HTML Dinámico

Desarrollo de Aplicaciones Web
Departamento Informática y Sistemas
Universidad de Murcia
Curso 2014/15

Motivación

- Añadir capacidades de interacción a las páginas web.
- Ejemplo**: validación de datos en formularios**.

- **Estrategias**:
  - Compilación: Applets Java, ActiveX*,* **Adobe Flash**
  - Interpretación → **JavaScript**

- En general, la **programación dinámica en HTML** permite:
  - Controlar el contenido y apariencia de un documento en respuesta a los eventos de usuario.

Contenido

- Parte I: Introducción a **JavaScript**
- Parte II: **Expresiones Regulares** en JavaScript
- Parte III: El modelo de objetos **DOM**

Parte I: Introducción a JavaScript

- Lenguaje **interpretado** incluido en los documentos HTML.
- Tiene un **propósito específico**: **actuar en páginas web**
- Inicialmente inspirado en el lenguaje Java, especialmente en las construcciones imperativas heredadas de C.
- **Lenguaje similar** (*Microsoft*)**:** JScript
- Estandarización: **ECMAScript**

Aplicaciones de JavaScript

- Hacer que una página web reaccione a la **interacción del usuario**. Ejemplo: desplegar un menú al pulsar un enlace.

- **Preprocesar datos** en el cliente antes de que sean enviados al servidor (**validación de formularios**).

- **Cambiar el contenido y el estilo** dinámicamente en respuesta a la interacción del usuario (**programación dinámica en HTML**)

Usos no permitidos

- **Principio de diseño del lenguaje**:
  - Proteger la privacidad de los usuarios y evitar accesos no autorizados a sus ordenadores.

- Por tanto, NO podemos:
  - Establecer o recuperar las preferencias del navegador y en general acceder a los menús y opciones del navegador.
  - Ejecutar una aplicación.
  - Leer o escribir ficheros o directorios.
  - Abrir conexiones de red.
  - Enviar correos sin intervención del usuario.

Características del lenguaje

- **Declaración de variables**:
  - **No se declaran tipos**
    - Ej.: `var i;`
  - **Alcance** variables: global y bloque (como en C)
- **Operadores:** igual que C/Java.
- **Control de flujo** (condicionales, bucles)
  - Como en C/Java
  - **switch** admite cadenas
- Comentarios como en C.

Tipos de Datos

- **Números** (`Number`): enteros y reales.
- **Booleanos** (`Boolean`): `true` y `false`
- `undefined`: indefinido → variable sin inicializar
- **Objetos**: tienen propiedades y métodos
- **Objetos del lenguaje**:
  - **Cadenas** (`String`)
  - **Arrays** (`Array`)
  - **Fechas** (`Date`)
  - **Funciones**
- **Los objetos se manejan por referencia (= Java).**

Tipos de Datos

- **Conversión implícita de tipos**:
  - `var i = "3" + 3; // i == "33"`
- Las cadenas tienen preferencia en el lenguaje.
- **Conversión cadenas a números**:
  - `var i = parseInt("10");`
  - `var f = parseFloat("10.2");`
- ¿No es un número?: `isNaN("hola"); // true`

- **Igualdad ==**
  - Aplicada a objetos compara referencias.
  - **Tipos primitivos**: realiza conversión tipos: `3 == "3"; // true`

Funciones

- **Declaración**:

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

- **No se declaran los tipos de los parámetros ni el valor de retorno.**

- **Paso de parámetros por valor**.

- El nombre de una función (ejemplo “suma”) es un objeto que puede ser asignado a una variable.

