

Introducción a JavaScript y a Ajax con Jquery (I)



Introducción

JAVASCRIPT

Introducción

- **JavaScript** es un sencillo lenguaje de (**script**) programación **interpretado** cuyo código se inserta dentro de un documento HTML para crear páginas Web dinámicas
 - Se interpreta y ejecuta en el navegador del usuario al cargar la página o cuando sucede un evento. Es el más utilizado en este ámbito
 - No lo confundamos con el lenguaje de programación Java. JavaScript es una marca registrada por Sun de manera independiente a Java.
 - Es un lenguaje de programación del lado del cliente (navegadores)
 - Son independientes de la plataforma en la que se muestre el documento html (independiente de mac, linux, windows).

Historia

- A principios de los noventa, se comenzó a necesitar más funcionalidad por parte de las páginas web, y eso se fue solventando con los applets de Java.
- Para evitar la dificultad de emplear los applets, se creó un nuevo lenguaje del lado de los clientes, y que se llamó *LiveScript*.
- Netscape y Sun firmaron un acuerdo y sacaron la primera versión oficial llamada *JavaScript*, nombre que se le dió fundamentalmente por marketing ya que la palabra Java estaba de moda

Herramientas

Los navegadores pueden no implementar correctamente las recomendaciones del estándar de JavaScript (<http://caniuse.com/> para consultar si algo funcionará o no en los navegadores).

<http://www.w3.org/standards/webdesign/script> (Especificaciones Web APIs Javascript)

Por ello debemos realizar pruebas con los siguientes navegadores:

- Internet Explorer (más de una versión).
- Mozilla Firefox.
- Chrome.
- Safari y Opera, a ser posible.

Cada uno de estos navegadores tiene herramientas o plugins para ayudarnos durante el proceso desarrollo.

Cabe destacar la herramienta para desarrolladores que tiene Chorme.

Herramientas

De cara a la edición tenemos muchas posibilidades, desde aplicaciones complejas tanto de pago como **Dreamweaver** como gratuitas en el caso de **Eclipse**. Otra aplicación más simples, aunque no por ello poco adecuada, es **Notepad++** .

En el extremo de la simplicidad estaría el bloc de notas de Windows (al fin y al cabo un archivo de javascript es un archivo de texto plano), poco recomendable trabajar con esta herramienta.

Sintaxis Básica

JAVASCRIPT

Elementos de JavaScript

- Palabras Reservadas: La lista de palabras reservadas en Javascript es: `break`, `case`, `catch`, `continue`, `default`, `delete`, `do`, `else`, `finally`, `for`, `function`, `if`, `in`, `instanceof`, `new`, `return`, `switch`, `this`, `throw`, `try`, `typeof`, `var`, `void`, `while`, `with`.

Se entiende por palabra reservada aquella que usan para construir las sentencias en Javascript.

- Sintaxis sentencias y programas
- Reglas para expresiones, variables y literales
- Modelo de objetos (aunque JavaScript cliente y servidor tienen objetos predefinidos diferentes)
- Objetos predefinidos y funciones (como Math)
- Motores javascript: Interpretes de Javascript, a destacar V8 de google Chrome (C++). Y Gecko(C++), Spidermonkey(C++) y Rhino (Java) de Mozilla. Chakra de IE9.

Usando JS

¿Cómo incluir código JavaScript?

- Escribiendo el código directamente en el fichero .html

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

```
    <!-- /* Sentencias o funciones Javascript NO DEBES INCLUIR HTML EN EL  
CÓDIGO JAVASCRIPT!!!!*/ //-->  
    </script>
```

- Incluyendo un fichero de Script externo ← la mejor opción

```
<script language = "JavaScript" type="text/javascript"src="miJSExterno.js">
```

Ojo!! a tomar en cuenta:

- Englobar nuestro código entre <!-- //-->
- Incluir, en **body**, la posibilidad de que el cliente no tenga activado JavaScript

```
<noscript>
```

```
Tu navegador no acepta JS<a href="pag_sin_script.html">. Pincha en esta página</a>
```

```
</noscript>
```

