Interacción con documentos HTML usando Javascript



Tabla de CONTENIDOS

Manipulación del DOM con Javascript: guía paso a paso
Introducción..... 03
¿Qué es el Document Object Model (DOM)?.. 03
Selección de elementos del DOM..... 04
Modificación de elementos del DOM..... 05
Manejo de eventos..... 06
Ejemplos prácticos..... 07
Recursos complementarios..... 07
Cierre..... 07

Validación y modularización de formularios y código en Javascript: manual práctico
Introducción..... 08
Validación de formularios con JavaScript..... 08
Modularización del código en JavaScript: estrategias y buenas prácticas..... 11
Recursos complementarios..... 15
Cierre..... 16

Manejo de arreglos y objetos en Javascript: ejercicios y ejemplos prácticos
Introducción..... 17
Arreglos en JavaScript..... 17
Objetos en JavaScript..... 18
Aplicaciones CRUD utilizando arreglos y objetos..... 19
Recursos complementarios..... 21
Cierre..... 21

Síntesis temática: interacción con documentos HTML usando JavaScript
Introducción..... 21
Manipulación del DOM con Javascript: guía paso a paso..... 22
Validación y modularización de formularios y código en Javascript: manual práctico..... 22
Manejo de arreglos y objetos en Javascript: ejercicios y ejemplos prácticos..... 23
Evaluación sumativa 2: desarrollo de aplicación web con validación de formulario..... 23

Manipulación del DOM con Javascript: guía paso a paso

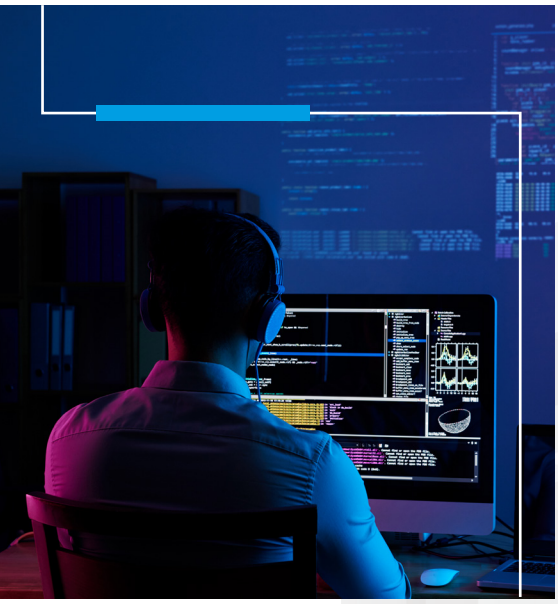
Introducción

El Document Object Model (DOM) es una representación estructurada del documento HTML que permite a los desarrolladores manipular su contenido y estructura mediante JavaScript. La capacidad de modificar el DOM es esencial para crear páginas web dinámicas e interactivas.

Esta representación convierte cada elemento HTML en un nodo dentro de una estructura jerárquica, lo que permite acceder y modificar cada uno de estos nodos de manera programática.

Gracias al DOM, los desarrolladores pueden cambiar texto, atributos, estilos y hasta la estructura completa del documento en respuesta a eventos del usuario, como clics, movimientos del ratón o entradas de teclado, lo que permite una experiencia de usuario mucho más rica y personalizada.

El DOM no solo facilita la manipulación directa de los elementos HTML, sino que también permite la integración con otras tecnologías y librerías de JavaScript. Por ejemplo, a través del DOM, los desarrolladores pueden interactuar con APIs del navegador, realizar solicitudes AJAX para actualizar partes de la página sin recargarla completamente, y aplicar animaciones mediante CSS y JavaScript. Además, el uso eficiente del DOM puede mejorar el rendimiento de las aplicaciones web, ya que permite cargar y modificar contenido de manera asíncrona, optimizando el tiempo de respuesta y reduciendo la carga en el servidor. En resumen, dominar el DOM y su manipulación con JavaScript es fundamental para cualquier desarrollador front end que desee crear aplicaciones web modernas, eficientes e interactivas.



¿Qué es el Document Object Model (DOM)?

El DOM es una interfaz de programación para documentos HTML y XML.

De igual manera representa la página, de manera que los programas pueden cambiar la estructura del documento, su estilo y su contenido.

