

# Parte 1:

## Introducción al DOM y al uso de eventos

### *(Document Object Model)*

UD3: Interacción con el usuario y modelo de objetos del documento

María Rodríguez Fernández [mariarfer@educastur.org](mailto:mariarfer@educastur.org)

# Al final de este documento...

- Conocerás que es el DOM y sus funciones básicas
  - Más adelante dedicaremos una unidad didáctica a profundizar en este tema
- Empezaremos a usar el DOM para hacer ejercicios más “cercanos” al uso real de JavaScript – trabajar con HTML
- Conocerás los **principales eventos** que pueden desencadenarse al interactuar con una página web
- Sabrás codificar una gestión de eventos usando el **registro de eventos avanzado**, que es el más recomendado y compatible

# DOM (*Document Object Model*)

- Habitualmente utilizamos JavaScript para acceder al contenido de la página Web
- Permite acceder a una estructura de datos que representa (en forma de árbol) todo el contenido de una página Web y realizar operaciones de:
  - Lectura
  - Modificación
  - Inserción / eliminación de elementos

# Objeto document

- Representa el documento cargado en una ventana del navegador
- Permite el **acceso a todas las etiquetas HTML** dentro de una página
- Forma parte del objeto **window**, luego puede ser accedido mediante **window.document** o directamente **document**

# Accediendo al DOM

## Métodos de `document`

<code>getElementById(id)</code>	Devuelve el elemento con id <b>id</b>
<code>getElementsByClassName(class)</code>	Devuelve un array de elementos que tienen clase <b>class</b>
<code>getElementsByTagName(tagname)</code>	Devuelve un array de elementos cuya etiqueta es <b>tagname</b>
<code>querySelector(query)</code>	Devuelve el primer elemento que concuerda con el grupo de <b>selectores CSS</b> especificados entre paréntesis
<code>querySelectorAll(query)</code>	Devuelve un array de elementos que concuerdan con el grupo de <b>selectores CSS</b> especificados entre paréntesis

Síven para todos los casos

# Cambiar el contenido

## Es posible consultar y modificar

<code>elemento.innerHTML = valor</code>	Contenido de cualquier etiqueta HTML.
<code>elemento.innerText = valor</code>	Contenido de texto puro – aplica estilos CSS
<code>elemento.textContent = valor</code>	Contenido de texto puro – sin etiquetas.
<code>elemento.atributo = valor</code> <code>elemento.setAttribute(valor)</code>	Dos formas para modificar el contenido de un atributo.

## Se desaconsejan, más ineficientes

<code>open()</code>	Abre el flujo de escritura para poder usar <b>write()</b>
<code>close()</code>	Cierra el flujo de escritura abierto con <b>open()</b>
<code>write(cadena)</code> <code>writeln(cadena)</code>	Permite escribir expresiones HTML dentro de un documento (sin o con salto de línea al final)

# EJERCICIO PROPUESTO I

## Calendario



- Realiza un script que muestre la **fecha actual**
  - Utiliza una hoja de estilos que vincularás desde la página HTML para mejorar la apariencia del calendario:

```
<link rel="stylesheet" href="./css/estilos.css">
```



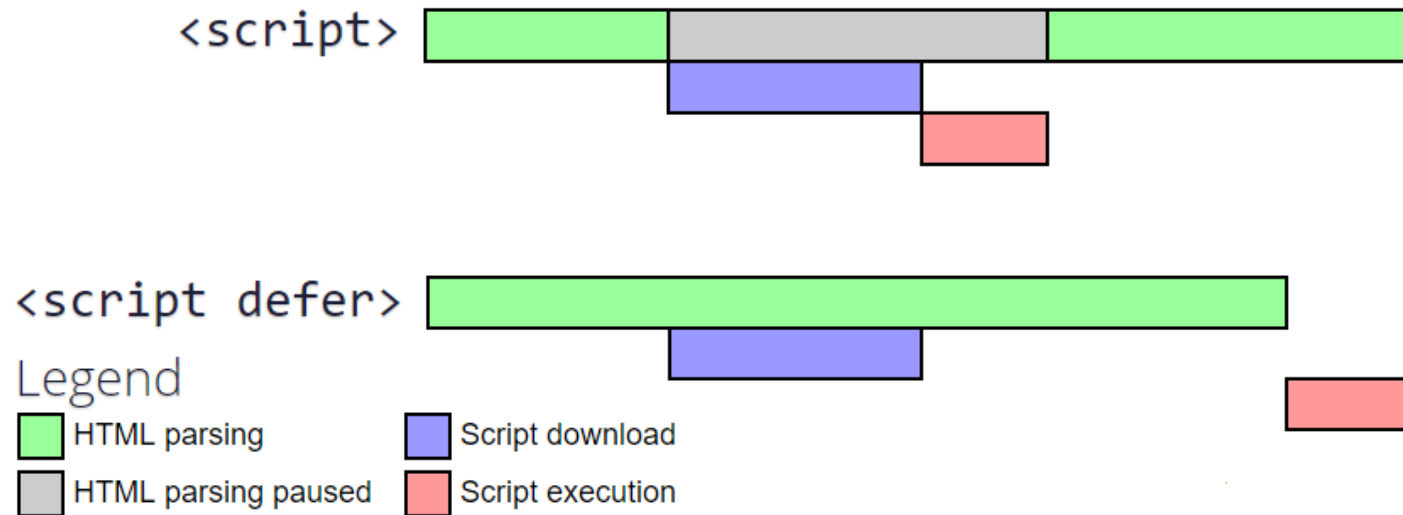
# Ten en cuenta...