Usando JS

Formas de usarlo:

- Inicializando algún atributo HTML.
- Llamando a una función ante algún evento

```
<form name="miForm">
```

```
    <input type="button"
```

```
    name="{miVar};"
```

```
    onClick="miFuncionJS;">
```

```
</form>
```

A tomar en cuenta al programar

- No se tienen en cuenta los espacios en blanco ni las nuevas líneas
- Es sensible a mayúsculas y minúsculas. Se recomienda escribir en minúsculas para que pasen los validadores de accesibilidad.
- Es un lenguaje sin tipo.
- No es obligatorio, pero sí se recomienda el hecho de acabar las sentencias con un punto y coma. JS acepta un salto de línea como separación de las sentencias.
- Se pueden incluir comentarios de una línea como `//` o de varias líneas `/**/`, tal como se hace en Java.
- Limitaciones (que podrían saltarse en determinados casos)
 - No puedes acceder a ficheros del ordenador.
 - No puedes acceder a recursos fuera del dominio donde está el JS ejecutándose.
 - No cierran ventanas que ellos no hayan abierto

Introducción

El primer script con JavaScript

- *Ver ejemplo "01_ejercicio.html"*

Funcionamiento:

1. Vamos a ver la inclusión de un JS desde un fichero externo.
2. Se mostrarán dos alerts o mensajes estilo popup.
3. Se añaden comentarios que expliquen el funcionamiento del código.
4. Se añade en la página HTML un mensaje de aviso para los navegadores que no tengan activado el soporte de JavaScript

Variables

Las variables en JavaScript se crean mediante la palabra reservada **var**, aunque no es obligatorio, porque JS la crearía como una variable global si no está declarada. Por ejemplo:

```
var variable1= 10;  
var variable2= 20;  
var total= variable1+ variable2;
```

Si cuando se declara una variable a la vez le da valor, se dice que la variable ha sido **inicializada**. En JavaScript no es obligatorio inicializar las variables.

```
var variable1;  
var variable2= 1;  
variable1= 3;  
var total= variable1+ variable2; //si hago total = variable1+variable2, total sería una variable global
```

Identificadores

Es el nombre de una variable, y al igual que en cualquier lenguaje debe cumplir unos criterios:

- Sólo puede estar formado por letras, números y los símbolos \$ (dólar) y _ (guión bajo)
- El primer carácter no puede ser un número.

Ejemplos correctos:

```
var $var1;
```

```
var _$var2;
```

```
var $$$$__$$$var3;
```

Tipos de variables

Las variables en JS no tienen tipo, por tanto, tomarán un valor, según sea la forma en la que las inicialicemos. Así será lo más parecido a asignarle un tipo.

- **Numéricas**

- Puedes guardar números enteros o decimales (int o float)
- Usando el operador . (punto) es como identificas un float.

`var edad = 30; //entero`

`var flotante= 142.82; //equivalente a float`

Cadenas de Texto

- Parecido a los Strings de Java.
- Se inicializan con el valor entre comillas dobles o simples.

```
var nombre= "Salimos a Comer";  
var nombreWeb = 'salimosacomer.com';
```
- Puedes mezclar comillas dobles y simples juntas.

```
var nombre1= "salimosacomer.com 'encuentra tu restaurante' ";  
var nombreWeb2 = 'salimosacomer.com "encuentra tu restaurante" ';
```
- Se pueden escapar las comillas y otros caracteres como en Java

```
var nombreMix1= ' Uso \'comillas simples\' dentro y fuera';  
var nombreMix2 = "Uso \"comillas dobles\" dentro y fuera";
```
- Secuencias de Escape:
 - Retorno de carro `\n`
 - Tabulador `\t`
 - Comilla simple `\'`
 - Comilla doble `\"`
 - Barra inclinada `\\`

Cadenas de Texto

Ejercicio 2

- Declarar una variable con un mensaje.
- Mostrar el mensaje en un alert.
- Declarar variables mezclando comillas simples y dobles.
- Mostrar el mensaje en un alert.
- Prueba a hacerlo en un código script dentro de la propia página.

