
<textarea spellcheck="true">
    Jere is some text widt some eror
</textarea>

<textarea lang="es" spellcheck="true">
    Aki ai un error
</textarea>
```

Lo puedes combinar con el atributo `lang` y funciona de manera que cuando el usuario se enfoca en el elemento subraya las palabras con errores ortográficos.

¿Cómo indicar el tipo de archivo que acepta un `input` file?

Con el atributo `accept` podemos indicarle al `<input type="file" />` que tipo de archivo puede recibir.

```
<!-- Solo imagenes -->
<input type="file" accept="image/*" />

<!-- Solo archivos .pdf -->
<input type="file" accept=".pdf" />
```

El valor que le pasamos al `accept` es un `string` separado por comas con un listado de "unique file type specifier".

Con esto, cuando la ventana de seleccionar archivos del sistema operativo se abra, solo le va a permitir al usuario seleccionar archivos del tipo indicado.

¿Cómo mostrar el teclado numérico para ingresar número de teléfono?

Para simplificar la tarea del usuario de llenar formularios es mejor mostrar el teclado del teléfono adecuado para cada campo, en el caso del teléfono el teclado numérico.

```
<input type="tel" />
```

Al pasarle el `type="tel"` le estás indicando al navegador que el valor que puede tener ese campo es un número de teléfono por lo cual el teclado que va a mostrar es el más adecuado para ingresar ese tipo de dato.

Permitir al usuario escanear tu tarjeta de crédito

Para un usuario meter la información de su tarjeta de crédito manualmente es un dolor, por esto los navegadores móviles implementaron la funcionalidad de escanear la tarjeta con la cámara, pero tienes que indicarlo en los campos con el atributo `autocomplete`.

```
<form>
    <input type="text" autocomplete="cc-name" />
    <input type="text" autocomplete="cc-number" />
    <input type="number" autocomplete="cc-exp-month" />
    <input type="number" autocomplete="cc-exp-year" />
    <input type="text" autocomplete="cc-csc">
</form>
```

Los valores son `cc-name` para el nombre, `cc-number` para el número de tarjeta y así sucesivamente para el resto de información.

Facilitar al usuario el llenado de un formulario ayuda a mejorar la conversión y cuando es un proceso de pago mucho mejor.

¿Qué es el elemento `template`?

Es una plantilla que contiene cierto código HTML que se mantiene oculto cuando la página carga y que posteriormente puedes mostrar con JavaScript.

```
<template>
    <h1>Un titulo</h1>
</template>
```

```
const temp = document.querySelector('template')
const newTemp = temp.content.cloneNode(true);
document.body.appendChild(newTemp);
```

Usar `<template>` te ahorra tener que crear todos los elementos con JavaScript.

¿Qué es el elemento `output`?

Es un elemento contenedor en el que un sitio o una aplicación puede inyectar los resultados de un cálculo o el resultado de una acción del usuario.

```
<form>
  <input type="range" name="b" value="50" /> +
  <input type="number" name="a" value="10" /> =
  <output for="a b">60</output>
</form>
```

Usando el atributo `for` estamos asociando el resultado de la suma de los elementos `a` y `b` al elemento `output`.

Elementos `<input>` de tipo “rango”

Permiten al usuario especificar un valor establecido entre un mínimo y un máximo.

```
<input type="range" min="0" max="100">
```

Es presentado mediante un control deslizante (slider) o dial en lugar de un cuadro de texto de tipo numérica.

Permitir a un usuario seleccionar múltiples archivos

Con el atributo `type="file"` le mostramos al usuario un `input` o elemento para seleccionar archivos, pero de 1 en 1, si queremos que pueda seleccionar más de uno tenemos que agregar otro atributo:

```
<input type="file" multiple />
```

El valor del elemento resulta siendo un objeto de tipo FileList.

¿Cómo reproducir audio?

El elemento HTML `<audio>` se utiliza para incrustar contenido sonoro en los documentos.

```
<!-- Usando el src -->
<audio controls src="/path/al/archivo.mp3">
    Navegador no sporta <code>audio</code>.
</audio>

<!-- Usando el <source> -->
<audio controls>
    <source src="/path/al/audio.mp3" type="audio/mp3" />
    <source src="/path/al/audio.ogg" type="audio/ogg" />
</audio>
```

Puede contener una o varias fuentes de audio, representadas mediante el atributo `src` o el elemento `<source>`: el navegador elegirá la más adecuada.

El atributo `controls` permite mostrar los botones del reproductor como play, pause, volumen, etc.

¿Cómo reproducir video?

El elemento HTML `<video>` inserta un reproductor multimedia que permite la reproducción de vídeo en el documento.