A continuación, conocerás los elementos que componen el DOM:

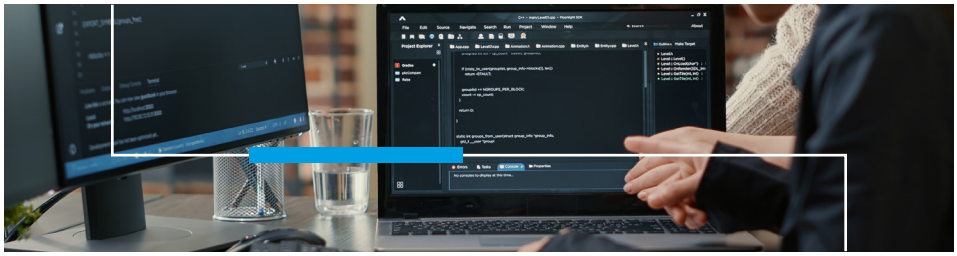
- **Árbol del DOM:** el DOM se organiza en una estructura de árbol donde cada nodo es un “hijo” del nodo que lo contiene, llamado “nodo padre”.

- **Nodos del DOM:** cada parte de un documento HTML es un nodo del DOM. Dichos nodos pueden ser elementos, atributos, texto, etc.

Selección de elementos del DOM

Para manipular el DOM, primero necesitas seleccionar los elementos que quieres cambiar.

A continuación, puedes usar el siguiente código:



Selección de elemento: getElementById: selecciona un elemento por su ID.

```
let elemento = document.getElementById('miElemento');
```

Elemento de clase específica: getElementsByClassName: selecciona todos los elementos con una clase específica.

```
let elementos = document.getElementsByClassName('miClase');
```

getElementByName: esta función permite seleccionar elementos por su nombre, devolviendo una lista de todos los ítems con el nombre especificado.

```
let camposUsuario = document.getElementsByName("username");
```

querySelector y querySelectorAll: selecciona el primer elemento que coincide con un selector CSS (querySelector) o todos los elementos que coinciden (querySelectorAll).

```
let elemento = document.querySelector('.miClase');
let elementos = document.querySelectorAll('.miClase');
```

Modificación de elementos del DOM

Una vez que has seleccionado un elemento, puedes modificarlo con las siguientes funciones:

Modificar el contenido:

innerHTML: permite obtener o establecer el contenido HTML de un elemento, esto incluye no solo el texto, sino también cualquier elemento HTML dentro del elemento.

textContent: permite obtener o establecer el texto dentro de un elemento. Esta propiedad no interpreta ni incluye etiquetas HTML, solo el texto plano.

```
elemento.innerHTML = '<h1>Nuevo contenido</h1>';
elemento.textContent = 'Nuevo texto';
```

Comparación: En esa misma línea, teniendo en cuenta las funciones que acabas de ver, es importante mencionar que hay tres criterios de comparación:

Contenido HTML: innerHTML permite la inclusión y manipulación de etiquetas HTML dentro del contenido del elemento, mientras que textContent solo maneja texto plano.

Seguridad: innerHTML puede ser vulnerable a ataques de Cross-Site Scripting (XSS) si se manipula con datos no confiables, ya que puede interpretar y ejecutar código HTML y JavaScript malicioso. textContent, por otro lado, se considera más seguro porque no interpreta el contenido como HTML.

Rendimiento: textContent suele ser más rápido y eficiente cuando se trabaja únicamente con texto, ya que no necesita analizar y renderizar el contenido HTML.

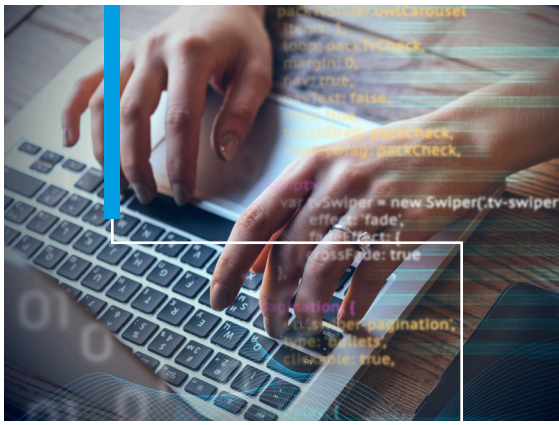
De igual manera, puedes cambiar los atributos de cualquier etiqueta HTML, por ejemplo, modificar la imagen de una etiqueta <img> reemplazando el atributo src, para ello puedes usar el siguiente código:

```
elemento.setAttribute('src', 'nuevaImagen.jpg');
let valor = elemento.getAttribute('src');
elemento.removeAttribute('src');
```

Modificar estilos: Adicional a lo anterior, puedes cambiar los estilos de cualquier etiqueta HTML, para ello puedes usar el siguiente código:

```
elemento.setAttribute('src', 'nuevaImagen.jpg');
let valor = elemento.getAttribute('src');
elemento.removeAttribute('src');
```

Manejo de eventos



Son acciones que ocurren en el navegador, como clics, movimientos del ratón y envíos de formularios.

Para ello, puedes usar JavaScript para escuchar estos eventos y responder a ellos.