Arrays

- Colección de variables, con la peculiaridad de que pueden ser números, cadenas,... porque recordemos que **no hay tipos!!!**
- Por ejemplo:
 - `var diasLab = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"];`
- Se accede al valor como en Java, es decir, usando los corchetes, y también comienzan en 0.
 - `var primerDia= diasLab[0]; // primerDia= "Lunes"`
 - `var otroDia = diasLab [3]; // otroDia = "Jueves"`

Ejercicio 3:

- Crear un array con 4 lenguajes de programación
- Mostrarlos con la función alert (1 alert por cada lenguaje).

Booleanos

- Como en Java, los valores para lo que serían booleanos son “true” y “false”
 - `var encontrado= false;`
 - `var terminado= true;`

Operadores

- **Asignación (=)**

- A la izquierda una variable y a la derecha, variable, valor, expresión, etc.
 - `var variable1 = 30; // Es un número`
 - `var variable2 = variable1 + 20; // Otro número`

- **Incremento/decremento (++/--)**

- Sólo lo podrás usar cuando hay números!.
- Como en Java, es un incremento o decremento unario.
- Se pueden poner como prefijo o como sufijo.
 - `var nume1 = 30;`
 - `var nume2 = 40;`
 - `//con sufijo`
 - `var resul1 = nume1++ + nume2; // incrementa nume1 después de sumar`
 - `alert(nume1); //31`
 - `alert(nume2); //40`
 - `alert(resul1); //70`
 - `// y con prefijo`
 - `var resul2 = ++nume1 + nume2; // incrementa nume1 antes de sumar`
 - `alert(nume1); //32`
 - `alert(nume2); //40`
 - `alert(resul2); //72`

Operadores Lógicos

- **Negación (!)**

- Si la variable es boolean, se niega (Cambia) su valor
- Si es un número, un 0 es false, y el resto true. Al negar, se cambia.
- Si es una cadena, false si es vacía (" ") y true en otro caso. Al negar se cambia su valor
 - `var num= 0;`
 - `var tiene= ! num; // tiene= true`
 - `num= 2;`
 - `tiene= ! num; // tiene= false`

Operadores Lógicos

- **AND (&&)**

- Como siempre, será true si los dos operandos que une son true
 - `var booleana1= true;`
 - `var booleana2= false;`
 - `var result = booleana1 && booleana2; // resultado = false`
 - `resultado = booleana1 && ! booleana2; // resultado = true`

- **OR (||)**

- Como en Java, true si alguno de los dos operandos es true
 - `var booleana1 = true;`
 - `var booleana2 = false;`
 - `var result = booleana1 || booleana2; // result = true`
 - `result = !booleana1 || booleana2; // result = false`

Operadores

- **Matemáticos**

- Al igual que en Java, tenemos: suma (+), resta (-), multiplicación (*) y división (/).
- Se puede hacer, como en Java, una simplificación en la asignación y operación matemática: +=, -=
- Tenemos la operación módulo (%), que devuelve el resto que resulta al dividir los dos números.

Operadores

- **Relacionales**

- Al igual que en Java, los operadores son:
 - mayor que (>)
 - menor que (<)
 - mayor o igual (>=)
 - menor o igual (<=)
 - igual (==)
 - distinto (!=)
- Se pueden usar comparando cadenas, no sólo números. A la hora de comparar cadenas para ver si son iguales, se hace con el == (una diferencia con Java).
 - Al comparar letras, recordemos, que como en Java las minúsculas irían primero.