```
<video controls width="250">
  <source src="/path/al/video.webm" type="video/webm" />
  <source src="/path/al/video.mp4" type="video/mp4" />
  Lo sentimos, su navegador no admite vídeos.
</video>
```

Puede contener una o varias fuentes de video en el elemento `<source>`: el navegador elegirá la más adecuada.

El atributo `controls` permite mostrar los botones del reproductor como play, pause, volumen, etc.

¿Cómo cargar archivos de JavaScript y asegurar el rendimiento?

Existe dos atributos que son importantes conocer `defer` y `async`.

```
<script src="app.js" defer></script>
```

El atributo `defer` le dice al navegador que no espere el archivo JavaScript cuando renderice el HTML. Así se renderizará primero el DOM y luego se ejecutará el script.

```
<script src="app.js" async></script>
```

El atributo `async` es similar a `defer` , la única diferencia es que `async` le dice al navegador que cargue y ejecute el `script` de forma asíncrona mientras se carga la UI.

Puedes usar uno u otro dependiendo si necesitas o no que se ejecuten ciertos archivos JavaScript antes de la carga del DOM.

JavaScript

¿Como asignar valores por defecto a las variables?

Usando el operador lógico “OR” con doble pipe || .

```
// OK
let result;

if (value) {
    result = value
} else {
    result = 'fallback value'
}

// ¡Mejor!
let result = value || 'fallback value'
```

Con el operador te ahorras unas líneas de código, pero además queda más conciso.

¿Cómo hacer condiciones más cortas?

El operador ternario `?` permite ejecutar código basado si una condición se cumple. Si es como un `if` pero más conciso.

```
// OK
let comer;

if (pizzaToppings.includes('piña')) {
    comer = 'NO';
} else {
    comer = 'CLARO QUE SI'
}

// ¡Mejor!
let comer = pizzaToppings.includes('piña') ?
    'NO'
    : // los : vienen siendo como el "else"
    'CLARO QUE SI';
```

Código más conciso y fácil de leer.

Mostrar más información al depurar

El `console.log` está bien, pero con `console.trace` puedes ver mucha más información, por ejemplo el stacktrace de la llamada de la función.

```
function hacerAlgo() {
    console.trace('Aqui estoy haciendo algo');
}
```

Puedes ver en detalle todo el camino que llevo a la llamada de una función, lo que te da más contexto de toda la operación.

¿Cómo saber cuanto tiempo tarda un código en ejecutarse?

Si quieres ver la performance de tu código una manera simple es ver cuanto tarda en ejecutarse.

```
console.time('timer-name');

for (let i = 0; i < 10000000; i++) {
  // Hacer algo
}

console.timeEnd('timer-name');
```

Lo que no se mide no se mejora, con esto puedes mejorar la velocidad de tus programas de JavaScript.

¿Cómo extraer valores de un array con "destructuración"?

La desestructuración de arreglos permite "descomprimir" los valores de un array distintas variables.

```
// OK
const array = ['Freddy', 'Montes'];
const firstName = array[0];
const lastName = array[1];

// ¡Mejor, usando "Destructuring"!
const array = ['Freddy', 'Montes'];
const [firstName, lastName] = array;
```

Extraer y crear múltiples variables en la misma línea.

¿Cómo extraer valores de un array con "destructuración" de objetos?

Otra manera de extraer valores de un array es con la sintaxis de objetos.

```
const array = ['Freddy', 'Eduardo', 'Montes', 'Angulo'];
const { 0: nombre, 2: apellido } = array;

console.log(nombre, apellido); // Freddy Montes
```

Con este método puedes extraer elementos de un array y asignarlos a las variables, pero usando enteros para las posiciones, lo que lo hace más legible y mantenible.

¿Cómo extraer propiedades de un objeto?

Al igual que con los array puedes usar destructuración para extraer propiedades del objeto y asignarlo a variables.

```
const objeto = {
    nombre: 'Freddy',
    apellido: 'Montes',
    web: 'fmontes.com'
}

const {nombre, apellido} = objeto;

console.log(nombre); // Freddy
console.log(apellido); // Montes
```

Es una manera más corta extraer elementos de un objeto.

¿Cómo “destructurar” elementos de un objeto en variables existentes?

Si necesitas extraer propiedades de un objeto y guardarlas en variables ya existentes también puedes usar destructuración pero necesitas paréntesis.

