PROGRAMACIÓN WEB – JS-II

Lenguaje de programación Javascript

JavaScript

Dr. José Raúl Romero Salguero
jrromero@uco.es

Contenidos

1. Aspectos básicos
2. Constructores básicos
3. Objetos en JavaScript
4. DOM
5. Alertas y validación
6. Programación basada en eventos con JavaScript

4.
DOM

Objetos globales en DOM

Los seis objetos DOM globales

Cada programa Javascript puede referirse a los siguientes objetos globales:

BOM
Browser Object Model

nombre	descripción
`document`	página HTML actual y su contenido
`history`	lista de páginas que el usuario ha visitado
`location`	URL de la página HTML actual
`navigator`	información sobre el navegador web que está utilizando
`screen`	información sobre el área de la pantalla ocupada por el navegador
`window`	la ventana del navegador

El objeto `document`

Es el documento actual y punto de entrada a los elementos del DOM (*Document Object Model*)

- Propiedades:
  - `anchors, body, cookie, domain, forms, images, links, referrer, title, URL`
- Métodos:
  - `getElementById` → `element`
  - `getElementsByName, getElementsByTagName`
  - `querySelector, querySelectorAll`
  - `close, open, write, writeln`

El objeto `history`

Es la lista de sitios que el navegador ha visitado en esta ventana.

- Propiedades:
  - `length`
- Métodos:
  - `back, forward, go`

A veces –por seguridad– el navegador no permite que los *scripts* vean las propiedades del historial

https://developer.mozilla.org/es/docs/Web/API/History

El objeto `location`

Es la URL de la página web actual.

- Propiedades:
  - `host`, `hostname`, `href`, `pathname`, `port`, `protocol`, `search`
- Métodos:
  - `assign(url)` – carga un nuevo documento
  - `reload(Boolean)` – recarga la página del servidor (true) o de la cache (false) enviando un GET forzado
  - `replace(url)` – como `assign` pero eliminando la URL actual del historial

El objeto `location`

```
<html>
<head>
<script>
function newDoc() {
  window.location.assign("https://www.w3schools.com")
}
</script>
</head>
<body>

<input type="button" value="Load new document" onclick="newDoc()">

</body>
</html>
```

Fuente: w3schools.com

El objeto `navigator`

Propiedades del objeto `navigator`:

`navigator.appName`	Nombre del navegador
`navigator.appVersion`	Versión del navegador
`navigator.language`	Idioma del navegador
`navigator.cookiesEnabled`	Indica si tiene cookies activadas

```
<html>
<head>
<title>Navigator example</ title>
<script>
if (navigator.appName == 'Netscape') {
  document.writeln('<link rel=stylesheet type="text/css" href= "netscape.css">');
} else if (navigator.appName == 'Opera') {
  document.writeln('<link rel=stylesheet type="text/css" href="Opera.css">');
} else {
  document.writeln('<link rel=stylesheet type ="text/css" href="Other.css">');
}
</ script >
</ head >
...
```

El objeto `screen`

Es información sobre la pantalla de visualización del cliente

- Propiedades:
  - `height, width`
  - `availHeight, availWidth`
  - `colorDepth, pixelDepth`

El objeto `window`

Un objeto `window` representa una ventana abierta en un navegador

- Si un documento contiene `frames`, entonces hay
  - un objeto `window` para el documento HTML
  - un objeto `window` adicional para cada marco, accesible a través de una matriz `window.frames`
- Un objeto `window` tiene **propiedades** que incluyen:
  - `innerHeight`, `innerWidth` – altura, anchura interior de la ventana del navegador en px (**no** incluye barras de herramientas ni de *scrool*)

```
var w = window.innerWidth || document.documentElement.clientWidth
|| document.body.clientWidth;

var h = window.innerHeight || document.documentElement.clientHeight
|| document.body.clientHeight;
```

  - Multitud de eventos `window.on`…
  - Objetos `document`, `screen`, `navigator`, `location`, etc.

https://www.w3schools.com/jsref/met_win_open.asp

El objeto `window`

Los métodos proporcionados por un objeto `window` incluyen

- `open(url,name[,features])` abre una nueva ventana / pestaña del navegador y devuelve una referencia a un objeto de ventana
  - `url` es la URL para acceder en la nueva ventana; puede ser la cadena vacía ("about:blank")
  - `name` es un nombre dado a la ventana para referencia posterior (_blank, _parent, _self, _top, o un *nombre* para la ventana nueva)
  - `features` es una cadena (sin espacios) de características (atributo=valor) de la ventana:

    height=pixels, left=pixels, menubar=yes|no|1|0, scrollbars=yes|no|1|0, status=yes|no|1|0, titlebar=yes|no|1|0, top=pixels, width=pixels

