SEMINARIO DE LENGUAJES

CLASE 4

JAVASCRIPT Y HTML: INTERACTIVIDAD Y CONTROL A TRAVÉS DE EVENTOS

AGENDA

1. Repaso De La Clase Anterior
2. Programación Orientada A Eventos
3. Funciones
4. Eventos Y Funciones

REPASO DE LA CLASE ANTERIOR

DOM: DOCUMENT OBJECT MODEL [1]

Es un estándar de W3C.

Representa la estructura del documento HTML (página Web) en memoria.

Funciona de interfaz para documentos HTML y XML.

De esta manera es posible manipularla a través de scripts.

DOM: DOCUMENT OBJECT MODEL [2]

Es una estructura jerárquica de nodos en forma de árbol.

Cada rama termina en nodos que contienen los objetos.

El primer elemento se lo llama `document`.

Se puede acceder a los elementos por medio de el id, clase o nombre de etiqueta.

PROGRAMACIÓN ORIENTADA A EVENTOS

Los eventos son notificaciones que se generan cuando ocurre algo significativo, como un click del mouse, una pulsación de tecla, o la carga completa de una página web.

EJEMPLOS DE EVENTOS

- Se carga la página web: `load`
- Se hace click sobre un elemento: `click`
- Se modifica un input field: `change`
- Se carga una imagen: `load`
- Se envía un formulario: `submit`
- El mouse se mueve sobre un elemento: `mouseover`
- El usuario presiona una tecla: `keydown`
- Se pone el foco sobre un elemento: `focus`

ORIGEN DE LOS EVENTOS

El sistema se encarga de producir una señal de cierto tipo cuando un evento ocurre, y proporciona un mecanismo para que una acción se lleve a cabo.

MANEJO DE EVENTOS

JavaScript permite "escuchar" estos eventos utilizando `event listeners`, que pueden ejecutar código en respuesta a acciones específicas llamados `event handlers`.

```
const boton = document.getElementById('miBoton')
boton.addEventListener("click", function() {
  alert("¡Hiciste click en el botón!")
})
```

IMPORTANCIA DE LOS EVENTOS

Los eventos son fundamentales para la interactividad en la Web.

El manejo de eventos es una parte esencial de la programación en JavaScript.

Permiten a los usuarios interactuar con la página, y a los desarrolladores crear aplicaciones web dinámicas y reactivas.

ELEMENTOS Y EVENTOS

Hay diversas maneras de asociar un elemento del DOM a un evento:

- Atributos de Evento HTML
- Propiedades del Objeto DOM
- addEventListener

ATRIBUTOS DE EVENTO HTML

Incorpora el event handler directamente en el HTML.

```
<button onclick='alert("¡Hiciste click!")'>Click aquí</button>
```

- Pros: Simple y directo para scripts pequeños.
- Contras: Mezcla código JavaScript con HTML, lo cual es menos ideal para mantenimiento y escalabilidad.

PROPIEDADES DEL OBJETO DOM

Asigna una función de JavaScript a una propiedad de evento de un elemento del DOM.

```
const boton = document.getElementById('miBoton')
boton.onclick = function() { alert("¡Hiciste click!") }
```

- Pros: Más organizado que los atributos HTML, fácil de entender.
- Contras: Sólo permite un manejador de evento por elemento y tipo de evento.

ADDEVENTLISTENER

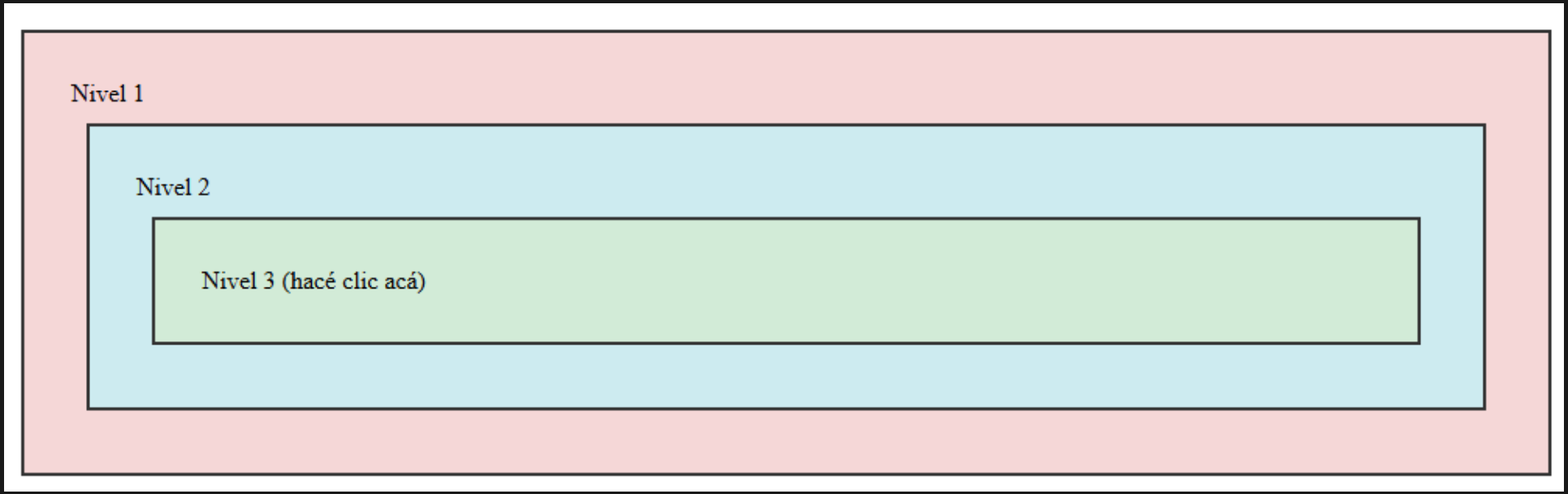
Método recomendado para añadir manejadores de eventos que permite más flexibilidad y control.

```
const boton = document.getElementById('miBoton')
boton.addEventListener('click', function() { alert('¡Hiciste click!'
```