```
const objeto = {
    nombre: 'Freddy',
    apellido: 'Montes',
    web: 'fmontes.com'
}

let nombre;
let apellido;

({ nombre, apellido }) = objeto;

console.log(nombre); // Freddy
console.log(apellido); // Montes
```

¿Cómo formatear la salida del método `JSON.stringify`?

El método `JSON.stringify` toma un objeto JSON y lo transforma en un string, pero sin formato.

Si quieres aplicarle un formato tienes que pasar un tercer parámetro de esta manera:

```
const objeto = {
    nombre: 'Freddy',
    apellido: 'Montes',
    web: 'fmontes.com'
}

console.log(JSON.stringify(objeto))
// "{\"nombre\":\"Freddy\",\"apellido\":\"Montes\"  ...

console.log(JSON.stringify(objeto, null, 2))
/*
"{
  \"nombre\": \"Freddy\",
  \"apellido\": \"Montes\",
  \"web\": \"fmontes.com\"
}"
*/
```

El resultado es más fácil de leer y perfecto para copiar y pegar.

¿Cómo hacer los números grandes más legible?

JavaScript provee un separador de numero _ el underscore se usa para separar números muy grandes.

```
const numeroGrande = 1_000_000_000 // 1000000000
```

Con este separador haces tu código mucho más legible.

Eliminar elementos duplicados de un array

Puedes usar el Objeto `Set` en lugar de tener que hacer lo manualmente.

```
const arrayConDuplicados = [
    'Freddy',
    'Montes',
    'Freddy',
    1,
    2,
    1,
    3,
    3
];

const sinDupes = [...new Set(arrayConDuplicados)]

console.log(sinDupes)
// 'Freddy', 'Montes', 1, 2, 3
```

No tener que manualmente eliminar los duplicados del array que te tomaría una función medio larga.

¿Cómo acceder a los media queries con JavaScript?

Los media queries son mayormente usados en CSS, pero también puedes usarlo en JavaScript y observar si el navegador está dentro o fuera de un media queries específico, pero además puedes escuchar eventos.

```
let mql = window.matchMedia('(max-width: 600px)');

if (mql.matches) {
    console.log('El viewport es 600px o menos')
}

mql.addEventListener('change', probarMediaQuery);

function probarMediaQuery(e) {
  if (e.matches) {
        console.log('El viewport es 600px o menos')
  }
}
```

Puedes ejecutar código JavaScript dependiendo del tamaño del viewport, por ejemplo podrías inicializar una librería que solo necesites en pantallas pequeñas.

¿Cómo saber si una función se llamó N cantidad de veces?

Puedes usar `console.count()` que es un método que cuenta la cantidad de veces que un código se ejecuta.

```
function hacerAlgo() {
    console.count('hacerAlgo se llamó')
}

hacerAlgo();
hacerAlgo();
hacerAlgo();

// "hacerAlgo se llamó: 1"
// "hacerAlgo se llamó: 2"
// "hacerAlgo se llamó: 3"
```

¿Cómo verificar si una variable tiene el valor que esperamos?

Si necesitas validar que una variable tenga el valor que esperas normalmente usas un `if`, pero podrías usar `console.assert`.

```
// Normalmente haces algo como
if (!variable) {
    console.log('variable no existe')
}

// Pero puedes hacer
console.assert(variable, 'variable no existe')
```

El método `assert` solo loguea el error cuando la variable tiene un valor falsey.

¿Cómo inspeccionar las propiedades y métodos es un elemento HTML?

Cuando haces `console.log` de un elemento HTML solo ves el HTML, pero si quieres ver las propiedades y métodos debes usar `console.dir`.

```
<p id="someElement">Hola gente</p>
```

```
const element = document.querySelector('#someElement');

console.log(element)
// <p id="someElement">Hola gente</p>

console.dir(element);
// {accessKey: "", align: "",  ... }
```

Agregar propiedades con nombres dinámicos a objetos

Cuando necesitas agregar una propiedad a un objeto con un nombre dinámico puedes usar `[]`.

```
// OK
const dynamic = 'color';
const car = {
    brand: 'Ford'
};

car[dynamic] = 'blue';

// ¡Mejor!
// Puedes agrregar propiedades dinamicas usando []
const dynamic = 'color';
const car = {
    brand: 'Ford',
  [dynamic]: 'blue'
};
```

Creas el objeto y asignas la propiedad con nombre dinamico en una sola operación.

Ejecutar funciones con cortocircuito

El cortocircuito no es más que ejecutar una función si se cumple una condición.

```
// OK
if (hungry) {
  getSomeFruit();
}

// ¡Mejor! y muy usado en React
hungry && getSomeFruit();
```

Código más corto y legible. Muy usado para mostrar components de React.

Filtrar valores `falsey` de un array

Los valores falsey son aquellos que se consideran false en un contexto de boolean, por ejemplo una condición. Por ejemplo `"" || 0 || NaN || undefined || null` .