NOTA: La secuencia estándar para la creación de una nueva ventana no es instanciar un nuevo objeto `window` con `new`, sino invocar a `window.open()`

El objeto `window`

Los métodos proporcionados por un objeto de ventana incluyen:

- `close()` – cierra una ventana / pestaña del navegador
- `focus()` – enfoca una ventana (acercar la ventana al frente)
- `blur()` – elimina el foco de una ventana (mueve la ventana detrás de otras)
- `scrollBy()`, `scrollTo()` – realiza un *scroll* de la ventana en ***x,y*** *px* (o desplaza a una posición dada la barra)

```
<!DOCTYPE html>
<html>
<body>

<p>Pulsar el bot&oacute;n para abrir una ventana y darle el foco.</p>

<button onclick="abrirVentana()">Pulsar aqu&iacute;</button>

<script>
function abrirVEntana() {
  var vent = window.open("", "", "width=200, height=100");
  vent.document.write("<p>Nueva ventana</p>");
  myWindow.focus(); //Usar blur() para no darle el foco
}
</script>

</body>
</html>
```

El objeto window__ Ejemplo

```
<html lang ="es-ES">
<head>
 <title>Lanzador de ventana de ayuda</title>
 <script>
function lanzarAyuda() {
 var msjAyuda = window.open('','Ayuda',
'height=300,width=150,menubar=no,scrollbars=no,status=no,titlebar=no,location=no,
toolbar=no');

 with (msjAyuda.document) {
  writeln ("<!DOCTYPE html><html><head><title> Help </title>\
            </head><body>Esto podr&iacute;a ser ayuda contextual, un mensaje u
otra alerta, seg&uacute;n el estado de la p&aacute;gina invocante.</body></html>");
  close();
 }
}
 </script>
</head>
<body>
 <form name ="ButtonForm" id ="ButtonForm" action ="">
   <p><input type ="button" value="Click for Help" onclick="Help();"></p>
 </form>
</body>
</html>
```

Exploración del árbol DOM

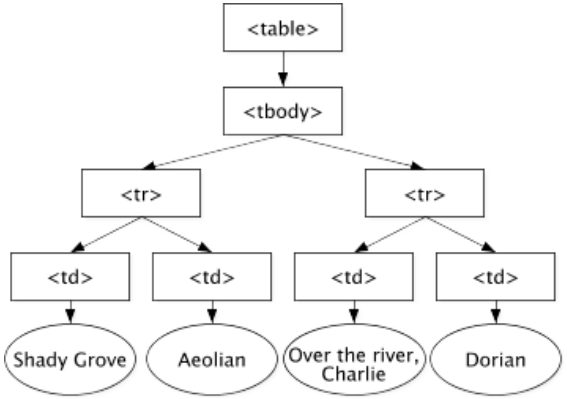
DOM__ Modelo de objetos del documento

Ejemplo:

La siguiente tabla HTML

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Over the River, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```

se analiza en el siguiente DOM



Arnaud Le Hors, et al, editors: Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Recommendation 07 April 2004. World Wide Web Consortium, 2004. https://www.w3.org/TR/DOM-Level-3-Core/

