Guía de HTML más JavaScript (Según contenido programático)

Autor: Ing Johann Latorre

Universidad Sergio Arboleda

Nivel Integrador

Grupo G1

Flexbox = Caja flexible

Grid = Red = se dice gred

Javascript = inicialmente se llamaba Mocha, luego LiveScript, y finalmente Javascript. ​ En diciembre de 1995, Sun Microsystems y Netscape = nesque, anunciaron JavaScript en una conferencia de prensa.

probar caja de input fecha y hora.

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Ejemplo de Cambio de Tamaño de Caja en HTML</title>

<style>

  /\* Estilo para agrandar la caja del input \*/

  .caja-input {

    width: 200px; /\* Ancho deseado \*/

    font-size: 30px; /\*Tamaño Letra\*/

    height: 40px; /\* Alto deseado \*/

    padding: 5px; /\* Espacio alrededor del contenido \*/

    border: 1px solid #ccc; /\* Borde \*/

    border-radius: 5px; /\* Bordes redondeados \*/

  }

</style>

</head>

<body>

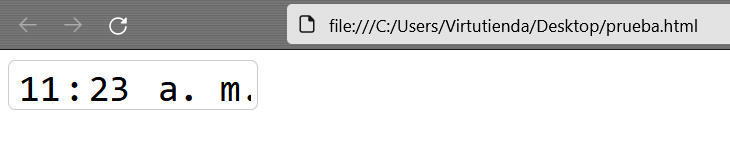
<!-- Input type time con la clase caja-input para aplicar estilos -->

<input type="time" class="caja-input">

</body>

</html>

Prueba



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

Campos required y pattern son ambos atributos de validación en HTML utilizados en formularios para garantizar que los datos ingresados por el usuario cumplan ciertos criterios.

En ingles pronunciación

required == requadier

pattern == pader

1. **required**: Este atributo se aplica a elementos <input>, <select> y <textarea>. Cuando se agrega required a uno de estos elementos, indica que el campo correspondiente debe ser completado antes de enviar el formulario. Si un campo marcado como required se deja vacío, el formulario no se enviará y se mostrará un mensaje de error al usuario.

Ejemplo:

<input type="text" name="nombre" required>

1. **pattern**: Este atributo se usa para especificar un patrón (expresión regular) que el valor de entrada del usuario debe coincidir para ser considerado válido. Se aplica solo a elementos <input> que tienen un tipo que acepta entrada de texto (como text, email, password, etc.). Cuando se usa pattern, el navegador verificará si el valor ingresado por el usuario coincide con la expresión regular especificada y solo permitirá enviar el formulario si el valor cumple con el patrón.

Ejemplo 1:

<input type="text" name="codigo\_postal" pattern="[0-9]{5}" title="Introduce un código postal de 5 dígitos">

Ejemplo 2: Cuando se deja en blanco

<input type="text" name="codigo\_postal" pattern="[0-9]{5}" required>

Etiqueta <audio>

Ejemplo:

 <audio controls>

        <source src="gta.mp3" type="audio/mp3">

        Tu navegador no soporta la etiqueta de audio.

      </audio>

Etiqueta <video>

<video width="320" height="240" controls>

<source src="mi\_video.mp4" type="video/mp4">

Tu navegador no soporta la etiqueta de video.

</video>

**WAI-ARIA Roles: Añade roles ARIA para mejorar la semántica y accesibilidad de componentes interactivos.**

**ROLES ARiA**

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Ejemplo de Roles ARIA</title>

</head>

<body>

<!-- Botón sin roles ARIA -->

<button>Haz clic aquí</button>

<!-- Botón con roles ARIA -->

<button role="button" aria-label="Haz clic aquí para continuar">Haz clic aquí</button>

</body>

</html>

En el primer botón, no se han agregado roles ARIA, por lo que su semántica se basa únicamente en la etiqueta <button>. Sin embargo, en el segundo botón, se han agregado dos atributos ARIA:

1. role="button": Este atributo indica que el elemento tiene el rol de un botón, lo que proporciona más información semántica, especialmente para lectores de pantalla.
2. aria-label="Haz clic aquí para continuar": Este atributo proporciona una etiqueta descriptiva para el botón que será leída por un lector de pantalla. Es útil cuando el texto visible del botón no es suficientemente descriptivo o cuando el botón no tiene texto visible en absoluto.

Al agregar roles ARIA de esta manera, mejoramos la accesibilidad de nuestro botón, lo que hace que sea más fácil para los usuarios con discapacidades visuales o cognitivas entender su propósito y cómo interactuar con él.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Tabindex:** Utiliza el atributo **tabindex** con precaución para asegurar un orden de navegación lógico.

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Ejemplo de Tabindex</title>

</head>

<body>

<!-- Lista de elementos -->

<ul>

<li tabindex="1">Elemento 1</li>

<li tabindex="2">Elemento 2</li>

<li tabindex="3">Elemento 3</li>

</ul>

</body>

</html>

En este ejemplo, tenemos una lista de elementos <li>. Hemos asignado valores de tabindex a cada <li> para controlar el orden de tabulación. Cuando el usuario presiona la tecla Tab para navegar a través de los elementos de la página, el orden de tabulación seguirá el orden de los valores de tabindex. En este caso, el usuario tabulará primero al "Elemento 1", luego al "Elemento 2" y finalmente al "Elemento 3".