```
const arr = [
    0,
    'Esto',
    '',
    undefined,
    false,
    'si',
    undefined,
    null,
    ,
    'queda',
    NaN
];
const filtered = arr.filter(Boolean);
console.log(filtered); // [ 'Esto', 'si', 'queda' ]
```

Beneficios

No necesitas pasar toda una función solo el objeto `Boolean`.

Combinar objetos

Usando el spread operator para combinar objetos es más directo que el método anterior de `Object.assign`.

```
const color = {
    color: 'brown'
}

const height = {
    height: '7'
}

const firstName = {
    firstName: 'Freddy'
}

const fullInfo = {...color, ...firstName, ...height};

console.log(fullInfo)
// { color: 'brown', height: '7', firstName: 'Freddy' }
```

Fácil de leer y de entender a primera vista.

Extraer los últimos elementos de un array

Usamos el método `slice` pasándole valores negativos.

```
const array = [0, 1, 2, 3, 4, 5, 6, 7];

console.log(array.slice(-1)); // [7]
console.log(array.slice(-2)); // [6, 7]
console.log(array.slice(-3)); // [5, 6, 7]
```

Un solo método y al usar números negativos hace referencia a "de adelante a atrás".

Parsear cualquier valor

El método `JSON.parse` hace más de lo que el nombre indica, por ejemplo parsear cualquier valor primitivo.

```
JSON.parse('true') // true
JSON.parse('false') // false
JSON.parse(true) // true
JSON.parse(false) // false
JSON.parse('100') // 100
JSON.parse('100.20') // 100.20
JSON.parse(100) // 100
JSON.parse(100.20) // 100.20

JSON.parse('some string') // Uncaught SyntaxError
```

Menos lógica y mismo resultado.

¿Cómo manejar funciones con muchos parámetros?

Existe una convención de que cuando tienes que pasar más de 2 parámetros a una función es mejor usar un objeto.

```
// No hacer esto
function hacerAlgo(nombre, apellido, email) {
    console.log(nombre, apellido, email)
}

// Mejor hacer
function hacerAlgo({ nombre, apellido, email }) {
    console.log(nombre, apellido, email)
}
```

Beneficios:

1. De la primera manera cuando llames a la función tienes que recordar el orden de los parámetros.
2. Mientras que pasando un objeto el orden no importa, pero además puedes usar el spread operator para obtener los valores

¿Cómo eliminar un elemento sin cambiar el objeto original?

Típicamente para eliminar una propiedad de un objeto se usa el keyword `delete` el problema es que esto muta (cambia) el objeto original y si esta no es tu intención puedes terminar con bugs difíciles de rastrear. Es mejor usar spread operator.

```
const miObjeto = {
    nombre: 'Freddy',
    apellido: 'Montes'
}

// Si quiero eliminar el apellido de este objeto
const {apellido, ...nuevoObjeto} = miObjeto;

console.log(nuevoObjeto)
// { nombre: 'Freddy' }
```

¿Cómo mostrar los elementos de un array de objetos como una tabla en la consola?

Cuando estás trabajando con un array de objetos y necesitas inspeccionar su contenido, te recomiendo usar `console.table`

```
const data = [
    {
        nombre: 'Freddy',
        apellido: 'Montes'
    },
    {
        nombre: 'Jose',
        apellido: 'Gonzalez'
    },
    {
        nombre: 'Jhon',
        apellido: 'Doe'
    }
]

console.table(data);
```

Una manera más fácil de ver y analizar tus datos.

¿Cómo detener la ejecución de tu código e inspeccionar?

Detener la ejecución de tu código es un método de depuración bien efectivo porque puedes ver las variables, el contexto, el scope. Los puntos donde detienes la ejecución se les llama "breakpoints" y puedes ponerlos en tu código con el statement `debugger`.

```
function hacerAlgo(param) {
    debugger;

    if (param) {
        // hacer algo
    }

    return param * 2;
}
```

Cuando ejecutes este código con el dev tools abierto, el navegador va a detener toda la ejecución y te muestra la pestaña para inspeccionar todo lo relacionado con ese código.

¿Cómo sumar todos los números de un array?

Muchas veces tenemos un array de productos y necesitamos sumar todos los precios, en lugar de hacer un `for` podemos usar el método `reduce` del array.

```
const precios = [100, 1232, 6453, 234, 2123];
const total = precios.reduce((sub, precio) => {}
    return sub + precio;
, 0);
console.log(total)
```

El método reduce recibe dos parámetros:

1. Una función
2. El valor inicial

La función es donde ocurre la sumatoria, porque esta recibe 2 parámetros, el valor acumulado y el elemento del array por el que está iterando.

El valor acumulado es lo que retorna la misma función en cada iteración.

Prevenir que se cambie un objeto

Mutar objetos en JavaScript puede terminar en bugs dificiles de rastrear. Podemos usar `Object.freeze` para protegernos.