- Ten en cuenta...
  - Para acceder al DOM la página tiene que haber terminado de cargarse
  - OPCIONES:
    - Usar eventos: Luego hablamos de ello.
    - Palabra reservada **defer**

```
<script defer src="scripts/calendario.js">
```



# Funcionamiento de defer



# Eventos

- Los scripts mostrados hasta ahora se ejecutan secuencialmente
  - Con las estructuras de control de flujo y funciones cambiamos dicho comportamiento ligeramente
- Con JavaScript podemos utilizar un modelo de funcionamiento **basado en eventos**
  - Los scripts se ejecutan una vez que el usuario haya “*hecho algo*” (pulsar un botón, mover el ratón, cerrar la ventana..), es decir, cuando se produzca un **evento**
  - Puedo asignar una función a cada uno de dichos eventos. Este tipo de funciones se llaman **manejadores de eventos**

# Modelos de eventos

- La mayor parte de **incompatibilidades** entre navegadores se produce en el modelo de eventos
  - El **modelo básico de eventos** es el que cumplen la mayoría
- Cada elemento HTML define su propia lista de **posibles eventos** que se le pueden asignar
  - Un mismo evento (ej: pulsar el ratón) puede estar definido para varios elementos HTML distintos, y un elemento puede tener asociados varios eventos distintos

# Principales eventos I

Evento	Descripción	Elementos para los que está definido
<u>click</u>	Pinchar y soltar el ratón	Todos los elementos
dblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
focus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
keydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
keypress	Pulsar una tecla	Elementos de formulario y <body>
keyup	Soltar una tecla pulsada	Elementos de formulario y <body>
<u>load</u>	La página se ha cargado completamente	<body>
mousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
blur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
change	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>

# Principales eventos II

Evento	Descripción	Elementos para los que está definido
mousemove	Mover el ratón	Todos los elementos
mouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
mouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
mouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
reset	Inicializar el formulario (borrar todos sus datos)	<form>
resize	Se ha modificado el tamaño de la ventana del navegador	<body>
select	Seleccionar un texto	<input>, <textarea>
submit	Enviar el formulario	<form>
unload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

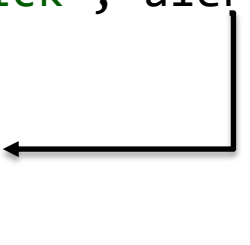
# Manejadores de eventos

- Contienen el **código que se ejecutará** cuando sucede un evento
- Existen varias maneras de indicar los manejadores de eventos
  - El W3C propone una manera de registrar los eventos sobre cualquier objeto (“Registro de eventos avanzado”) usando el método **addEventListener()**:
    - **addEventListener(tipo, manejador)**
    - **removeEventListener(tipo, manejador)** para eliminarlo

# Ejemplo

- Definiendo el comportamiento en una función:

```
document.getElementById("enlace").addEventListener('click', alertar);  
  
function alertar(){  
  alert("Te conectaremos con la página: "+this.href);  
}
```



- Usando funciones anónimas:

```
element.addEventListener('click', function () {  
  this.style.backgroundColor = '#cc0000';  
})
```



Función  
manejadora

# En el calendario...

- Para que el script se ejecute al cargar la página (alternativa al defer):

```
window.addEventListener("load", principal);  
  
function principal(){  
  //esto se ejecuta al cargar la página  
}
```

- Aunque también es muy habitual:

```
window.onload = principal;  
  
function principal(){  
  //esto se ejecuta al cargar la página  
}
```



# EJERCICIO PROPUESTO-II: Minion



- A partir de la siguiente página cuyo html y css se te proporciona implementar cuatro funcionalidades:



- **Botón 1:** Al hacer click sobre él cambia su propio texto por “Ay, me pinchaste”