Arrays

- Los arrays pueden contener datos de distinto tipo.
- **Creación**:
  - `var array = new Array();`
  - `var array2 = ["juan", 5, false];`
- **Acceso a un array**:
  - Índice a partir de 0.
  - **Longitud**: propiedad **`length`**
  - **Indexación con cadenas**: `array["cantidad"] = 3;`
- Pueden contener **posiciones vacías**:
  - `array[0] = 20;`
  - `array[20] = 5;`
    - → `array[5]` es **`undefined`**, `length == 21`

Cadenas

- **Concatenación**: operador **+**
- **Tamaño:** propiedad **`length`**
- Acceso a un carácter: método **`charAt`**`(i)`
- Buscar **subcadenas**: método **`indexOf`**(“subcadena”)
- Extraer subcadenas: método **`substring`**`(i, j)`

Conversión array - cadena

- De un array se puede obtener una cadena utilizando la función **join**:
  - `var array = ["uno", "dos", "tres"];`
  - `var resultado = array.join("-");`
    `// "uno-dos-tres"`

- A partir de una cadena se puede conseguir un array con la función **split**:
  - `var cadena = "cuatro-cinco-seis";`
  - `var array = cadena.split("-");`

Fechas

- Clase **`Date`**: representa una fecha y hora

- **Construcción:**

  - **Fecha actual**: `var hoy = new Date();`

  - `var fecha = new Date(2004, 9, 20);`
    `// año, mes y día → 20/10/2004`

  - `var fecha = new Date(2004,9, 20, 22, 50, 50);`
    `// 20/10/2004, 22:50:50`

Fechas

- **Acceso y modificación**:
  - **Propiedades***: fullYear, month, date* (día del mes)*, hours, minutes, seconds*
  - Tiene métodos **get/set** para acceder a sus propiedades.
    - Ejemplo: `fecha.getMonth();`
  - Los meses comienzan a numerarse por 0: enero(0), febrero(1), etc.

- **Aritmética de fechas**:
  - Ejemplo: podemos incrementar el día del mes en 45 y actualiza el mes y el día para dentro de 45 días:
  - `fecha.setDate(fecha.getDate() + 45);`

Situación del código JavaScript

- **Declaración en una página HTML**:

```
<script type="text/javascript" language="javascript">
…
</script>
```

  - Habitualmente el código se sitúa en la **cabecera.**

- **En un fichero externo** (útil para librerías de funciones):

```
<script type="text/javascript"
      src="libreria.js"></script>
```

- **En un manejador de evento.** Se estudiará más adelante.

Parte II: Expresiones Regulares

- **Motivación**:
  - Búsqueda y reemplazo de un patrón en una cadena.
  - **Muy útiles para validar campos de un formulario**
- Patrones expresados de forma flexible y concisa.
- Ejemplos:
  - `var er = /hola/; // patrón "hola"`
  - `var er = /\d{3}/; // Número de 3 cifras`

- **Tipos de patrones**:
  - **Básico**: `/Hola/` → patrón "Hola"
  - **Insensible mayúsculas/minúsculas**: `/hola/i`

Caracteres especiales

- **\b** límite palabra
- **\d** dígito
- **\s** espacio en blanco
- **\w** letra, dígito o "_"
- **.** cualquier carácter excepto salto de línea
- **[ ]** conjunto de caracteres
- **[^]** cualquier carácter excepto …
- **[a-m]** rango de caracteres
- ***** 0 ó más apariciones
- **?** opcional
- **+** 1 ó más apariciones
- **{n}** n apariciones
- **{n, m}** entre n y m
- **^** comienzo línea
- **$** fin línea
- **( )** agrupación
- **|** alternativa
- **** carácter escape

Ejemplos

- **Cualquier dígito**: **\d**
  - `var er = /\d\d/;` → patrón de búsqueda formado por dos dígitos: 01, 12, etc.
- **Espacio en blanco**: **\s**
  - `var er = /hola\smundo/;` → secuencia “hola” seguida de un espacio y la secuencia “mundo”.
- **Carácter alfanumérico**: **\w** → letra, dígito o “_”
  - `var er = /\w\w\w/;` → tres caracteres alfanuméricos: “h32” “aaa”, “321”, “a_b”, etc.
  - No se incluyen la ñ y los caracteres acentuados en español.