```
const misDatos = {
  nombre: 'Freddy'
};

Object.freeze(misDatos);

misDatos.nombre = 'Carlos';
// Error en strict mode

console.log(misDatos.nombre);
// expected output: Freddy
```

Al “congelar” el objeto no se puede agregar, editar o eliminar propiedades

Hacer scroll a un elemento específico

Cuando necesites hacer scroll dinámicamente hasta una parte especifica de la página puedes usar el método `scrollIntoView`.

Le puedes pasar la propiedad `behavior: 'smooth'` para que el scroll sea animado hasta llegar a la posición.

```
const scrollAqui = document.querySelector('#aqui');

scrollAqui.scrollIntoView({
    behavior: 'smooth'
})
```

Este método funciona para scroll vertical y horizontal.

Iterar un objeto con `for ... in`

La sentencia `for ... in` itera sobre todas las propiedades enumerables de un objeto.

Una vez dentro tienes que acceder al valor usando la notación de `[]`.

```
const misDatos = {
    nombre: 'Freddy',
    apellido: 'Montes',
    web: 'fmontes.com'
};

for (const property in misDatos) {
  console.log(`${property}: ${misDatos[property]}`);
}

// nombre: Freddy
// apellido: Montes
// web: fmontes.com
```

Este método funciona, pero si quieren algo más funcional, vean el siguiente método.

Iterar un objeto con `Object.entries()`

El método `Object.entries()` retorna una matriz (un array de arrays) de pares de propiedades `[key, value]`.

```
const misDatos = {
    nombre: 'Freddy',
    apellido: 'Montes',
    web: 'fmontes.com'
};
const matriz = Object.entries(misDatos);
console.log(matriz)
/*
[
    [nombre, Freddy],
    [apellido, Montes],
    [web, fmontes.com"]
]
*/

Object.entries(misDatos).forEach(([key, value]) => {
    console.log(`${key}: ${value}`);
})

// nombre: Freddy
// apellido: Montes
// web: fmontes.com
```

¿Cómo guardar todos los keys de un objeto en un array?

Para sacar todos los keys de un objeto solo necesitas usar el método del Objeto `.keys`;

```
const misDatos = {
    nombre: 'Freddy',
    apellido: 'Montes',
    web: 'fmontes.com'
};
const misDatosKeys = Object.keys(misDatos);

console.log(misDatosKeys);
// ['nombre', 'apellido', 'web']
```

¿Cómo guardar todos los value de un objeto en un array?

Para sacar todos los value de un objeto solo necesitas usar el método del Objeto `.values()`

```
const misDatos = {
    nombre: 'Freddy',
    apellido: 'Montes',
    web: 'fmontes.com'
};
const misDatosKeys = Object.values(misDatos);

console.log(misDatosKeys);
// ['Freddy', 'Montes', 'fmontes.com']
```

¿Cómo poner una elemento en fullscreen?

Con el API Fullscreen del navegador puedes tomar un elemento de la página y mostrarlo en full screen o pantalla completa.

```
<div id="fullscreen"></div>
```

Solo tenemos que seleccionar el elemento y llamar al método `requestFullscreen`. Usamos un `if` para confirmar que el método existe antes de llamarlo y evitar un error.

```
let fullscreen = document.querySelector("#fullscreen");

if (fullscreen.requestFullscreen) {
  fullscreen.requestFullscreen();
}
```

Te sirve si estás poniendo un video en full screen o si estás trabajando en un videojuego.

CSS

Centrar elementos en pantalla completa

Porque siempre vamos a necesitar centrar elementos en una pantalla.

```
<div class="parent">
    <h1>Mi nombre es Freddy</h1>
</div>
```

```
.parent {
    width: 100vw;
    height: 100vh;
    display: flex;
    place-content: center;
    place-items: center;
}
```

Usar `flex` nos permite centrar elementos fácilmente.

Delimitar texto con puntitos

En algunos diseños responsive no puedes hacer saltos de líneas, así que una solución es delimitar el texto con "ellipsis" que son los ... de "continuará".

```
<h1>Mi nombre es Freddy</h1>
```

```
h1 {
    width: 400px;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis
}
```

Es una de tantas soluciones para mantener el texto responsive, sin embargo si puedes hacer el salto de línea es mejor porque no cortas el texto.

Agregar estilos al placeholder de los inputs

El placeholder es el texto que aparece en el control del formulario cuando no tiene ningún valor establecido.