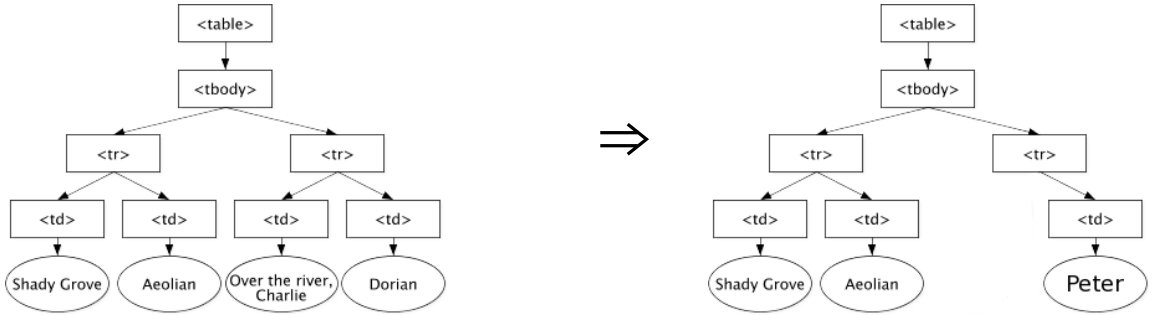
DOM__ Acceso a elementos HTML

```
// accede a elemento TBODY de la tabla
var myTbodyElement = myTableElement.firstChild;

// accede al segundo elemento TR; la lista de hijos empieza en 0
var mySecondTrElement = myTbodyElement.childNodes[1];

// elimina el primer elemento TD
mySecondTrElement.removeChild(mySecondTrElement.firstChild);

// Cambia el contenido del texto del elemento TD restante
mySecondTrElement.firstChild.firstChild.data = "Peter";
```



DOM__ Acceso a elementos HTML

Resulta más intuitivo el uso de nombres a los elementos de HTML, en lugar del acceso con métodos como `firstChild` y `childNodes[n]`

Ejemplo:

```
<form name="form1" action ="">
  <label>Temperatura en Farenheit:</label>
    <input type ="text" name="fahrenheit" size=10 value="0" /><br/>
  <label>Temperatura en Celsius:</label>
    <input type="text" name="celsius" size="10" value ="" />
</form>
```

- `document.form1` se refiere al formulario llamado 'form1'
- `document.form1.celsius` se refiere al campo de texto llamado 'celsius' en 'document.form1'
- `document.form1.celsius.value` se refiere al valor del atributo en el campo de texto llamado 'Celsius' en 'document.form1'

DOM__ Acceso a elementos HTML

La navegación por DOM implica saber los nombres y utilizar `paths` en su estructura de árbol , que puede cambiar → **¡Problemático!**

~ Si esa estructura de árbol cambia, los `paths` ya no funcionan.

Ejemplo: insertamos un DIV en el formulario anterior.

```
<form name="form1" action ="">
<div class="field" name="fdiv">
  <label>Temperatura en Farenheit:</label>
    <input type ="text" name="fahrenheit" size=10 value="0" /><br/>
  <label>Temperatura en Celsius:</label>
    <input type="text" name="celsius" size="10" value ="" />
</div>
</form>
```

La implicación es que `document.form1.celsius` ya no funciona, ya que ahora hay un elemento `DIV` entre formulario y campo de texto. Ahora, habría que utilizar `document.form1.cdiv.celsius`

DOM__ Acceso mediante ID

Una forma **más confiable** es dar a cada elemento HTML un ID (usando el atributo `id` de HTML) y usar un método de consulta DOM, como `getElementById` para recuperar el elemento por su ID

```
<form name="form1" action ="">
  <label>Temperatura en Farenheit:</label>
    <input type ="text" name="fahrenheit" size=10 value="0" /><br/>
  <label>Temperatura en Celsius:</label>
    <input type="text" name="celsius" size="10" value ="" />
</form>
```

- `document.getElementById('celsius')` se refiere al elemento HTML con ID “Celsius”
- `document.getElementById('celsius').value` se refiere al valor del atributo en el elemento con ID “celsius"

DOM__ Manipulando elementos HTML

No solo es posible **consultar** elementos HTML, sino también es posible **cambiar y manipular** sus valores

```
<html>
 <head>  <title>Manipulando elementos HTML</title>
   <style>
td.FondoRojo { background: #f00; }
   </style>  <script>
function cambiarFondo1(id) {
 var cell=document.getElementById(id);
 // No se recomienda acceder al atributo STYLE desde JS
 cell.style.background = "#00f";
 cell.style.color = "white";
 cell.innerHTML = "azul!";
}
function cambiarFondo2(cell) {
// Se asigna clase de CSS - recomendable
 cell.className = "FondoRojo";
 cell.innerHTML = "rojo!";
}
 </script></head>
<body>
 <table border="1"> <tr>
  <td id="elem0" onclick="cambiarFondo1('elem0');">blanco</td>
  <td id="elem1" onclick="cambiarFondo2(this);">blanco</td></tr>
 </table> </body> </html>
```

5.
Alertas y validación

Alertas

Cajas de diálogo del objeto `window`

A menudo solo queremos abrir una nueva ventana para:

- mostrar un mensaje
- pedir confirmación de una acción
- solicitar una entrada

Para estos fines, el objeto `window` en JS proporciona mecanismos predefinidos para el manejo de "`dialog boxes`" (cuadros de diálogo simples):

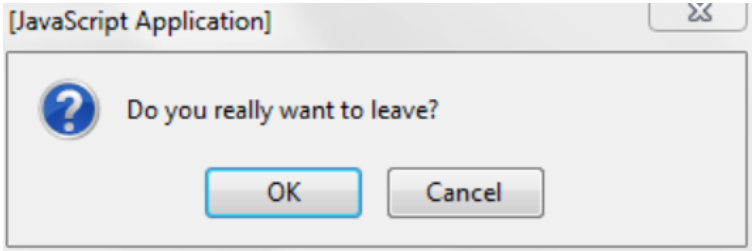
- `null alert(msg_str)`
- `bool confirm(msg_str)`
- `string prompt(msg_str, default_value)`

Cajas de diálogo del objeto `window`

Para estos fines, el objeto `window` en JS proporciona mecanismos predefinidos para el manejo de "`dialog boxes`" (cuadros de diálogo simples):

- `null alert(msg_str)`

- `bool confirm(msg_str)`



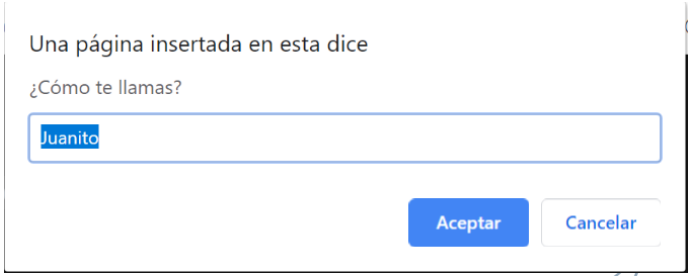
Cajas de diálogo del objeto `window`

`string prompt(msg_str, default_value)`

- Crea un cuadro de diálogo que muestra `msg_str` y un área de entrada
- Si se indica `default_value`, se mostrará en el campo de entrada
- Muestra dos botones 'Cancel' and 'OK'
- Si el usuario selecciona 'OK', el valor actual ingresado en el campo de entrada se devuelve como `String`; de lo contrario, se devuelve un valor `null`

Ejemplo:

```
var nombre = prompt("¿Cómo te llamas?",
"Juanito");
```



Cajas de diálogo del objeto `window`

- `prompt()` siempre devuelve una cadena (tipo `String`)
- Para **convertir** una cadena en tipo `Number`:
  - `number parseInt(string [,base])`
    - Convierte la cadena a un entero en base `base`
    - Convierte hasta el primer carácter no válido en la cadena (omite espacios al principio)
    - Si la conversión no es viable, devuelve `NaN`
  - `number parseFloat(string)`
    - Convierte `string` en un número de coma flotante
    - Convierte hasta el primer carácter no válido en la cadena (omite espacios al principio)
    - Si no es un dígito, devuelve `NaN`
  - `number Number(string)`
    - Devuelve `NaN` si `string` contiene algún carácter no válido (omite espacios al principio y final).

Cajas de diálogo del objeto `window`

```
<html><head><title>Calculadora</title></head>
<body>
<script type="text/javascript">
// Se solicitan los operandos
do {
  opdo1 = prompt("Indique el primer operando: entero positivo");
} while (isNaN(parseInt(opdo1)) || parseInt(opdo1) < 0);

do {
  opdo2 = prompt("Indique el segundo operando: flotante positivo");
} while (isNaN(parseFloat(opdo2)) || parseFloat(opdo2) < 0);

// Se solicita el operador
opdr = prompt("Indique la operacion: * + - /", "+");

// Se complete la operación y, si se confirma, se muestra el resultado
opera = opdo1 + opdr + opdo2;
ok = confirm("Deseas calcular <" + opera + ">?");
if (ok) alert("El resultado es: " + eval(opera));
</script>
</body>
</html>
```

Validación de entradas de usuario

Es habitual utilizar JS para la validación de entradas del usuario desde un formulario antes de que este se procese.