Control de flujo

JAVASCRIPT

Control de flujo

- **if (Condicional simple)**

```
if(condicion) {  
    // Sentencias  
}
```

- **if-else (Condicional compuesto)**

```
if(condicion) {  
    // Sentencias  
}  
else {  
    // Sentencias  
}
```

Switch

- Como en Java, se evalúa el valor de una variable, y se miran los posibles casos (como if-else), si no coincide con ninguna pasará por default.

```
switch(variable) {  
    case v1: { // si el valor de la variable es igual a v1  
        //sentencias  
        break;  
    }  
    case vn: {  
        //sentencias  
        break;  
    }  
    default { //pasa por aquí si no coincide ningún valor  
        //sentencias  
        break;  
    }  
}
```

Bucles

Para realizar repeticiones la forma de hacer es mediante bucles, en javascript dispones de varias formas de hacerlas:

- for
- for... in
- do{...}while(condicion)
- while(condicion){}

for

Ejemplo:

```
for (var i=0;i<10;i++){  
    //las sentencias dentro de este bloque se repetiran 10 veces  
}
```

Bucles

for ... in

Esta forma «deriva» del bucle for. Este tipo de bucles se utiliza de forma conjunta con objetos. Veamos un ejemplo con un array.

```
var miArray= new Array(1,2,3,4,5,6,7);  
for (var numero in miArray){  
    document.write(numero);  
}
```

Bucles

do{ ... }while(condicion)

Este bucle se ejecuta una vez al menos, y se repite mientras se cumpla la condición indicada entre paréntesis.

En el ejemplo siguiente se repite 5 veces las sentencias dentro del bucle.

```
var contador =0;  
do{  
    console.log(contador);  
    contador++;  
}while (contador<5)
```

Bucles

while(condicion){}

Este bucle se repite mientras se cumpla la condición indicada entre paréntesis.

En el ejemplo siguiente se repite 5 veces las sentencias dentro del bucle.

```
var contador =0;  
while (contador<5){  
    console.log(contador);  
    contador++;  
}
```

Operadores y Control de Flujo

• Ejercicio 5

- Vamos a simular una asignación a mesas electorales. Aplicaremos una fórmula de mentira, y finalmente le asignaremos letras a la persona que se le vaya introduciendo.
- Pasos:
 - Pedir con la función `prompt()` (vamos a investigar) el dni de la persona.
 - Verificar que el DNI sea mayor que 0 y menor que 99 millones, sino mostrar un alert de error. Aplicar `if`
 - Con el DNI correcto, hay que asignarle mesa electoral. Las mesas deberían guardarse en un array.
 - Para asignarle la mesa, devolveremos un valor del array, y para conocer la posición usaremos la función `random`.
`Math.floor((Math.random()*7)+0);`

• Ejercicio 6

- Mostrar la tabla de multiplicar de un número, usando un bucle `for`.
- Hay que pedir el número por pantalla e ir mostrando con un alert cada multiplicación.

Ámbito de las variables

Como en cualquier lenguaje de programación, al ver las funciones es habitual definir cuál es el ámbito de uso de una variable

- Hay variables locales y variables globales.
- Las variables dentro de una función tienen un ámbito de ejecución supeditado a la ejecución de esa función, es decir, son locales a la función.
- Las que están fuera de una función son globales.
- Las que no tienen var delante, son globales, sea o no dentro de una función.
- Si se tiene una variable global y dentro de una función otra local con el mismo nombre, como en Java, prevalece la de la función.
- Si se tiene una variable global y dentro de una función otra global igual nombre, es como si fuese la misma, y por tanto, la que está dentro de la función, si es modificada, machaca el valor de la que está fuera.

Funciones

JAVASCRIPT

Funciones

Como en programación estructurada, una función al final es una agrupación de código con el fin de reutilizar y organizar.

- En JS se usa la palabra reservada **function** para indicar que es una función.

```
function miFuncion() {  
    ...  
}
```

- Las funciones pueden tener parámetros, si no los tienen siempre su llamada se hará con los paréntesis vacíos (como en Java).