```
<!-- Borde verde -->
<input type="text" placeholder="Ingresa tu nombre" />

<!-- Borde azul -->
<input type="text"
    placeholder="Ingresa tu apellido"
    value="Montes" />
```

```
input {
    border: solid 1px blue;
}

input:placeholder-shown {
    border: solid 1px green;
}
```

Te permite personalizar hasta el último nivel los `<input>` para ajustarlo hasta el más mínimo detalle.

Agregar estilos a los inputs en sus estados válido e inválido

Cuando usas las validaciones nativas de HTML, como el atributo `required` luego puedes hacer estilos a los elementos del formulario según su estado de validez.

```
<!-- Rojo -->
<input type="text"
    placeholder="Ingresa tu nombre" required>

<!--  Verde -->
<input type="text"
    placeholder="Ingresa tu apellido"
    value="Montes" required>
```

```
input:valid {
    border: solid 1px green;
}

input:invalid {
    border: solid 1px red;
}
```

No JavaScript, con solo CSS puedes indicar al usuario el estado de los campos del formulario.

Aplicar estilos a un elemento solo cuando tenga contenido

La combinación de las pseudo clases “not” y “empty” (cuando no está vacío) nos permite aplicar estilos a elementos SOLO cuando estos tengas contenido.

```
<!-- Se muestra con borde -->
<p>Contenido</p>

<!-- No se muestra nada -->
<p></p>
```

```
p:not(:empty) {
    border: solid 1px;
}
```

Si necesitas tener un elemento HTML donde agregas y eliminas contenido dinámicamente, esta combinación te permite ocultarlo/mostrarlo fácilmente.

Efecto de libro antiguo con la primera letra del párrafo

Se pueden hacer estilos específicos para la primera letra de un párrafo con el psudo-selector `:first-letter`

```
<p>Lorem ipsum dolor sit ... </p>
```

```
p:first-letter {
    font-family: Georgia;
    font-size: 4rem;
    line-height: 1;
}
```

Puedes crear un diseño inspirado en diagramación de libros físicos.

Recortar una imagen con CSS

Si necesitas cambiar la proporción de la imagen en tu página web, solo usa `object-fit`.

```
<img src="/path/a/la/image" />
```

```
img {
    width: 200px;
    height: 200px;
    object-fit: cover;
}
```

Aunque siempre es mejor recortar con un editor de imagen para ahorrar tamaño del archivo, puedes usar esta aproximación para hacerlo solo con css.

Agregar gradiente a un texto

Los gradients son efectos interesantes para resaltar títulos y textos grandes. Evita usarlo en textos pequeños porque los va a hacer muy difícil de leer.

```
h1 {
  font-size: 72px;
  background: -webkit-linear-gradient(#eee, #333);
  background-clip: text;
  text-fill-color: transparent;
}
```

```
<h1>Hello World</h1>
```

Efecto moderno y fácil de implementar.

Poner una imagen de fondo a un texto

Al igual que con texto de gradiente usando background-clip puedes ponerle una imagen de fondo a un texto.

```
h1 {
  background: blue url('path/a/la/imagen');
  background-clip: text;
  color: transparent;
  font-size: 10rem;
}
```

Nada más considera que este efecto se ve bien si el texto es bien grande y una fuente gruesa.

¿Cómo aplicar filtros a un elemento?

La propiedad `filter` aplica efectos gráficos como el desenfoque o el cambio de color a un elemento.

```
.element {
    filter: blur(1rem);
}
```

Uno de los usos es esconder informacion sensible o crear un efecto de “censurado” sobre una imagen.

¿Cómo aplicar filtros al background?

La propiedad `filter` tambien se puede aplicar al fondo de un elemento usando la propiedad `backdrop-filter`.

```
.element {
    backdrop-filter: blur(1rem);
}
```

Aplicando efectos al fondo puedes crear una superficie para colocar texto sobre una imagen.

Hacer una imagen en escala de grises

Una imagen en escala de grises en tu página web da un efecto de elegancia.

```
img {
    filter: grayscale(100%);
}
```

Permite darle un efecto elegante a tu página web.

Quitar los estilos nativos del navegador

La propiedad CSS de `appearance` se utiliza para mostrar un elemento utilizando el estilo nativo de la plataforma.

```
select, input {
    appearance: none;
}
```

Te permite tener un cuadro en blanco para empezar los estilos desde cero en el elemento deseado.

Crear una imagen monocromática

Una imagen monocromática es como escala de grises pero con un color en particular.

```
<div class="color">
  <img src="/ruta/a/la/image.jpg">
</div>
```

```
.color {
    background: #00c2ff;
    /* el color que le queremos dar a la imagen */
}

.color img {
  mix-blend-mode: multiply;
  filter: grayscale(100%);
}
```

Solo necesitas envolver la imagen con un elemento y aplicarle un background y con la propiedad `mix-blend-mode` le decimos a la `img` como queremos que se mezcle con el fondo de color y con el filtro de `grayscale` para hacer la imagen "monotono".

Permitir al usuario cambiar el tamaño de un textarea

Con la propiedad `resize` le permites al usuario cambiar el tamaño del textarea para que se le haga más cómodo ingresar información.