- Se debe comprobar que los campos obligatorios no se hayan dejado vacíos
- Se debe verificar que los campos solo contengan caracteres permitidos o cumplan con una cierta gramática
- Se debe verificar que los valores estén dentro de los límites/rangos permitidos

```
<form method="post" action="XController.jsp" onSubmit="return validar(this)">
<label> User name: <input type="text" name="user"></label>
<label> Email address : <input type ="text" name="email"></label>
<input type ="submit" name ="submit" />
</form>
<script>
function validar (form) {
  fallo = validarUsuario(form.user.value);
  fallo += validarEmail(form.email.value);
  if (fallo == "") return true;
  else {
    alert(fallo); return false;
  }
}
</script>
```

```
function validarUsuario (inp) {
  if (inp == "")
    return "No se introdujo nombre de usuario\n";
  else if (inp.length < 5)
    return "Nombre de usuario demasiado corto\n";
  else if (/[^a-zA-Z0-9_-]/.test(inp))
    return "Caracteres inválidos en nombre de usuario\n"
  else return "";
}

function validarEmail(inp) {
  if (inp == "") return "No se introdujo email\n";
  else if (!((inp.indexOf("@") > 0)) ||
       /[^a-zA-Z0-9\.\@\_\-]/.test(inp))
    return "Caracteres inválidos en email\n";
  else return "";
}
```

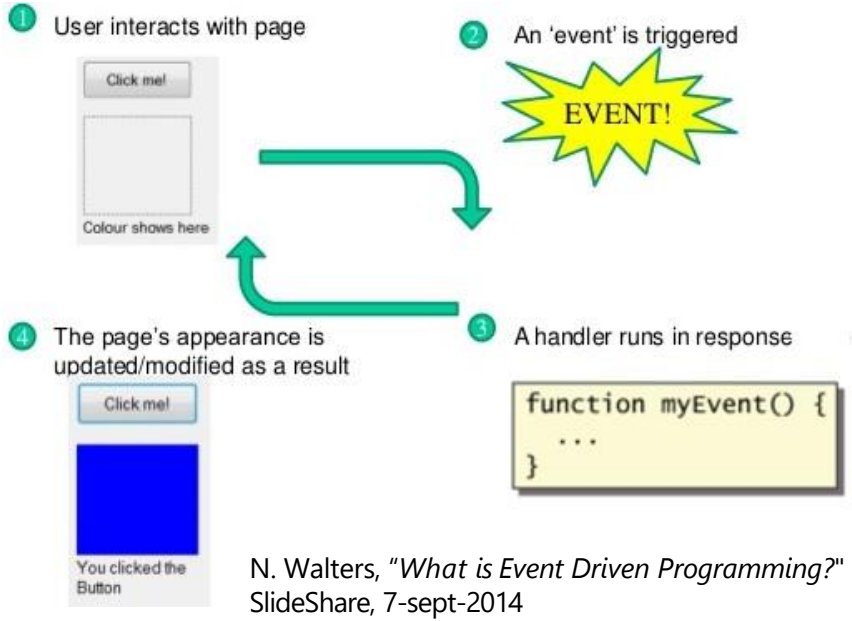
6.

Programación basada en eventos con JavaScript

Eventos JavaScript

Programación dirigida por eventos en JS

Las aplicaciones web son dirigidas por eventos (event-driven), esto es, reaccionan a eventos del usuario como clicks, pulsaciones, movimientos de ratón, etc.



N. Walters, "*What is Event Driven Programming?*" SlideShare, 7-sept-2014
https://tinyurl.com/ya58xbs9

- Podemos definir funciones manipuladoras de eventos (*event **handler** functions*) para una amplia variedad de eventos
- Estas funciones pueden manipular el objeto `document` (cambiando la página web)

https://www.w3schools.com/jsref/obj_event.asp

Tipos de eventos__ Interfaz `UIEvent`

- `UIEvent` (hereda de `Event`) recoge eventos sencillos de la interfaz de usuario
- Cada evento tiene su handler:

error	**onerror**
load	**onload**
unload	**onunload**
resize	**onresize**
Scroll	**onscroll**

- De `UIEvent` heredan las interfaces:
  - **MouseEvent**, **KeyboardEvent**, **InputEvent**, **FocusEvent**, **TouchEvent**, **WheelEvent**

