CLASE 9 · 03/06/2026

JavaScript: funciones, eventos e interacción

Cómo pasar de una página estática a una interfaz que responde a las acciones del usuario.

VIRTUAL JAVASCRIPT DOM BÁSICO

DÓNDE ESTAMOS

Ya tenemos estructura, diseño y lógica inicial

HTML

Estructura el contenido: títulos, textos, formularios, botones.

CSS

Define la presentación visual: colores, tamaños, espaciados y layout.

JavaScript

Agrega comportamiento: decisiones, cálculos e interacción.

OBJETIVO

Qué deberían poder hacer al final

- Crear funciones simples con parámetros y **return**.
- Detectar acciones del usuario mediante eventos.
- Seleccionar elementos HTML desde JavaScript.
- Leer datos de formularios, mostrar mensajes y validar campos básicos.
- Cambiar clases CSS desde JavaScript para modificar estados visuales.

IDEA CENTRAL

Una página puede reaccionar

HTML

```
<h1>Inscripción a curso</h1>
<input type="text" placeholder="Nombre">
<button>Enviar</button>
```

- El botón puede responder al clic.
- El sistema puede leer lo escrito.
- Puede mostrar mensajes.
- Puede validar datos.
- Puede cambiar el estado visual de la página.

PROBLEMA

Repetir código funciona, pero escala mal

JS

```
let nombre1 = "Ana";
console.log("Hola " + nombre1);

let nombre2 = "Luis";
console.log("Hola " + nombre2);

let nombre3 = "María";
console.log("Hola " + nombre3);
```

Problema

La lógica está repetida. Si cambia el mensaje, hay que corregirlo en varios lugares.

1. Más difícil de leer.
2. Más fácil de romper.
3. Más costoso de mantener.

FUNCIÓN

Un bloque de código reutilizable

JS

```
function saludar() {
  console.log("Hola, bienvenida al sistema.");
}

saludar();
```

- Una función tiene un nombre.
- Encierra instrucciones entre llaves.
- No se ejecuta sola: hay que llamarla.
- Permite evitar repeticiones.

Definir no es lo mismo que ejecutar

ESTRUCTURA

Cómo se escribe una función

JS

```
function mostrarMensaje() {
  console.log("El sistema está funcionando.");
}

mostrarMensaje();
```

function	Palabra reservada para declarar la función.
nombre	Debe expresar qué hace la función.
{ }	Bloque de instrucciones.
()	Se usan para ejecutar la función.

PARÁMETROS

La misma función, distintos datos

JS

```
function saludar(nombre) {
  console.log("Hola " + nombre);
}

saludar("Ana");
saludar("Luis");
saludar("María");
```

- **nombre** es el parámetro.
- **"Ana"**, **"Luis"** y **"María"** son argumentos.
- La lógica se escribe una vez y se reutiliza.

DOS PARÁMETROS

Una función también puede calcular

JS

```
function calcularTotal(precio, cantidad) {
  let total = precio * cantidad;
  console.log("El total es: $" + total);
}

calcularTotal(1000, 3);
```

Lectura

La función recibe precio y cantidad, multiplica ambos valores y muestra el total.

1. Entrada: datos.
2. Proceso: operación.
3. Salida: resultado.

RETURN

Mostrar no es lo mismo que devolver

console.log()

Muestra un dato en la consola. Sirve para revisar o depurar.

JS

```
function sumar(a, b) {
  console.log(a + b);
}

let resultado = sumar(5, 3);
console.log(resultado);
```

return

Devuelve un valor para usarlo después.

JS

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

let resultado = sumar(5, 3);
console.log(resultado);
```

APLICACIÓN

Calcular precio final con descuento

JS

```
function calcularPrecioFinal(precio, descuento) {
  let rebaja = precio * descuento / 100;
  let precioFinal = precio - rebaja;
  return precioFinal;
}

let total = calcularPrecioFinal(10000, 20);
console.log(total);
```

Este tipo de función puede aparecer en:

- Tiendas online.
- Carritos de compra.
- Landing pages con promociones.
- Formularios de presupuesto.

MINI EJERCICIO

Crear una función calcularEdad

Consigna

Crear una función llamada **calcularEdad**.

1. Debe recibir año de nacimiento.
2. Debe recibir año actual.
3. Debe devolver la edad aproximada.
4. Probarla con el año 2000 y el año 2026.

JS ESPERADO