Ejemplos

- **Cualquier carácter**: **.**
  - `var er = /abc.123/;` → secuencia “abc” seguida de cualquier carácter y finalizada con la secuencia “123”.
- **Conjunto de caracteres**: **[ ]**
  - `var er = /[abc]/;` → un carácter que puede ser ‘a’, ‘b’ o ‘c’.
- **Negación de un conjunto**: **[^]**
  - `var er = /[^abc]/;` → un carácter que no sea ni ‘a’, ‘b’ o ‘c’.
- **Conjunto expresado como rango ASCII**:
  - `var er = /[a-z]/;` → un carácter que va desde la ‘a’ hasta la ‘z’ en el código ASCII.
  - Nota: la ñ no se encuentra en el rango.

Ejemplos

- **Cualquier número de veces**: *
  - `var er = /hola\d*/;` → secuencia 'hola' seguida de un dígito que puede repetirse cero o varias veces.
  - `var er = /a[bc]*/;` → patrón de búsqueda que empieza por 'a' y que puede ir seguido de 'b' o 'c' repetidos cero o varias veces: 'abbbcb', 'accbcc', etc.
- **Repetición al menos una vez**: +
  - `var er = /\d+/;` → un dígito que puede repetirse varias veces.
- **Opcional:** ?
  - `var er = /a?\d+/;` → una secuencia de dígitos que puede comenzar opcionalmente por 'a'.
- **Número fijo de repeticiones**: {n}
  - `var er = /\d{3}/;` → números de tres cifras.

Ejemplos

- **Rango de repeticiones**: **{n, m}**
  - `var er = /\d{3, 5}/;` → números entre 3 y 5 cifras.
  - `var er = /[ab]{3, 5}/;` → caracteres ‘a’ o ‘b’ repetidos entre 3 y 5 veces.
- **Grupo**: **( )**
  - `var er = /(hola){3}/;` → secuencia ‘hola’ que se repite 3 veces.
  - `var er = /(ab\d{2}){3}/;` → secuencia ‘ab’ seguida de dos dígitos, todo ello repetido tres veces.
- **Alternativa**: **(|)**
  - `var er = /(hola|hello){3}/;` → secuencia ‘hola’ o ‘hello’ repetida 3 veces. Ejemplo “holahellohello”.

Expresiones regulares

- Comprobar si una cadena cumple una expresión regular:
  - `er.test(cadena);`
  - Si retorna `true`, la cadena cumple el patrón.
- **Consejos de uso**:
  - Las expresiones regulares **buscan patrones**.

```
var er=/hola/;
er.test("abcaholaabc"); // true
```

  - Habitualmente nos interesa que una **cadena completa** cumpla un patrón. Utilizaremos los **caracteres especiales ^** y **$** para delimitar el **principio y fin** del patrón.

```
var er=/^hola$/;
er.test("abcaholaabc"); // false
```

Expresiones regulares

- **Consejos de uso** (continuación):
  - Hay que llevar cuidado con los **caracteres especiales** al definir las expresiones. Utilizamos el carácter de escape \

    `/^\w+\.\w*$/` → uno o más caracteres alfanuméricos que incluyen un “.”

  - **No se pueden definir rangos numéricos**.

    `/^[12-40]$/` → “1”, “2”, “3”, “4” ó “0”. En el patrón, 2-4 define un rango de caracteres.