A continuación, conocerás las principales funciones de esta herramienta:

Agregar un evento: Esta es una función que puede ser ejecutada a través del siguiente código:

```
elemento.addEventListener('click', function() {
   alert('Elemento clickeado');
});
```

Eliminar un evento: Esta es una función que puede ser ejecutada a través del siguiente código:

```
function miFuncion() {
   alert('Evento eliminado');
}
elemento.removeEventListener('click', miFuncion);
```

Para profundizar en la percepción de tu conocimiento, tienes la siguiente tabla:

Tabla resumen de eventos comunes en JavaScript	
click	Ocurre cuando un elemento es clicleado.
dblclick	Ocurre cuando un elemento es doble clicleado.
mouseover	Ocurre cuando el puntero del ratón se coloca sobre un elemento.
mouseout	Ocurre cuando el puntero del ratón se aleja de un elemento.
keydown	Ocurre cuando una tecla es presionada.
keyup	Ocurre cuando una tecla es presionada.
change	Ocurre cuando el valor de un elemento de formulario cambia.
submit	Ocurre cuando un formulario es enviado.
focus	Ocurre cuando un elemento gana el foco.
blur	Ocurre cuando un elemento pierde el foco.
input	Ocurre cuando un valor de entrada es modificado.

Ejemplos prácticos

Para garantizar la comprensión de las temáticas vistas, aquí tienes dos ejemplos:

Ejemplo 1: cambiar el contenido de un elemento

HTML

```
<div id=”miDiv”>Contenido original</div>
<button onclick=”cambiarContenido()”>Cambiar Contenido</button>
```

Javascript

```
function cambiarContenido() {
   let div = document.getElementById(‘miDiv’);
   div.innerHTML = ‘Contenido cambiado’;
}
```

Ejemplo 2: cambiar estilos con un evento

HTML

```
<div id=”miDiv” style=”width: 100px; height: 100px; background-color: yellow;”></div>
<button id=”miBoton”>Cambiar Color</button>
```

Javascript

```
document.getElementById(‘miBoton’).addEventListener(‘click’, function() {
   document.getElementById(‘miDiv’).style.backgroundColor = ‘blue’;
});
```

Recursos complementarios

A continuación, encontrarás recursos para complementar lo aprendido sobre la manipulación del DOM con JavaScript: guía paso a paso).

- W3 Schools. (s. f.). *JavaScript tutorial*. W3 Schools. https://www.w3schools.com/js/

Cierre

Esta semana hemos explorado la manipulación del DOM utilizando JavaScript, una habilidad fundamental para cualquier desarrollador web. Hemos aprendido a seleccionar y modificar elementos, a cambiar sus atributos y estilos, y a manejar eventos para crear interactividad en nuestras páginas web.

Estos conocimientos no solo permiten mejorar la apariencia y funcionalidad de los sitios web, sino que también son esenciales para la creación de aplicaciones web dinámicas y responsivas.

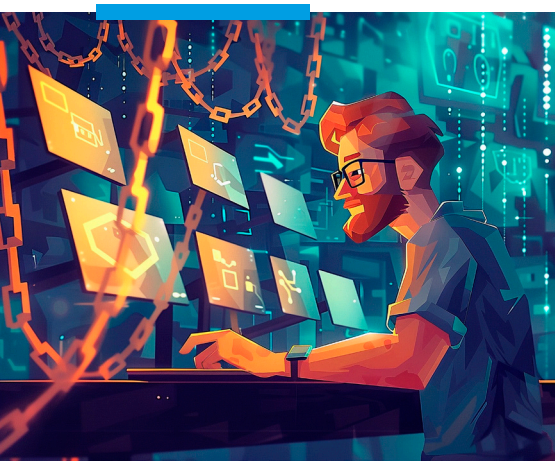
Validación y modularización de formularios y código en Javascript: manual práctico

Introducción

La validación de formularios es un aspecto crucial en el desarrollo web, ya que garantiza que los datos ingresados por los usuarios sean correctos y adecuados antes de ser procesados o almacenados.

La validación del lado del cliente con JavaScript mejora la experiencia del usuario al proporcionar retroalimentación inmediata y reducir la carga en el servidor.

Las funciones en JavaScript son bloques de código reutilizables que permiten organizar y estructurar mejor el código de forma general. Utilizar funciones adecuadamente no solo mejora la legibilidad y mantenibilidad del código, sino que también facilita la depuración y la reutilización de la lógica.



Validación de formularios con JavaScript

¿Qué es la validación de formularios?

Se refiere al proceso de verificar que los datos ingresados por los usuarios cumplen con ciertos criterios antes de ser enviados al servidor. Esto incluye asegurarse de que los campos obligatorios no estén vacíos, que los datos sean del tipo correcto (como un correo electrónico o un número) y que cumplan con las restricciones de formato.