Tipos de eventos__ Interfaz `UIEvent` Ejemplo `onload`

- Se produce un evento `(on)load` cuando se carga un objeto
- Normalmente los *handlers* para eventos onload están asociados con el objeto `window` o el elemento `body` de un documento HTML


```
<! DOCTYPE html>
<html lang="es-ES">
  <head>
    <title>Ejemplo onload</title>
    <script>
      function hola() {alert("Bienvenido!"); }
    </script>
  </head>
  <body onload="hola()">
    <p>Contenido aqu&iacute;</p>
  </body>
</html>
```

https://www.w3schools.com/jsref/obj_mouseevent.asp

Tipos de eventos__ Interfaz `MouseEvent`

- `MouseEvent` ocurre cuando el ratón del usuario interactúa con el documento
- Cuenta con unas propiedades propias del evento: `buttons`, `clientX`, `clientY`, `pageX`, `pageY`, etc.
- Además, maneja una serie de eventos:

onclick	**ondblclick**
oncontextmenu	**onmousedown**
onmouseenter	**onmouseleave**
onmouseover	**onmouseout**
onmouseup	

https://www.w3schools.com/jsref/obj_mouseevent.asp

Tipos de eventos__ Interfaz `KeyboardEvent`

- `KeyboardEvent` ocurre cuando el usuario pulsa una tecla
- Cuenta con unas propiedades propias del evento: `charCode`, `ctrlKey`, `key`, `location`, `shiftKey`, etc.
- Además, maneja una serie de eventos:

onkeydown	**onkeyup**
onkeypress	

https://www.w3schools.com/jsref/obj_mouseevent.asp

Tipos de eventos__ Interfaz `InputEvent`

- `InputEvent` ocurre cuando el usuario cambia el contenido de algún elemento de entrada de un formulario
- Cuenta con unas propiedades propias del evento: `inputType`, `data`, etc.
- Además, maneja el evento `oninput`

Tipos de eventos__ Interfaz `FocusEvent`

- Un evento de foco (`onFocus`) ocurre cuando un campo de formulario recibe foco de entrada presionando con el teclado o haciendo clic con el mouse
- Un evento de desenfoque (`onBlur`) cuando un campo de formulario pierde el foco por clic o presión de teclado por parte del usuario en otro elemento
- Un evento de cambio (`onChange`) ocurre cuando un campo de selección, texto o área de texto pierde el foco y su valor ha sido modificado

```
<form name="form1 "method="post" action="Xcontroller.jsp">
   <select name="select" required onChange="document.form1.submit();">
          <option value=""> Indique su juego favorito: </option>
          <option value="200812345"> The Last of Us </option>
          <option value="200867890"> Red Dead Redemption </option>
   </select>
</form>
```

```
<html>
 <head>
  <title>Ejemplo onchange</title>
  <script type="text/javascript">
function FahrenheitToCelsius(tFahr) {
 return (5/9)*(tFahr - 32);
}
  </script>
 </head>
<body>
 <form>
 Indique la temperatura en Fahrenheit:
  <input type="text" id="fahr" size="10" value="0“
    onchange="document.getElementById('celsius').value =
       FahrenheitToCelsius(parseFloat(document.getElementById('fahr').value)).toFixed(1);“ /><br/>
 La temperatura en Celsius:
  <input type="text" id="celsius“ size="10" value="" onfocus="blur();" />
</form>
</body>
</html>
```

blur() hace perder el foco al elemento; de este modo, #celsius nunca estará focalizado

https://www.w3schools.com/jsref/dom_obj_event.asp

Otros tipos de eventos de `Event`

- Interfaz DragEvent: hereda de MouseEvent
  - **`ondrag, ondragend, ondragenter, ondragleave, ondragover, ondragstart, ondrop`**
- Form Events: ocurre cuando un usuario interactúa con el elemento de formulario.
  - **`input, change, submit, reset, cut, copy, paste, select`**
- Eventos DOM (*mutation events*): diferentes eventos registrados por la interacción con un documento de HTML (*DOM Level 3 events*) -- se han sustituido por la interfaz `MutationObserver`(*DOM Level 4 – Sept. 2019*)

https://developer.mozilla.org/en-US/docs/Web/Guide/Events/Mutation_events

https://www.w3.org/TR/DOM-Level-3-Events/#events-mutationevents https://dom.spec.whatwg.org/

Manejadores de eventos y elementos HTML

- Los eventos HTML indican que algo le ha ocurrido a algún elemento(s) HTML
- Los manejadores (*event handlers*) son funciones de JS que procesan eventos
- Los *handlers* **deben estar asociados** con elementos HTML para eventos específicos
- Esto se puede hacer a través de los atributos de los elementos HTML

```
<input type="button" value="Ayuda" onclick ="ayuda()">
```