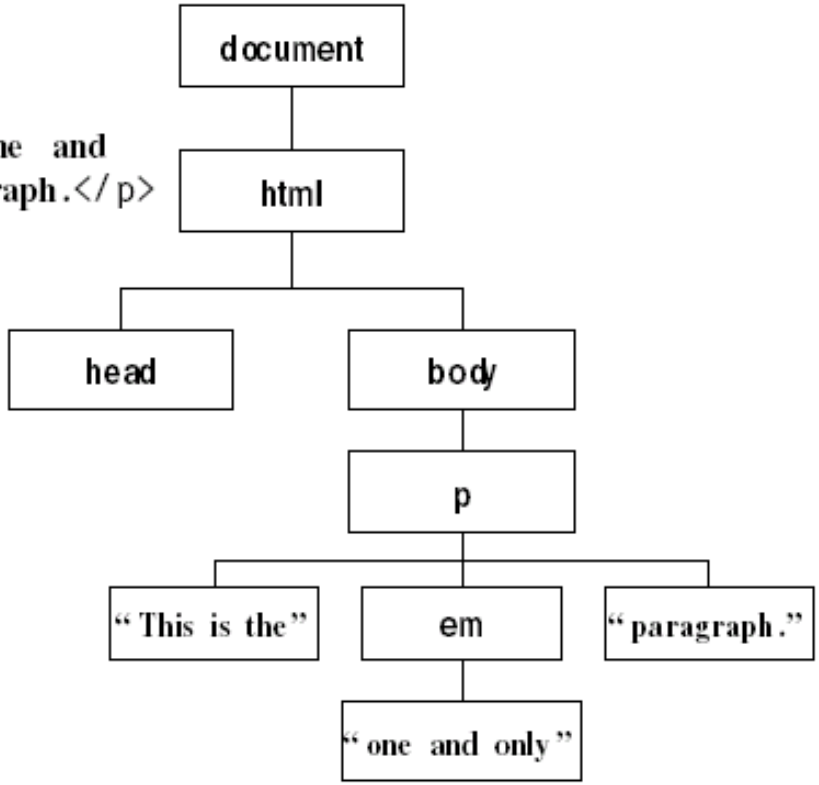
Parte III: Modelo de objetos DOM

- El código *JavaScript* puede acceder al contenido de una página utilizando un API denominada **DOM**.
- El API DOM ha sido **estandarizada** por el consorcio W3C.
- DOM permite acceder al contenido de una página como si fuera un **árbol de objetos**.

Árbol del documento

- El navegador crea un **árbol** que representa los elementos de la página HTML

```
<html>
  <head></head>
  <body>
      <p>This is the <em>one and
          only</em> paragraph.</p>
  </body>
</html>
```



Objeto `window`

- **window** ofrece el **contexto general del navegador**
- Abrir **diálogos**:
  - `window.alert("…")` → aviso
  - `window.confirm("…")` → petición de confirmación. Devuelve *true* o *false*

- **Propiedad location**:
  - Consulta y modificación de la página que visualiza la ventana.
  - `window.location.href = "http://www.um.es"`

- **Propiedad status**: acceso a la barra de estado.

Objeto `window`

- **Propiedad navigator**:
  - Propiedad **`userAgent`**: cadena que identifica al navegador.

- **Propiedad history**:
  - Historial de navegación de la ventana.
  - Métodos **`back()`** y **`forward()`** equivalen a los botones "Atrás" y "Adelante".

- **Propiedad screen** (pantalla):
  - Acceso a las dimensiones de la pantalla.
  - Propiedades: **`width`**, **`height`**, **`availWidth`**, **`availHeight`**

Acceso al documento

- **document** representa la **página HTML**
- Permite el acceso a **toda** la información del documento.
- **Obtener referencias a declaraciones por el atributo id.**