Ahora que ya conoces la definición, abordaremos los tipos de validación:

Validación del lado del cliente
• Se realiza en el navegador del usuario utilizando JavaScript. • Proporciona retroalimentación inmediata y reduce la cantidad de solicitudes al servidor.
Validación del lado del servidor
• Se realiza en el servidor después de que los datos son enviados. • Es esencial para la seguridad, ya que no se puede confiar completamente en la validación del lado del cliente.

Validación básica con JavaScript

Para validar formularios con JavaScript, primero necesitamos seleccionar los elementos del formulario y luego aplicar las reglas de validación. A continuación, tienes un ejemplo de código de formulario HTML:

```
<form id=”miFormulario”>
  <label for=”nombre”>Nombre:</label>
  <input type=”text” id=”nombre” name=”nombre”><br><br>
  <label for=”email”>Email:</label>
  <input type=”email” id=”email” name=”email”><br><br>
  <label for=”edad”>Edad:</label>
  <input type=”number” id=”edad” name=”edad”><br><br>
  <button type=”submit”>Enviar</button>
</form>
<div id=”errores”></div>
```

Validación de campos vacíos

Es un proceso en el desarrollo de aplicaciones web y formularios en el que se verifica que ciertos campos de entrada no estén vacíos antes de permitir que el usuario envíe el formulario. Esto es crucial para asegurar la integridad de los datos ingresados y evitar errores en el procesamiento de la información.

A continuación, tienes el código para validar campos vacíos:

```
document.getElementById(‘miFormulario’).addEventListener(‘sub-
mit’, function(event) {
  let errores = [];
  let nombre = document.getElementById(‘nombre’).value;
  let email = document.getElementById(‘email’).value;
  let edad = document.getElementById(‘edad’).value;

  if (nombre === ‘’) {
    errores.push(‘El campo nombre es obligatorio.’);
  }

  if (email === ‘’) {
    errores.push(‘El campo email es obligatorio.’);
  }

  if (edad === ‘’) {
    errores.push(‘El campo edad es obligatorio.’);
  }

  if (errores.length > 0) {
    event.preventDefault();
    document.getElementById(‘errores’).innerHTML = errores.
join(‘<br>’);
  }
});
```

Validación de tipos de datos

Además de verificar si los campos están vacíos, también es importante validar el tipo de datos ingresados, por ejemplo, un correo electrónico. Aquí tienes el código para hacerlo:

```
function esEmailValido(email) {
  let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}
document.getElementById('miFormulario').addEventListener('sub-
mit', function(event) {
  let errores = [];
  let email = document.getElementById('email').value;

  if (!esEmailValido(email)) {
    errores.push('El formato del email es incorrecto.');
  }

  if (errores.length > 0) {
    event.preventDefault();
    document.getElementById('errores').innerHTML = errores.
join('<br>');
  }
});
```

Validación de rango numérico

Para campos numéricos, puedes validar que el valor ingresado esté dentro de un rango específico, por ejemplo, edades entre 18 y 65 años. Aquí tienes el código para hacerlo:

```
function esEdadValida(edad) {
  let min = 18;
  let max = 65;
  return edad >= min && edad <= max;
}

document.getElementById('miFormulario').addEventListener('submit', function(e-
vent) {
  let errores = [];
  let edad = document.getElementById('edad').value;

  if (!esEdadValida(edad)) {
    errores.push('La edad debe estar entre 18 y 65 años.');
  }

  if (errores.length > 0) {
    event.preventDefault();
    document.getElementById('errores').innerHTML = errores.join('<br>');
  }
});
```

Prevención del envío del formulario

Para evitar que el formulario se envíe si hay errores, se utiliza el método preventDefault() dentro del manejador de eventos del formulario.

A continuación, tienes un ejemplo completo de validación:

```
document.getElementById('miFormulario').addEventListener('sub-
mit', function(event) {
  let errores = [];
  let nombre = document.getElementById('nombre').value;
  let email = document.getElementById('email').value;
  let edad = document.getElementById('edad').value;

  if (nombre === '') {
    errores.push('El campo nombre es obligatorio.');
  }

  if (email === '') {
    errores.push('El campo email es obligatorio.');
  } else if (!esEmailValido(email)) {
    errores.push('El formato del email es incorrecto.');
if (edad === '') {
    errores.push('El campo edad es obligatorio.');
  } else if (!esEdadValida(edad)) {
    errores.push('La edad debe estar entre 18 y 65 años.');
  }

  if (errores.length > 0) {
    event.preventDefault();
    document.getElementById('errores').innerHTML = errores.
join('<br>');
  }
});
function esEmailValido(email) {
  let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}
function esEdadValida(edad) {
  let min = 18;
  let max = 65;
  return edad >= min && edad <= max;
}
```

Modularización del código en JavaScript: estrategias y buenas prácticas

¿Qué es una función?

Es un bloque de código que realiza una tarea específica y que puede ser invocado en cualquier parte del programa. Las funciones pueden aceptar parámetros y devolver un valor.