```
<textarea>Hello World</textarea>
```

```
textarea {
    resize: both;
    /* También puedes usar 'vertical' y 'horizonta' */
}
```

Siempre que puedas darle opciones al usuario de controlar su interfaz hay ganancia.

Usar la “calculadora” de CSS

CSS tiene una función que se llama `calc()` y te permite realizar cálculos matemáticos y el resultado usado en cualquier propiedad que necesites números.

```
div {
    width: calc(100% - 100px);
    height: calc(100vh - 200px);
}
```

Lo mejor de todos es que puedes mezclar unidades y hacer operaciones matemáticas de `px` con `rem` o cualquier unidad que necesites.

¿Cómo aplicar estilos a todos los elementos de la página?

Cuidado con como usas el selector universal `*` porque afecta todos los elementos de la página.

```
* {
    border: solid 1px red;
}
```

El selector universal `*` para aplicar estilos a cualquier tipo de elemento en una página web.

Cambiar el color de fondo del texto seleccionado

Para cambiar el color de fondo del texto seleccionado en tu página solo le haces estilos al pseudo-element `::selection`

```
*::selection {
  background-color: yellow;
  color: blue;
}
```

Usamos el selector `*` para aplicárselo a todos los elementos de la página.

Cambiar el color del caret de inserción

El “caret de inserción” es ese “palito que titila” en los campos de texto donde puedes escribir y a este le puedes cambiar el color.

```
input, textarea {
    caret-color: red;
}
```

La propiedad `caret-color` te permite cambiar el color de los campos de texto de tu formulario.

¿Cómo ponerle estilos a la barra de scroll?

El pseudo-element `::-webkit-scrollbar` afecta al estilo de la barra de desplazamiento de un elemento.

Para nuestro proposito necesitamos hacer estilos para dos:

`::-webkit-scrollbar-track`: la "pista" de la barra de desplazamiento.

`::-webkit-scrollbar-thumb`: la "manija" de desplazamiento arrastrada.

```
.elemento::-webkit-scrollbar-track {
  background-color: red;
}

.elemento::-webkit-scrollbar-thumb {
  border-radius:5px;
  background-color: yellow;
}
```

Esto, por ahora **solo funciona en navegadores basados en webkit**, se espera que funcione en todos en un futuro cercano.

¿Cómo hacerle estilos a los elementos dentro y fuera de rango?

Algunos inputs tienen "rangos" por ejemplo, `number` y `date`. Con los pseudo selectores de `:in-range` y `:out-of-range` podemos agregar estilos a estos elementos cuando su valor dentro o fuera del rango asignado.

```
<!-- En rango -->
<input type="number" min="0" max="100" value="99" />

<!-- Fuera de rango -->
<input type="number" min="0" max="100" value="200" />

<!-- Fuera de rango -->
<input type="date" max="2019-12-25" step="1"
    value="2020-12-25">
```

```
input:in-range {
    border: solid 1px green;
}

input:out-of-range {
    border: solid 1px red;
}
```

Esto funciona muy bien para validar formularios.

Escribir CSS que el navegador soporte

El CSS sigue evolucionando y a veces los navegadores no implementan lo más nuevo rápidamente, es por esto que tenemos que ser cuidadoso cuando escribimos CSS nuevo y para eso podemos usar la regla `@supports`.

```
div {
    display: flex;
}

@supports (display: grid) {
  div {
    display: grid;
  }
}
```

En este caso tenemos `flex`, pero si el navegador soporta `grid` entonces usamos grid.

`@supports` se puede usar con cualquier propiedad y valor de CSS.

Aplicar estilos cuando la página está en fullscreen

Cuando el navegador está en modo pantalla completa, puedes usar pseudo-selector de `:fullscreen` para hacer estilos a los elementos.

```
.content:fullscreen {
  background-color: #faa;
}
```

Para cambiar la experiencia de usuario cuando está en modo fullscreen funciona bastante bien.

Agregar un reflejo al contenido

La propiedad CSS `-webkit-box-reflect` permite reflejar el contenido de un elemento en una dirección específica. Los valores que acepta son `above, below, right, left` y además puedes pasarte un segundo valor para el `offset`.

```
h1 {
    -webkit-box-reflect: below 10px;
}
```

Esto, por ahora **solo funciona en navegadores basados en webkit**, se espera que funcione en todos en un futuro cercano.

Crear un grid de cards responsive sin media queries

Podemos crear un layout de cards o cajas que sean completamente responsive y que se ajuste a cualquier pantalla sin necesidad de usar media queries.