- Pros: Permite múltiples manejadores para el mismo evento en un solo elemento, control sobre la propagación de eventos.
- Contras: Sintaxis ligeramente más compleja.

PROPAGACIÓN DE EVENTOS

CAPTURA Y BURBUJEO



PROPAGACIÓN DE EVENTOS

- event.stopPropagation(): detiene la propagación a otros elementos.
- event.stopImmediatePropagation(): detiene todo, incluso otros listeners en el mismo elemento.
- event.preventDefault() → cancela la acción predeterminada (como seguir un link).

FUNCIONES

CONTEXTO: THIS [1]

`this` es una palabra clave que tiene un comportamiento especial y su valor es determinado por cómo es llamada la función en la que se utiliza.

`this` se refiere al objeto que es el "contexto actual" de la ejecución del código, y puede variar considerablemente dependiendo del modo en que la función es llamada.

CONTEXTO: THIS [2]

```
function whoIs(){
  console.log(this);
}
whoIs(); // Window

const persona = {
  nombre: 'Luke',
  saludar: function() {
    console.log(this.nombre)
  }
}
persona.saludar() // Luke
```

FUNCIONES: DECLARATIVAS

Son las funciones que vimos hasta ahora, se declaran con `function nombre(params){/* código */}`.

```
function suma(n1, n2) {
  return n1 + n2
}
const valor = suma(1, 2)
```

FUNCIONES: EXPRESIÓN

Se crean declarando una función anónima o una asignada a una variable, no utilizan el nombre de la función en la declaración.

```
const suma = function(n1, n2) {
  return n1 + n2
}
const valor = suma(1, 2)

const boton = document.getElementById('miBoton')
boton.addEventListener('click', function() { alert('¡Hiciste click!'
```

DECLARACIÓN VS EXPRESIÓN [1]

/* El hoisting es el comportamiento de JavaScript de "elevar" (mover hacia arriba) las declaraciones de variables y funciones al comienzo del contexto de ejecución (scope), antes de que se ejecute el código. */

Características	Declarativa	Expresión
Hoisting	Fáciles de entender y mantener	Pueden ser menos intuitivas
Recursividad	Facilidad para llamarse a sí mismas	No aplicable directamente, necesita ser

asignada a
una variable

DECLARACIÓN VS EXPRESIÓN [2]

Características	Declarativa	Expresión
Expresiones	Limitadas; no se usan como expresiones directas	Flexibles; ideales para usar como callbacks o asignaciones
Legibilidad	Pueden causar errores por el hoisting	Evita errores de sobreescritura y uso prematuro

FUNCIONES FLECHA [1]

Son funciones fechas son de expresión, y proporcionan una sintaxis más corta y clara en comparación con las funciones tradicionales:

```
const sumaFlecha = (a, b) => a + b

const sumaFlecha2 = (a, b) => {
  return a + b
}
```

FUNCIONES FLECHA [2]

No tienen la palabra clave function.

No necesitan llaves en el caso de que la función sea de una sola expresión.

Capturan el valor de `this` del entorno en el que fueron creadas.

FUNCIONES FLECHA [3]

Las funciones flecha son ideales para:

- Callbacks en métodos como map, filter, y reduce.
- Funciones cortas que se pasan como argumentos a otras funciones.
- Métodos donde el uso de `this` debe referirse al contexto circundante y no al propio método.

FUNCIONES FLECHA: COMPARACIÓN DE THIS

```
const persona = {
  nombre: 'Juan',
  consoleThis: function() {
    console.log(this)
  },
  consoleThisFlecha: () => console.log(this)
}
persona.consoleThis() // {nombre: "Juan", consoleThis: ƒ, consoleThi
persona.consoleThisFlecha() // Window
```

Como en funciones flecha, `this` es el contexto de dónde se llamó, siendo el ambito global, es Window.

EVENTOS Y FUNCIONES

Los eventos listeners están fuertemente relacionado a las funciones, ya que en ellas se encuentra el comportamiento ante el suceso del evento.

Todo evento tiene asociada una función, ya sea declarativa y expresiva (inclusive flecha).

EVENTOS Y FUNCIONES: EJEMPLOS [1]

```
function handleClick() {
  console.log('Botón clickeado!')
}
const boton = document.getElementById('miBoton')
boton.addEventListener('click', handleClick)
```

EVENTOS Y FUNCIONES: EJEMPLOS 2

```
const boton = document.getElementById('miBoton')
boton.addEventListener('click', function() {
  console.log('Botón clickeado!')
})
```

EVENTOS Y FUNCIONES: EJEMPLOS 3

```
const boton = document.getElementById('miBoton')
boton.addEventListener('click', () => console.log('Botón clickeado!'
```

FIN DE LA CLASE