```
let edad = calcularEdad(2000, 2026);
console.log(edad);
```

RESOLUCIÓN

Función calcularEdad

JS

```
function calcularEdad(anioNacimiento, anioActual) {
  let edad = anioActual - anioNacimiento;
  return edad;
}

let edad = calcularEdad(2000, 2026);
console.log(edad);
```

- La función recibe dos datos.
- Calcula la diferencia.
- Devuelve el resultado.
- El resultado se puede guardar en una variable.

EVENTOS

Un evento es algo que ocurre en la página

click

El usuario presiona un botón o elemento.

input

El usuario escribe en un campo.

submit

El usuario envía un formulario.

JavaScript puede escuchar esos eventos y ejecutar código como respuesta.

PRIMER EVENTO

Botón que cambia un mensaje

HTML + JS

```
<button id="botonSaludo">Saludar</button>
<p id="mensaje"></p>

<script>
let boton = document.getElementById("botonSaludo");
let mensaje = document.getElementById("mensaje");

boton.addEventListener("click", function() {
  mensaje.textContent = "Hola, hiciste clic en el botón.";
});
</script>
```

- Se selecciona el botón.
- Se selecciona el párrafo.
- Se escucha el evento **click**.
- Se cambia el texto del mensaje.

SELECCIÓN

getElementById busca un elemento por su id

HTML

HTML

```
<button id="botonSaludo">Saludar</button>
```

JavaScript

JS

```
let boton = document.getElementById("botonSaludo");
console.log(boton);
```

Si el id no coincide exactamente, JavaScript no encuentra el elemento.

ESCUCHAR EVENTOS

addEventListener conecta una acción con una función

JS

```
boton.addEventListener("click", function() {
  mensaje.textContent = "Hola, hiciste clic en el botón.";
});
```

1
Usuario hace clic

2
JavaScript escucha

3
Se ejecuta una función

4
La página cambia

Patrón central

DOM

El navegador convierte el HTML en algo manipulable

- DOM significa **Document Object Model**.
- Es una representación de la página creada por el navegador.
- JavaScript puede usar esa representación para cambiar la interfaz.

Leer

Obtener datos de elementos HTML.

Cambiar

Modificar textos, clases y valores.

Responder

Actualizar la página ante acciones del usuario.

DOM = puente entre HTML y JavaScript

CAMBIAR TEXTO

textContent modifica el texto interno

HTML + JS

```
<h1 id="titulo">Bienvenido</h1>
<button id="boton">Cambiar título</button>

<script>
let titulo = document.getElementById("titulo");
let boton = document.getElementById("boton");

boton.addEventListener("click", function() {
  titulo.textContent = "Bienvenido al sistema";
});
</script>
```

- Sirve para etiquetas como **p**, **h1**, **span** o **div**.
- No se usa para leer lo escrito en inputs.

LEER DATOS

value lee lo que el usuario escribió

HTML + JS

```
<input id="nombre" type="text" placeholder="Escribí tu nombre">
<button id="boton">Mostrar saludo</button>
<p id="saludo"></p>

<script>
let nombre = document.getElementById("nombre");
let boton = document.getElementById("boton");
let saludo = document.getElementById("saludo");

boton.addEventListener("click", function() {
  saludo.textContent = "Hola " + nombre.value;
});
</script>
```

Regla simple

textContent cambia texto de elementos comunes.

value lee campos de formulario.

ESTADOS VISUALES

Mejor cambiar clases que estilos sueltos

Menos recomendable

JS

```
mensaje.style.backgroundColor = "yellow";
mensaje.style.padding = "10px";
mensaje.style.fontWeight = "bold";
```

Más ordenado

CSS + JS

```
<style>
.destacado {
  background-color: yellow;
  padding: 10px;
  font-weight: bold;
}
</style>

<p id="mensaje">Mensaje importante</p>
<button onclick="mensaje.classList.add('destacado')">Destacar</button>
```

CLASSLIST

Agregar, quitar, alternar y consultar clases

add `elemento.classList.add("activo")` agrega una clase.

remove `elemento.classList.remove("activo")` quita una clase.

toggle `elemento.classList.toggle("activo")` agrega o quita según el estado.

contains `elemento.classList.contains("activo")` consulta si la clase existe.

EJEMPLO

Mostrar y ocultar información

HTML + CSS + JS