A continuación, tienes el código de la sintaxis básica:

```
function nombreDeLaFuncion(param1, param2) {
  // Código a ejecutar
  return resultado;
}
```

Un ejemplo claro de esta función incluye el siguiente código:

```
function sumar(a, b) {
  return a + b;
}
let resultado = sumar(3, 5); // resultado es 8
```

Tipos de funciones

Existen tres tipos de funciones diferentes, las cuales conocerás a continuación:

Funciones declarativas:

Son definidas con la palabra clave function.

Pueden ser llamadas antes de su declaración debido al hosting.

El hosting en JavaScript es el comportamiento que mueve las declaraciones de variables y funciones al principio del código durante la compilación. Esto permite que se utilicen antes de ser declaradas en el código.

Comprender el hosting es crucial para evitar errores y asegurar un uso adecuado de variables y funciones en el código JavaScript.

A continuación, tienes el código ejemplo de la función declarativa:

```
function saludar(nombre) {
  return `Hola, ${nombre}`;
}
 console.log(saludar('Juan')); // Hola, Juan
```

Funciones expresivas

Son definidas como expresiones y están asignadas a una variable.

No son elevadas (hoisted).

A continuación, tienes el código ejemplo de la función expresiva:

```
const saludar = function(nombre) {
  return `Hola, ${nombre}`;
};
 console.log(saludar('Ana')); // Hola, Ana
```

Funciones flecha

Introducidas en ES6 (lenguaje de scripting), tienen una sintaxis más concisa.

No tienen su propio this ni arguments.

A continuación, tienes el código ejemplo de la función flecha:

```
const saludar = (nombre) => `Hola, ${nombre}`;
 console.log(saludar('Carlos')); // Hola, Carlos
```

Parámetros y argumentos

Parámetros: son variables listadas como parte de la declaración de una función.

Argumentos: son valores pasados a la función cuando se invoca.

A continuación, tienes el código ejemplo que se utiliza:

```
function restar(a, b) {
  return a - b;
}
 let resultado = restar(10, 5); // resultado es 5
```

Funciones anidadas y ámbito

Funciones anidadas: es una función definida dentro de otra función.

A continuación, tienes el código ejemplo que se utiliza:

```
function exterior() {
  let variableExterior = 'Soy exterior';

  function interior() {
    let variableInterior = 'Soy interior';
    console.log(variableExterior); // Accede a variable exterior
  }
   interior();
  console.log(variableInterior); // Error: variableInterior no está
definida
}
 exterior();
```

Ámbito: define la accesibilidad de variables y funciones en diferentes partes del código. En JavaScript existen principalmente tres tipos de ámbitos: global, de función y de bloque. Las variables y funciones definidas en un ámbito están disponibles solo dentro de ese ámbito y sus ámbitos anidados, lo que ayuda a controlar la visibilidad y la vida útil de las variables.

Existen dos tipos de variables y, a continuación, conocerás sus definiciones:

Ámbito global:
Son variables definidas fuera de cualquier función, accesibles desde cualquier parte del código.



Ámbito local:
Son variables definidas dentro de una función, accesibles solo dentro de esa función.

Funciones de orden superior

Son funciones que aceptan otras funciones como argumentos o devuelven funciones.

A continuación, tienes el código ejemplo que se utiliza:

```
function operacion(a, b, callback) {
  return callback(a, b);
}

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

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

console.log(operacion(5, 3, sumar)); // 8
console.log(operacion(5, 3, multiplicar)); // 15
```

Ahora que ya conoces todos los códigos de esta lección, observa dos ejemplos de código práctico:

Ejemplo 1: validación de formulario con funciones

```
function validarNombre(nombre) {
  return nombre.trim() !== '';
}

function validarEmail(email) {
  let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}

function validarFormulario(nombre, email) {
  if (!validarNombre(nombre)) {
    console.log('El nombre es obligatorio.');
    return false;
  }
  if (!validarEmail(email)) {
    console.log('El email es inválido.');
    return false;
  }
  console.log('Formulario válido.');
  return true;
}

validarFormulario('Juan', 'juan@example.com'); // Formulario válido.
```

Ejemplo 2: función de orden superior para manejo de Arrays

```
const numeros = [1, 2, 3, 4, 5];

const cuadrados = numeros.map(function(numero) {
  return numero * numero;
});

console.log(cuadrados); // [1, 4, 9, 16, 25]
```

Recursos complementarios

A continuación, encontrarás recursos para complementar lo aprendido sobre la validación y modularización de formularios y código en Javascript: manual práctico.

La guía de ejercicios para practicar funciones en JavaScript ofrece instrucciones y ejemplos para cada ejercicio. Se recomienda leer cuidadosamente las instrucciones, intentar resolver los ejercicios por cuenta propia, revisar ejemplos y buscar información adicional si es necesario.