Es importante usar el atributo tabindex con precaución para mantener un orden de navegación lógico y predecible para los usuarios. Evita asignar valores de tabindex que no tengan sentido o que interrumpan el flujo natural de navegación de la página.

**:nth-child:**

Selecciona elementos basados en su posición en relación con sus hermanos.

Ejemplo: **:nth-child(2n)** selecciona cada segundo elemento

Ejemplo:

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Ejemplo de :nth-child()</title>

<style>

/\* Estilo para seleccionar cada segundo elemento \*/

ul li:nth-child(2n) {

color: red; /\* Cambiar color de texto a rojo \*/

font-weight: bold; /\* Hacer el texto en negrita \*/

}

</style>

</head>

<body>

<!-- Lista de elementos -->

<ul>

<li>Elemento 1</li>

<li>Elemento 2</li> <!-- Seleccionado -->

<li>Elemento 3</li>

<li>Elemento 4</li> <!-- Seleccionado -->

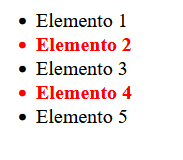
<li>Elemento 5</li>

</ul>

</body>

</html>

Prueba



**:focus:**

Selecciona elementos que están actualmente enfocados, útil para estilos en formularios.

Ejemplo: **input:focus** aplica estilos al input que está activo.

* <!DOCTYPE html>
* <html lang="es">
* <head>
* <meta charset="UTF-8">
* <meta name="viewport" content="width=device-width, initial-scale=1.0">
* <title>Ejemplo de :focus</title>
* <style>
* /\* Estilo para el input cuando está enfocado \*/
* input:focus {
* border: 2px solid blue; /\* Cambia el borde a azul cuando está enfocado \*/
* outline: none; /\* Quita el contorno predeterminado cuando está enfocado \*/
* }
* </style>
* </head>
* <body>
* <!-- Input -->
* <input type="text" placeholder="Haz clic aquí para enfocar">
* </body>
* </html>

En este ejemplo, cuando el usuario hace clic en el input, se le aplicarán los estilos especificados en la regla CSS input:focus. En este caso, el borde del input se cambiará a azul cuando esté enfocado y el contorno predeterminado se eliminará. Esto mejora la apariencia visual y la usabilidad del formulario al resaltar claramente el campo que está siendo interactuado.

¡Claro! Las transiciones en CSS son una forma de animar cambios en las propiedades de los elementos de una página web. Permiten suavizar estos cambios para que no ocurran de manera brusca.

En el ejemplo que proporcionaste:

<!DOCTYPE html>

<html lang="es">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Transiciones CSS</title>

  <style>

    /\* Estilo base del elemento \*/

    .elemento {

      color: black;

      transition: color 0.3s ease; /\* Transición de la propiedad color durante 0.3 segundos con función de aceleración \*/

    }

    /\* Cambio de color al pasar el mouse sobre el elemento \*/

    .elemento:hover {

      color: red;

    }

  </style>

</head>

<body>

  <div class="elemento">

    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed ac dui non quam finibus venenatis.

  </div>

</body>

</html>

* **Animaciones (@keyframes):**
  + Define secuencias de cambios de estilo a lo largo del tiempo.
* <!DOCTYPE html>
* <html lang="es">
* <head>
* <meta charset="UTF-8">
* <meta name="viewport" content="width=device-width, initial-scale=1.0">
* <title>Animación CSS</title>
* <style>
* /\* Definición de la animación \*/
* @keyframes ejemploAnimacion {
* 0% {
* opacity: 0; /\* Al inicio de la animación, la opacidad es 0 (transparente) \*/
* }
* 100% {
* opacity: 1; /\* Al final de la animación, la opacidad es 1 (completamente visible) \*/
* }
* }
* /\* Aplicación de la animación a un elemento \*/
* .elemento {
* animation: ejemploAnimacion 2s ease; /\* Se aplica la animación 'ejemploAnimacion' durante 2 segundos con una función de aceleración \*/
* }
* </style>
* </head>
* <body>
* <div class="elemento">
* Esta es una animación de ejemplo.
* </div>
* </body>
* </html>

**Diferencias Principales entre let y var**

1. **Ámbito (Scope)**
   * **var**: Tiene un ámbito de función. Esto significa que una variable declarada con var es accesible dentro de la función en la que se declara, incluso si se declara dentro de un bloque (como un bucle o una condición).
   * **let**: Tiene un ámbito de bloque. Esto significa que una variable declarada con let es accesible solo dentro del bloque en el que se declara (delimitado por {}).

function ejemploVar() {

if (true) {

var x = 10;

}

console.log(x); // 10

}

function ejemploLet() {

if (true) {

let y = 10;

}

console.log(y); // ReferenceError: y is not defined

}

Enviar datos a una función Javascripts.

**TIPO 1**

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Enviar Número desde HTML a JavaScript</title>

<script>

function recibirNumero(numero) {

console.log("Número recibido:", numero);

}

</script>

</head>

<body>

<button onclick="recibirNumero(42)">Enviar Número</button>

</body>

</html>

**TIPO 2**

<!DOCTYPE html>

<html lang="es">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Enviar Número desde HTML a JavaScript</title>

<script>

function recibirNumero() {

const numero = document.getElementById("numeroInput").value;

console.log("Número recibido:", numero);

}

</script>

</head>

<body>

<input type="number" id="numeroInput" placeholder="Ingresa un número">

<button onclick="recibirNumero()">Enviar Número</button>

</body>

</html>