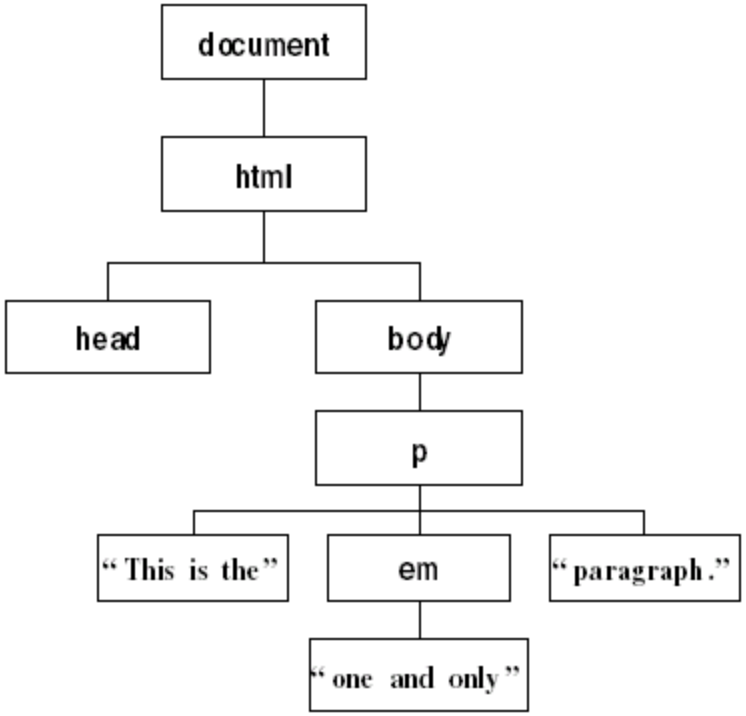
```
var logo = document.getElementById("logo");
```

- **Obtener todos los objetos de un tipo de etiqueta**:

```
var array = document.getElementsByTagName("div");
```

Recorrido del documento

- **Tipos de nodos**:
  - Nodos de **elementos (etiquetas)**
  - Nodos de **texto**.
  - Raíz: **document**.



- **Propiedades** de los nodos:
  - `firstChild, lastChild, childNodes[],`
  - `parentNode, nextSibling, previousSibling`

Modificación del documento

- **Modificación de los hijos** desde el nodo padre:
  - `insertBefore(nuevoHijo, nodoHijoReferencia)`
  - `replaceChild(nuevoHijo, antiguoHijo)`
  - `appendChild(nuevoHijo)` → añade al final
  - `removeChild(hijo)`

- Los **atributos de HTML** son accesibles mediante propiedades:
  - Ejemplo: `imagen.src = "imagen.gif";`

Construcción de nodos

- Propiedad: **innerHTML**
  - Asignamos el texto HTML que representa los hijos de un nodo.

```
var bloque = document.getElementById("contenido");
bloque.innerHTML = "<strong>Texto</strong> y más texto";
```

- Construir nuevos nodos con la función **createElement**:

```
var nuevo = document.createElement("span");
```

  - El nuevo nodo se puede añadir como hijo de otro nodo.
- Podemos asignar contenido al nuevo nodo con **innerHTML**:
  - `nuevo.innerHTML = "contenido span";`

Formularios

- Acceso a los **controles de un formulario:**
  - Array **elements**
  - `formulario.elements["nif"]`
  - Alternativa: `formulario.nif`

- La acción de los **botones** de tipo *submit* y *reset* puede ser realizada aplicando métodos del formulario:
  - `formulario.submit()`
  - `formulario.reset()`

Formularios - Controles

- **Controles de tipo input:**
  - Están disponibles los **métodos**:
    - `focus()`: ganar el foco de control
    - `blur()`: sacar el foco de control
    - `select()`: si contiene texto, lo marca seleccionado

- **Controles tipo select**: (menús y listas)
  - *Opciones* accesibles a través del **array** `options`
  - Propiedad `selectedIndex`:
    - Almacena el índice de la opción seleccionada.
    - Si su valor es -1, no hay ninguno seleccionado.

Formularios - Controles

- **Botones de radio**:
  - El control es un **array** de botones.