Es importante verificar las respuestas y repasar el funcionamiento de cada ejercicio una vez completados. En caso de dudas, no dudes en buscar ayuda en recursos en línea o al docente de la asignatura. Continúa practicando hasta completar todos los ejercicios de la guía.

Ejercicio 1: función suma

Objetivo: crear una función que sume dos números.

Instrucciones:

1. Define una función llamada sumar que acepte dos parámetros a y b.

2. La función debe retornar la suma de a y b.

3. Llama a la función con diferentes valores y muestra los resultados en la consola.

Código utilizado en la función suma:

```
function sumar(a, b) {
  return a + b;
}
 console.log(sumar(3, 4)); // 7
console.log(sumar(10, 20)); // 30
console.log(sumar(-5, 5)); // 0
```

Ejercicio 2: función de validación de email

Objetivo: crear una función que valide el formato de un correo electrónico.

Instrucciones:

1. Define una función llamada esEmailValido que acepte un parámetro email.

2. La función debe retornar true si el correo electrónico tiene un formato válido y false en caso contrario.

3. Utiliza una expresión regular para validar el formato del correo electrónico.

4. Llama a la función con diferentes correos electrónicos y muestra los resultados en la consola.

Código utilizado en la función validación de email:

```
function esEmailValido(email) {
  let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}
console.log(esEmailValido('test@example.com')); // true
console.log(esEmailValido('test@.com')); // false
console.log(esEmailValido('test@example')); // false
```

Cierre

La validación de formularios con JavaScript es una técnica esencial para mejorar la calidad y seguridad de los datos ingresados por los usuarios. Además, proporciona una experiencia de usuario más fluida y reduce la carga en el servidor.

A través de ejemplos prácticos, has podido entender cómo implementar validaciones básicas y avanzadas en tus proyectos.

Manejo de arreglos y objetos en Javascript: ejercicios y ejemplos prácticos

Introducción

Los arreglos y objetos son estructuras de datos fundamentales en JavaScript que permiten almacenar y manipular conjuntos de datos de manera eficiente, lo cual es esencial para el desarrollo de aplicaciones web dinámicas.

Los arreglos en JavaScript son listas ordenadas de elementos que pueden ser accedidos mediante índices y son útiles para manejar colecciones de datos homogéneos, como listas de nombres o números. Durante esta semana, aprenderemos a crear, acceder y manipular arreglos mediante diversas técnicas y métodos nativos de JavaScript. Además, veremos cómo recorrer arreglos utilizando bucles y funciones de orden superior, lo que nos permitirá realizar operaciones complejas de manera sencilla y eficiente.

Arreglos en JavaScript

Los arreglos son colecciones ordenadas de elementos que pueden ser de cualquier tipo de dato. Cada elemento en un arreglo tiene un índice, comenzando desde 0.

Declaración de arreglos

Es importante aclarar que esta acción consiste en crear una estructura de datos que puede almacenar múltiples valores en una sola variable. A continuación, conocerás el código que utiliza:

```
et numeros = [1, 2, 3, 4, 5];
let frutas = [“manzana”, “naranja”, “plátano”];
```

Acceso a elementos del arreglo

El acceso a elementos de un arreglo en JavaScript se realiza utilizando índices que comienzan en 0. A continuación, verás el código que se utiliza:

```
let primeraFruta = frutas[0]; // “manzana”
```

Métodos comunes de arreglos

Existen distintos métodos que se usan para facilitar arreglos comunes, a continuación, conocerás algunos de ellos:

push: se utiliza para añadir un elemento al final del arreglo. Esta es la función escrita:

```
frutas.push(“uva”);
```

pop: se utiliza para eliminar el último elemento del arreglo. Esta es la función escrita:

```
frutas.pop();
```

shift: se utiliza para eliminar el primer elemento del arreglo. Esta es la función escrita:

```
frutas.shift();
```

unshift: añade un elemento al inicio del arreglo. Esta es la función escrita:

```
let numerosDobles = numeros.map(num => num * 2);
```

map: crea un nuevo arreglo con los resultados de la llamada a una función para cada elemento. Esta es la función escrita:

```
let numerosDobles = numeros.map(num => num * 2);
```

filter: crea un nuevo arreglo con todos los elementos que cumplan una condición. Esta es la función escrita:

```
let numerosPares = numeros.filter(num => num % 2 === 0);
```

reduce: aplica una función a un acumulador y a cada valor del arreglo (de izquierda a derecha) para reducirlo a un único valor. Esta es la función escrita:

```
let suma = numeros.reduce((acumulador, num) => acumulador +
num, 0);
```

Objetos en JavaScript

Los arreglos son colecciones ordenadas de elementos que pueden ser de cualquier tipo de dato. Cada elemento en un arreglo tiene un índice, comenzando desde 0.