```
<style>
.oculto { display: none; }
</style>

<button id="botonInfo">Ver información</button>
<div id="info" class="oculto">
  <p>Esta es información adicional.</p>
</div>

<script>
let botonInfo = document.getElementById("botonInfo");
let info = document.getElementById("info");

botonInfo.addEventListener("click", function() {
  info.classList.toggle("oculto");
});
</script>
```

- La clase **oculto** define el estado visual.
- JavaScript solo alterna esa clase.
- El mismo patrón sirve para menús, modales y secciones desplegables.

MEJORA

El botón también puede acompañar el estado

HTML + JS

```
<style>.oculto { display: none; }</style>

<button id="botonInfo">Ver información</button>
<div id="info" class="oculto">Información adicional</div>

<script>
let botonInfo = document.getElementById("botonInfo");
let info = document.getElementById("info");

botonInfo.addEventListener("click", function() {
  info.classList.toggle("oculto");

  if (info.classList.contains("oculto")) {
    botonInfo.textContent = "Ver información";
  } else {
    botonInfo.textContent = "Ocultar información";
  }
});
</script>
```

Lectura

La interfaz no solo cambia: también comunica claramente en qué estado se encuentra.

EVENTO INPUT

Responder mientras el usuario escribe

HTML + JS

```
<input id="nombre" type="text" placeholder="Escribí tu nombre">
<p id="saludo"></p>

<script>
let nombre = document.getElementById("nombre");
let saludo = document.getElementById("saludo");

nombre.addEventListener("input", function() {
  saludo.textContent = "Hola " + nombre.value;
});
</script>
```

- El evento se dispara cada vez que cambia el valor del input.
- Sirve para búsquedas, contadores, filtros y previsualizaciones.

EVENTO SUBMIT

Controlar el envío de un formulario

HTML + JS

```
<form id="formulario">
  <input id="email" type="email" placeholder="Ingresá tu email">
  <button type="submit">Enviar</button>
</form>
<p id="mensaje"></p>

<script>
let formulario = document.getElementById("formulario");
let email = document.getElementById("email");
let mensaje = document.getElementById("mensaje");

formulario.addEventListener("submit", function(evento) {
  evento.preventDefault();
  mensaje.textContent = "Email ingresado: " + email.value;
});
</script>
```

- **submit** escucha el envío del formulario.
- **preventDefault()** evita la recarga automática.
- Después podemos validar antes de continuar.

VALIDACIÓN BÁSICA

Detectar un campo vacío

HTML + JS

```
<form id="formulario">
  <input id="nombre" type="text" placeholder="Nombre">
  <button type="submit">Enviar</button>
</form>
<p id="mensaje"></p>

<script>
let formulario = document.getElementById("formulario");
let nombre = document.getElementById("nombre");
let mensaje = document.getElementById("mensaje");

formulario.addEventListener("submit", function(evento) {
  evento.preventDefault();

  if (nombre.value.trim() === "") {
    mensaje.textContent = "El nombre es obligatorio.";
    return;
  }

  mensaje.textContent = "Formulario enviado correctamente.";
});
```

Por qué trim()

El valor **" "** no está vacío técnicamente, pero no es un nombre válido. **trim()** elimina espacios al principio y al final.

MEJORAR EL CÓDIGO

Separar la validación en una función

JS

```
function campoVacio(valor) {
  return valor.trim() === "";
}

let nombre = "   ";

if (campoVacio(nombre)) {
  console.log("El campo está vacío.");
} else {
  console.log("El campo tiene datos.");
}
```

- La lógica queda centralizada.
- El código se lee mejor.
- Podemos reutilizar la función para nombre, email, curso u otros campos.

ACTIVIDAD INTEGRADORA

Mini interfaz de inscripción a curso

Consigna

Construir una interfaz simple que permita cargar una inscripción.

1. Campo para nombre.
2. Campo para email.
3. Selector de curso.
4. Botón para confirmar.
5. Mensaje de error si falta algún dato.
6. Mensaje de confirmación si está todo correcto.

La idea es integrar funciones, eventos, DOM, value, textContent, classList y preventDefault.

CÓDIGO BASE

HTML de la inscripción

HTML