# Información del evento: Event

- En muchos casos, los **manejadores de eventos** requieren de **información adicional**:
  - Qué tecla he pulsado
  - Qué botón del ratón he tocado
  - Qué evento se ha producido
  - Dónde se ha desencadenado el evento
  - Etc...
- Dicha información se obtiene mediante la clase **Event**



```
function manejadorEventos(e:Event){  
  //puedo acceder al parámetro e:Event  
}
```

Los manejadores de eventos reciben de forma implícita una instancia de un objeto de este tipo

# Propiedad type

- Propiedad **type** (Clase Event)
  - Devuelve el tipo de evento producido

```
document.querySelector("#contenidos").addEventListener("mouseover",resalta);  
document.querySelector("#contenidos").addEventListener("mouseout",resalta);
```

```
function resalta(eEvento){  
  switch(eEvento.type){  
    case "mouseover":  
      this.className="estilo1",  
      break;  
    case "mouseout":  
      this.className="estilo2",  
      break;  
  }  
}
```

La función manejadora  
puede recibir como  
parámetro el evento en sí  
(no importa el nombre)

# Propiedad target

- Propiedad **target**: Distinguir sobre qué elemento ha ocurrido:

```
document.getElementById("parrafo1").addEventListener("click", saludo);
document.getElementById("parrafo2").addEventListener("click", saludo);


function saludo(e) {
    if (e.target.id == "parrafo1")
        alert("Has pulsado el primer párrafo");
    else if (e.target.id == "parrafo2")
        alert("Has pulsado el segundo párrafo");
    alert("Has pulsado el " + e.target.id);
}
```

# Ten en cuenta...

- El elemento anidado más profundo que causó el evento es llamado elemento objetivo, accesible como **`event.target`**
- Nota la diferencia:
  - `event.target` – es el elemento “objetivo” que inició el evento, no cambia a través de todo el proceso de propagación
  - `this` (`=event.currentTarget`) – es el elemento “actual”, el que tiene un manejador ejecutándose en el momento

# EJERCICIO PROPUESTO-II: Minion



- **Botón 1:** Modifícalo si no habías usando `e.target`
- **Botón 2:** Al pasar el ratón por encima, muestra por consola información sobre el evento
- **Botón 3:** Coge la información de la caja de texto “nombre”, y hace un saludo personalizado en la caja 4.
- **RETO:** Dibújale un  al Minion



# EJERCICIO PROPUESTO-III



- Tenemos una página HTML con sus estilos CSS, con N fotos de perros y gatos, al lado de cada una de las cuales hay una etiqueta, además de haber otra etiqueta general.
- Queremos que:
  - Al pulsar sobre cada una de las imágenes la etiqueta muestre el número de pulsaciones sobre su imagen correspondiente.
  - La etiqueta general muestre el nombre del último animal sobre el cual hemos pinchado.



# Eventos de teclado



- Cuando un usuario pulsa una tecla se producen tres eventos de teclado:
  - **keydown/keyup**
    - Pulsar/levantar una **tecla**
  - **keypress**
    - Cuando la tecla produce un valor
      - Ciertos caracteres van asociados a combinaciones de varias teclas
        - » Por ejemplo, si pulsamos Alt + e (€), solo se desencadena 1 vez el evento **keypress**



# Detectando la tecla pulsada

## [Propiedades `code` y `key`]

```
document.body.addEventListener("keydown", pulsaTecla);
```

- Propiedad **code** [Clase **Event**]
  - Representa la tecla física en el teclado
- Propiedad **key** [Clase **Event**]
  - Representa el carácter pulsado
    - Si es imprimible, se devuelve el Unicode
    - Si no es imprimible (control) devuelve un [valor predefinido](#)

```
function pulsaTecla(e) {  
  var codigo=e.code;  
  var tecla=e.key;  
  console.log("Código: "+codigo+ ". Tecla: "+tecla);  
  if(codigo==="Enter")  
    console.log("Pulsé el Enter");  
  else if(codigo==="Space")  
    console.log("Pulsé el espacio");  
}
```

# Eventos de ratón (aparte de click)



- **mousedown/mouseup**
  - Se oprime/suelta el botón del ratón sobre un elemento
- **mouseover/mouseout**
  - El puntero del mouse se mueve sobre/sale de un elemento
- **mousemove**
  - Cualquier movimiento del mouse sobre un elemento activa el evento

# Detectando la tecla pulsada y posición

- Tecla pulsada:
  - Propiedad **evento.button**
    - » 0: Izquierdo
    - » 1: Derecho
    - » 2: Central
- **Coordenadas** del puntero desde la esquina superior izquierda
  - Respecto de la pantalla del ordenador.
    - Propiedades **evento.screenX** y **evento.screenY**
  - Respecto de la ventana del navegador
    - Propiedades **evento.clientX** y **evento.clientY**
  - Respecto de la página HTML
    - Propiedades **evento.pageX** y **evento.pageY**

# ¿Cómo ha ido?