- Alternativamente, se puede usar una función de JS para agregar un controlador a un elemento HTML

```
// La mayoría de navegadores
window.addEventListener("load", ayuda)
// MS IExplorer
window.attachEvent("onload", ayuda)
```

Manejadores de eventos y elementos HTML

- Se debe considerar la posible variedad de navegadores:

```
if (window.addEventListener) {
   window.addEventListener("load", hola);
} else {
   window.attachEvent("onload", hola);
}
```

- Los *handlers* pueden eliminarse:

```
If (window.removeEventListener) {
   window.removeEventListener("load", hola);
} else {
   window.detachEvent("onload", hola);
}
```

Manejadores de eventos y elementos HTML


```
<head> <title>DOM Event Example</title>
  <style>
    #t { border: 1px solid red }
    #t1 { background-color: pink; }
  </style>
  <script>
  // Function to change the content of t2
  function modifyText(new_text) {
    var t2 = document.getElementById("t2");
    t2.firstChild.nodeValue = new_text; }

  // Function to add event listener to t
  function load() {
    var el = document.getElementById("t");
    el.addEventListener("click", function(){modifyText("four")}, false);
}
  </script>
</head>
<body onload="load();">
  <table id="t">
    <tr><td id="t1">one</td></tr>
    <tr><td id="t2">two</td></tr>
  </table>
</body>
```

Fuente: Mozilla.org

Vinculando un evento a un elemento

HTML event handlers: ¡¡Evitar!! (¡mala práctica!)

- HTML introdujo en sus primeras versiones un conjunto de atributos que podrían responder a eventos en los elementos a los que se agregaron
- Los nombres de los atributos coinciden con los nombres de los eventos
- P.ej. **`<a onclick="hide()">`**

DOM event handlers: (mejor opción que los manejadores de eventos HTML)

- Permiten separar HTML and JavaScript
- Compatible con los principales navegadores
- **Inconveniente**: solo se puede asociar una sola función a cualquier evento

  `element.onevent = functionName;`

  e.j., *`tileElement.onclick = moveTile;`*

Vinculando un evento a un elemento

DOM Level 2 Event Listeners: (El mejor enfoque hasta *DOM Level 4*).

- Los *event listeners* se agregaron a DOM Level 2 en el 2000 y **permiten** a un evento **invocar múltiples funciones**
- No es compatible con IE8 y versiones anteriores

```
element.addEventListener('event', functionName [, useCapture]);
```

```
document.getElementById("myDIV").addEventListener("mousemove", myFunction);
```

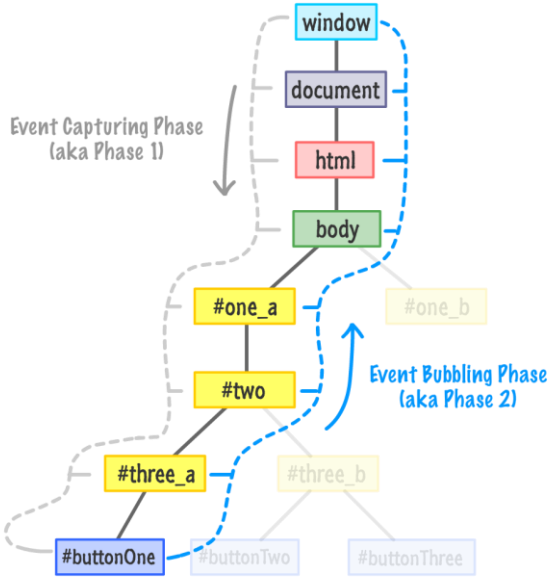
Para eliminar un *listener* (añadido con addEventListener) se debe invocar:

```
element.removeEventListener('event', functionName [, useCapture]);
```

Vinculando un evento a un elemento

DOM Level 2 Event Listeners: (El mejor enfoque hasta *DOM Level 4*).

```
<style>
  body * {
      margin: 10px;
      border: 1px solid blue;
  }
</style>
<form onclick="alert('form')">FORM
  <div onclick="alert('div')">DIV
    <p onclick="alert('p')">P</p>
  </div>
</form>
```



Fuente: kirupa.com

Programación Web

Presentación de la asignatura__ **Curso 2019/20**