- Todos los **atributos de HTML** son accesibles como propiedades:

```
formulario.elements["nif"].value;
```

- Los **atributos vacíos** (ejemplo *checked*) son de tipo booleano:

```
formulario.elements["licencia"].checked = false;
```

Acceso a propiedades de estilo

- Estilos de una etiqueta accesibles mediante **propiedad style**:

```
document.getElementById("intro").style.color = "red";
```

- **Precaución con nombres de las propiedades de estilo**:
  - **Guiones** → cambiar a mayúscula.
    - Ej. `font-size` a `fontSize`
  - Los valores de TODAS las propiedades son cadenas, incluso las medidas: ejemplo “1.5”

- Consultar y cambiar la **clase** de una etiqueta:
  - Propiedad **className.**

Eventos

- **Eventos**:
  - Acciones del usuario sobre algún elemento de la página: clic, doble clic, modificar un campo, etc.
  - Cambio de estado interno de la página (la página está disponible, se abandona la página)
- **Los eventos son tratados con funciones (manejadores).**
- **Registrar un manejador de evento**:
  - Habitualmente, dentro de la etiqueta:
    - `<button ... onclick="manejador()">`
  - En ocasiones, asignando la referencia a la función:
    - `boton.onclick = manejador;`

Eventos

- **Acción por defecto**:
  - Algunos eventos tienen asociada una acción por defecto. Ejemplo: pulsar el botón *submit* de un formulario.
  - Es posible controlar si se permite la acción por defecto.
    - `<input ... onchange="return manejador()">`
  - Para anular la acción por defecto el manejador devuelve `false`.
- Establecer en el manejador el nodo que recibe el evento utilizando **this**:
  - `<input ... onchange="manejador(this)">`
  - **`this`** referencia al nodo. En el manejador, el parámetro se puede utilizar para acceder a las propiedades del nodo.

Tipos de Eventos

- **Foco de control y cambios** (controles de formularios):
  - onblur: pérdida foco de control
  - onfocus: gana el foco
  - onchange: contenido ha cambiado y pérdida del foco
  - onselect: selección texto.
- **Ratón**: sobre cualquier elemento
  - onclick: clic sobre un elemento
  - ondblclick: doble clic
  - onmousedown: pulsar el ratón
  - onmouseup: liberar pulsación ratón
  - onmouseover: entrar puntero zona elemento
  - onmouseout: salir con el puntero de la zona del elemento

Tipos de Eventos

- **Teclado**:
  - onkeypress: pulsación tecla (completa)
  - onkeydown: tecla pulsada (no liberada)
  - onkeyup: liberación tecla

- **Formularios:**
  - onsubmit: pulsación botón envío formulario

- **Carga de la página (`<body>`):**
  - onload: carga de la página → documento accesible
  - onunload: la página va a ser descargada, sale de la página

Ejemplo: Mostrar y ocultar bloques

- **Uno de los efectos más útiles** en la programación dinámica HTML es mostrar y ocultar un bloque.
- Normalmente, se define una función del siguiente modo:
  - Parámetro: **"id"** del elemento sobre el que actúa.
  - Utiliza la propiedad de **estilo "display"**.
  - El bloque debe tener establecido explícitamente la propiedad de estilo display para que funcione la función.

```
function muestraBloque(id) {
        var bloque = document.getElementById(id);
        if (bloque.style.display == "none")
                bloque.style.display = "block";
        else bloque.style.display = "none";
}
```

Validación de Formularios

- **Motivación**:
  - Evitar el envío de información al servidor con datos erróneos.

- Tipos de **validaciones**:
  - En el momento de la introducción de cada dato.
  - Antes del envío del formulario.

- Utilizaremos **funciones JavaScript** que validen los campos.

Validación de Formularios

- **En la entrada:**
  - Recomendable cuando el formulario es extenso
  - **Evento**: **onchange** del control
    - Cuando ocurre un cambio en un control se llama a la función de validación de ese control.

- **Antes del envío** (**siempre**):
  - Evitar que se envíe un formulario con errores
  - Evento **onsubmit del formulario**:

    ```
    onsubmit="return validarFormulario(this)"
    ```
  - Si hay **errores**, no deben enviarse los datos:
  - → el manejador debe **detener el evento**.