```html
<div class="cards">
    <div class="box">Box 1</div>
    <div class="box">Box 2</div>
    <div class="box">Box 3</div>
    <div class="box">Box 4</div>
</div>
```

```css
.cards {
    display: grid;
    grid-template-columns:
        repeat(auto-fit, minmax(200px, 1fr));
    grid-gap: 1rem;
}
```

1. Usamos la función de `repeat` para decirle que se repita N cantidad de veces
2. `auto-fit` es para que los elementos hijos se ajuste al espacio que tenga
3. `minmax` lo que dice es que la caja va a medir `1fr` y mínimo `200px`

Aplicar sombra a una imagen png transparente

Si le agregas `box-shadow` a una imagen le pone la sombra a la caja, pero si le agregas `filter` entonces le pone la sombra a la imagen transparente como tal.

```
<img src="path/to/image.png" />
```

```
img {
    filter: drop-shadow(5px 5px 5px #222222);
}
```

Hacer que el texto rodee una imagen

La imagen tiene que ser un png transparente y luego solo pasamos la propiedad `shape-outside` a la que le puedes pasar como valor el url de la imagen png.

```
<img src="/path/a/la.imagen.png" width="150" />
Lorem ipsum dolor sit amet consectetur
adipisicing elit. Dolores, quibusdam sequi
ipsa libero soluta asperiores a. Ullam quas
unde mollitia officia aspernatur natus iste
quae dolor ipsum expedita. Sunt, necessitatibus!
```

```
img {
    float: left;
    width: 150px;
    margin: 20px;
    shape-outside: url("/path/a/la.imagen.png");
}
```

Este es un efecto inspirado en el diseño editorial.

Prevenir la seleccion de texto

La propiedad `user-select` te permite deshabilitar la selección de texto en tu página web.

```
.no-seleccionar {
  user-select: none;
}
```

Esto funciona bien cuando estás trabajando con drag and drop y es buena práctica que el usuario no pueda seleccionar el texto.

Quitar los eventos de un elemento

Si necesitas que un elemento no emita eventos puedes usar la propiedad `pointer-events`.

```
button {
    pointer-events: none;
}
```

Él `<button>` no va a emitir ningún evento y por ende no se van a poder "escuchar" en JavaScript.

¿Qué es `:is`?

La función de pseudoclase CSS `:is()` recibe una lista de selectores y selecciona cualquier elemento que este dentro de esa lista.

```
:is(header, main, footer) p: {
  color: red;
}
```

Esto selecciona cualquier `p` que este dentro de un `header`, `main` o `footer`, es el equivalente de hacer:

```
header p,
main p,
footer p {
  color: red;
}
```

¿Qué es `:not`?

La pseudoclase CSS `:not()` representa los elementos que no coinciden con una lista de selectores.

Para seleccionar todos los elementos que no sean un `<p>`

```
:not(p) {
  color: blue;
}
```

O podemos seleccionar todos los `<p>` que no tengan la clase `.fancy`

```
p:not(.fancy) {
  color: green;
}
```

Incluso podemos concatenar, aqui un ejemplo de como seleccionar todos los elementos dentro del `body` que no sea `div` ni `span`.

```
body :not(div):not(span) {
  font-weight: bold;
}
```

¿Cómo crear y usar variables?

Las variables en CSS se llaman oficialmente **Custom Properties** y son entidades definidas por los autores de CSS que contienen valores específicos que se reutilizan en todo el documento.

Típicamente se declaran en el nivel más alto del CSS que es el root, aunque también se pueden declarar en cualquier clase:

```
:root {
  --main-color: red;
  --secondary-color: blue;
}
```

Y para usarlas se tiene que utilizar la función `var()`.

```
h1, h2, h3 {
  color: var(--main-color);
}

footer {
 color: var(--secondary-color);
}
```

¿Qué es box-sizing y porque usarla?

El `box-sizing` es una propiedad que te permite establecer como se va a calcular el alto y ancho de una caja.

Por defecto si una caja mide 100x100px, pero tiene `padding` de 10px la caja termina midiendo 120x120px para cambiar eso usamos

```
*,
*:before,
*:after {
  box-sizing: border-box;
}
```

Asignándole `border-box` a todos los elementos de la página cuando pongamos un `width` y `height` no importa el padding o borde que tenga siempre va a medir lo que le pusimos.

¡Muchas gracias por leerme!

Que honor que descargaron o compraron este libro y llegaron hasta aquí de verdad les agradezco muchísimo por confiar en mi contenido.

Para más contenido sobre diseño y desarrollo web o sugerencias estoy en Twitter: https://twitter.com/fmontes y en Instagram https://instagram.com/fmontes.