```
<section class="contenedor">
  <h1>Inscripción a curso</h1>

  <form id="formulario">
    <label for="nombre">Nombre</label>
    <input id="nombre" type="text">

    <label for="email">Email</label>
    <input id="email" type="email">

    <label for="curso">Curso</label>
    <select id="curso">
      <option value="">Seleccionar curso</option>
      <option value="HTML y CSS">HTML y CSS</option>
      <option value="JavaScript inicial">JavaScript inicial</option>
      <option value="Front-End completo">Front-End completo</option>
    </select>

    <button type="submit">Confirmar inscripción</button>
  </form>
```

Estructura del formulario

CÓDIGO BASE

CSS de la inscripción

CSS

```css
body {
  font-family: Arial, sans-serif;
  background-color: #f3f5f7;
  margin: 0;
  padding: 30px;
}

.contenedor {
  max-width: 500px;
  margin: auto;
  background-color: white;
  padding: 24px;
  border-radius: 14px;
}

label {
  display: block;
  margin-top: 14px;
  font-weight: bold;
}
```

CÓDIGO BASE

JavaScript de la inscripción

JS

```
let formulario = document.getElementById("formulario");
let nombre = document.getElementById("nombre");
let email = document.getElementById("email");
let curso = document.getElementById("curso");
let mensaje = document.getElementById("mensaje");

function campoVacio(valor) {
  return valor.trim() === "";
}

formulario.addEventListener("submit", function(evento) {
  evento.preventDefault();

  mensaje.className = "";

  if (campoVacio(nombre.value)) {
```

Validación completa del caso

ANÁLISIS

Qué conceptos aparecen en el ejercicio

DOM	Seleccionamos formulario, inputs, select y mensaje con `getElementById`.
Evento	Escuchamos el `submit` del formulario.
Función	Usamos `campoVacio` para evitar repetir lógica.
Validación	Si falta un dato, mostramos error y cortamos con `return`.
CSS	Usamos `classList.add` para marcar error o éxito.

PRÁCTICA

Mejoras para resolver en clase

Agregar validaciones

1. Que el nombre tenga al menos 3 caracteres.
2. Que el email contenga **@**.
3. Que el botón cambie a “Inscripción enviada” cuando el formulario sea válido.
4. Que el formulario se limpie después de confirmar.

AYUDA

```
function emailValido(valor) {
  return valor.includes("@");
}

console.log(emailValido("alumno@mail.com"));
console.log(emailValido("alumno.mail.com"));
```

PATRÓN HABITUAL

La mayoría de las interacciones siguen la misma secuencia

1
Seleccionar elementos del HTML

2
Escuchar un evento

3
Ejecutar una función

4
Modificar el DOM

JS

```
let boton = document.getElementById("boton");

boton.addEventListener("click", function() {
  // cambiar algo en la página
});
```

Esto se repite en botones, menús, formularios y modales

ERRORES COMUNES

Qué revisar cuando “no funciona”

id distinto	El id del HTML y el usado en JavaScript no coinciden exactamente.
script antes	El JS se ejecuta antes de que exista el HTML. Solución simple: script al final del body.
value/textContent	En inputs se usa `value`; en textos comunes se usa `textContent`.
submit	Si falta `preventDefault()`, el formulario puede recargar la página.
return	Si una función no devuelve nada, el resultado será `undefined`.

PUENTE

Esto prepara la integración final

- Hoy conectamos JavaScript con eventos y elementos HTML.
- La próxima clase se profundiza DOM y validación de formularios.
- La lógica del trabajo práctico debería apoyarse en este mismo patrón.
- El parcial no busca magia: busca entender estructura, estilos e interacción básica.

Idea clave

Una interfaz simple pero clara vale más que una página cargada de efectos mal entendidos.

CIERRE

Qué tiene que quedar claro

Conceptos clave

- Una función organiza código.
- Un parámetro permite reutilizar lógica con distintos datos.
- `return` devuelve valores.
- Un evento representa una acción del usuario.
- El DOM permite modificar la página.
- `value`, `textContent` y `classList` son herramientas básicas de interacción.

El usuario hace algo. JavaScript escucha. Se ejecuta una función. La página cambia.

Próxima clase: DOM, validación de formularios e integración final.