Declaración de objetos

Es importante aclarar que esta acción consiste en crear estructuras de datos que almacenan pares de clave-valor. A continuación, conocerás el código que utiliza:

```
let persona = {
  nombre: “Juan”,
  edad: 30,
  profesion: “Desarrollador”
};
```

Acceso a propiedades del objeto

El acceso a propiedades de un objeto en JavaScript se realiza utilizando la notación de punto o la notación de corchetes. A continuación, aparece el código que se utiliza:

```
let nombre = persona.nombre; // “Juan”
let edad = persona[“edad”]; // 30
```

Modificación de propiedades

Se refiere al proceso de cambiar el valor de una propiedad existente de un objeto. A continuación, está el código que se utiliza:

```
persona.edad = 31;
persona[“profesion”] = “Ingeniero de Software”;
```

Añadir nuevas propiedades

Consiste en incorporar pares de clave-valor adicionales a un objeto existente. A continuación, aparece el código que se utiliza:

```
persona.pais = “España”;
```

Eliminar propiedades

Implica quitar un par de clave-valor del objeto. A continuación, está el código que se utiliza:

```
delete persona.profesion;
```

Aplicaciones CRUD utilizando arreglos y objetos

CRUD se refiere a las operaciones crear, leer, actualizar y eliminar. Estas operaciones son fundamentales en muchas aplicaciones web.

Ejemplo de aplicación CRUD

Imaginemos una aplicación de gestión de tareas. Para ello, en primer lugar, deberás crear la tarea y aquí tienes el código para hacerlo:

```
let tareas = [];

function crearTarea(nombre, descripcion) {
  let nuevaTarea = {
    id: Date.now(),
    nombre: nombre,
    descripcion: descripcion,
    completada: false
  };
  tareas.push(nuevaTarea);
}
```

En segundo lugar, deberás leer tareas y aquí tienes el código para hacerlo:

```
function leerTareas() {
  return tareas;
}
```

En tercer lugar, deberás actualizar tareas y aquí tienes el código para hacerlo:

```
function actualizarTarea(id, datosActualizados) {
  let tarea = tareas.find(t => t.id === id);
  if (tarea) {
    Object.assign(tarea, datosActualizados);
  }
}
```

En cuarto y último lugar, deberás eliminar tareas y aquí tienes el código para hacerlo:

```
function eliminarTarea(id) {
  tareas = tareas.filter(t => t.id !== id);
}
```

Ejemplos prácticos

Para profundizar en tu conocimiento, tienes algunos ejemplos:

Ejemplo de manipulación de arreglos

```
let numeros = [1, 2, 3, 4, 5];
let numerosDobles = numeros.map(num => num * 2);
console.log(numerosDobles); // [2, 4, 6, 8, 10]
```

Ejemplo de manipulación de objetos:

```
let persona = {
  nombre: “Juan”,
  edad: 30,
  profesion: “Desarrollador”
};

persona.edad = 31;
persona.pais = “España”;
delete persona.profesion;

console.log(persona);
```

Recursos complementarios

A continuación, encontrarás recursos para complementar lo aprendido sobre el manejo de arreglos y objetos en Javascript: ejercicios y ejemplos prácticos.

W3 Schools. (s. f.). JavaScript array methods. W3 Schools. https://www.w3schools.com/js/js_array_methods.asp

Cierre

Esta semana hemos aprendido a utilizar arreglos y objetos en JavaScript para almacenar y manipular datos. Estas estructuras son esenciales para desarrollar aplicaciones web dinámicas y eficientes. Además, hemos explorado cómo implementar operaciones CRUD, las cuales son fundamentales en muchas aplicaciones prácticas.

Síntesis temática: interacción con documentos HTML usando JavaScript

Introducción

La interacción con documentos HTML usando JavaScript es una técnica esencial en el desarrollo web moderno, que permite a los desarrolladores crear experiencias de usuario dinámicas e interactivas. A través del Document Object Model (DOM), JavaScript puede acceder y modificar la estructura, el contenido y el estilo de una página web en tiempo real. Esto incluye la capacidad de responder a eventos del usuario, como clics de botones y entradas de teclado, así como actualizar y manipular elementos HTML para reflejar cambios instantáneos sin necesidad de recargar la página. Esta capacidad transforma las páginas estáticas en aplicaciones web interactivas, mejorando significativamente la usabilidad y la funcionalidad de los sitios web.

Manipulación del DOM con Javascript: guía paso a paso

Tema e importancia: la manipulación del DOM con JavaScript es esencial para crear aplicaciones web dinámicas e interactivas, permitiendo modificar y actualizar el contenido y la estructura de las páginas web en tiempo real.

Conceptos clave: DOM (Document Object Model), selectores DOM y eventos DOM.

La manipulación del DOM con JavaScript permite a los desarrolladores modificar la estructura, el contenido y el estilo de una página web de manera dinámica. A través de selectores y métodos de manipulación, como appendChild, removeChild y setAttribute, se pueden crear interfaces de usuario interactivas que respondan a las acciones del usuario sin necesidad de recargar la página. Esta capacidad de alterar el DOM en tiempo real es fundamental para el desarrollo de aplicaciones web modernas, mejorando la experiencia del usuario mediante actualizaciones inmediatas y personalizadas del contenido.

Ideas fuerza

1. La manipulación del DOM permite actualizaciones dinámicas y en tiempo real del contenido y la estructura de la página web.

2. Los selectores y métodos DOM son herramientas clave para acceder y modificar elementos específicos del documento HTML.

3. La gestión de eventos DOM es fundamental para crear aplicaciones web interactivas y responsivas, mejorando la experiencia del usuario.

Validación y modularización de formularios y código en Javascript: manual práctico

Importancia: la validación y modularización de formularios y código en JavaScript es crucial para asegurar la integridad de los datos y mantener un código limpio, reutilizable y fácil de mantener en aplicaciones web.

Conceptos clave: validación de formularios, modularización del código y JavaScript.

La validación de formularios en JavaScript es una práctica esencial para asegurar que los datos ingresados por los usuarios sean correctos y seguros antes de ser enviados al servidor. Esto incluye verificar que los campos no estén vacíos, que los valores sean del tipo y formato correctos, y que cumplan con todas las reglas de negocio definidas. La validación puede ser realizada tanto en el lado del cliente como en el servidor, aunque realizarla en el lado del cliente mejora la experiencia del usuario al proporcionar retroalimentación inmediata y reduce la carga en el servidor.

Ideas fuerza

1. La validación de formularios garantiza la integridad y seguridad de los datos ingresados por los usuarios.

2. La modularización del código facilita la reutilización, mantenimiento y escalabilidad de aplicaciones web.

3. JavaScript es una herramienta clave para implementar tanto la validación de formularios como la modularización del código en el desarrollo web.

Manejo de arreglos y objetos en Javascript: ejercicios y ejemplos prácticos

Importancia: el manejo de arreglos y objetos en JavaScript es fundamental para gestionar y manipular datos complejos en aplicaciones web, permitiendo un desarrollo más eficiente y organizado.

Conceptos claves: arreglos, objetos y métodos de arreglos y objetos.

El manejo de arreglos y objetos en JavaScript es una habilidad crucial para cualquier desarrollador web, ya que estas estructuras de datos son esenciales para organizar y manipular grandes volúmenes de información de manera eficiente. Los arreglos permiten almacenar y acceder a colecciones ordenadas de elementos, y sus métodos integrados facilitan la realización de operaciones comunes como la iteración, la búsqueda y la transformación de datos. Por otro lado, los objetos son ideales para representar entidades con múltiples atributos, permitiendo acceder y modificar sus propiedades de manera directa.

Ideas fuerza

1. El manejo de arreglos y objetos en JavaScript es esencial para organizar y manipular datos complejos en aplicaciones web.

2. Los métodos integrados de arreglos y objetos permiten realizar operaciones comunes de manera eficiente y concisa.

3. Dominar estas estructuras de datos mejora la limpieza, la reutilización y el mantenimiento del código en el desarrollo web.

Evaluación sumativa 2: desarrollo de aplicación web con validación de formulario

Importancia: la evaluación sumativa sobre el desarrollo de una aplicación web con validación de formularios es crucial para consolidar conocimientos y habilidades prácticas en el manejo de datos y la creación de interfaces seguras y eficientes.

Conceptos clave: validación de formularios, aplicación web y evaluación sumativa.

La evaluación sumativa en el desarrollo de una aplicación web con validación de formularios es una herramienta pedagógica clave que te permite demostrar tu comprensión y habilidades en un contexto práctico. Durante esta evaluación, debes aplicar sus conocimientos en HTML, CSS y JavaScript, para crear una aplicación web funcional que incluye mecanismos de validación de formularios.

Ideas fuerza

1. La evaluación sumativa en el desarrollo web refuerza conocimientos y habilidades prácticas en la creación de aplicaciones con validación de formularios.

2. La validación de formularios mejora la calidad y seguridad de la información ingresada por los usuarios.

3. Crear una aplicación web funcional permite a los estudiantes demostrar su competencia en tecnologías web y estándares de desarrollo.

Bibliografía

Antani, V. (2016). *Mastering JavaScript*. Packt Publishing.

Flanagan, D. (2011). *JavaScript: The definitive guide*. 6.a ed. O'Reilly Media. https://books.google.es/

W3 Schools. (s. f.). *JavaScript tutorial*. W3 Schools. https://www.w3schools.com/js/