Funciones – Parámetros y retorno

- Como en Java, se indican entre paréntesis y separados por coma, pero **no se indican tipos!!!**. Puede haber n parámetros
 - `function miFuncion(parametro1, parametro2) { //Sentencias }`
- Para **llamar** a la función desde otra parte del código:
 - `miFuncion(1, 2)`
- Para **devolver** un valor, se usa la palabra reservada `return`
 - Cuando haces un `return` la ejecución de la función no continúa. Sólo se puede devolver un único valor, pero nuevamente **no se indica el tipo**.

```
function miFuncion(parametro1, parametro2) {  
  var a=30;  
  return a;  
}
```

Ojo con lo siguiente: JS no te generará ningún error si pasas un número diferente de parámetros de los que espera.

Funciones – Parámetros y retorno

Uso de arguments.

En Javascript es posible pasar un número variable de argumentos, los argumentos de una función son guardados en el objeto arguments (un array).

```
function color(){  
  for (var i =0; i<arguments.length;i++){  
    console.log(arguments[i])  
  }  
}  
color("amarillo", "verde", "rojo" , "azul");
```

Funciones – Parámetros y retorno

Objeto Function


Toda función en Javascript es un objeto de tipo Function.

Por ello podemos declarar una función del modo siguiente:

```
ejemploRaro = new Function("a","b","return a+b");
```

```
ejemploRaro("uno ", "dos"); // resultado "uno dos"
```

Esto nos puede ser útil a la hora de comprender lo que viene a continuación.



Funciones anidadas y Closures

Un closure es un tipo especial de objeto que combina dos cosas: una función, y el entorno en que se creó esa función. El entorno está formado por las variables locales que estaban dentro del alcance en el momento que se creó el closure.

Podemos anidar funciones dentro de funciones, veamos el ejemplo siguiente:

```
function inicio(){  
  var nombre = "pepe";  
  function muestraNombre(){  
    console.log(nombre);  
  }  
  muestraNombre();  
}
```

Funciones anidadas y Closures

Se mostrará en la consola el nombre «pepe» al llamar a la función `inicio()`; . Es un buen ejemplo del ámbito de las variables.

Pero podríamos algo como esto:

```
function inicio(){  
  var nombre = "pepe";  
  function muestraNombre(){  
    alert(nombre);  
  }  
  return muestraNombre;  
}
```

En este caso no ocurre nada, la razón es que la función **`inicio()`** ha devuelto la función `muestra nombre` antes de que esta se ejecute.

Funciones anidadas y Closures

Probemos a escribir lo siguiente tras declarar nuestra nueva función **inicio()** del ejemplo anterior.

```
var mifuncion = inicio();  
mifuncion();
```

Vemos que ahora si que ocurre algo. Si nos paramos a analizar al situación hay algo que no debería de encajar y es que la variable **nombre** declarada en ámbito de la función **inicio()** debería de haber desaparecido una vez se terminó de ejecutar **inicio()** y no ha sido así . Esto es porque **miFuncion** se ha convertido en un **closure**.

Funciones predefinidas

eval : Evalúa una expresión que se le pasa a esta función como una cadena.

```
eval ("alert('aaa')")
```

isFinite (numero): Esta función comprueba si el parámetro que se le pasa es un número finito, en caso de no serlo devuelve false y en caso contrario true.

```
isFinite(1/0) // devuelve false  
isFinite(0/1) // devuelve true  
isFinite(«a») // devuelve false
```

parseInt (x) : Pasa a entero el parámetro que se le pasa (string), si no es un número devuelve NaN.

parseFloat(x) : Pasa a float el parámetro que se le pasa (string), si no es un número devuelve NaN.

Funciones predefinidas

isNaN (x) : Devuelve true si el parámetro que se le pasa no es un número y false en caso contrario.

```
isNaN("123123.3") // devuelve false
```

```
isNaN("a") // devuelve true
```

```
isNaN(312) // devuelve true
```

Funciones recursivas

Son las que se llaman a sí mismas. Un ejemplo típico es el de la función que calcula el factorial de un número:

```
function factorial(numero) {  
  if (numero > 1) {  
    return numero * factorial(numero - 1)  
  } else {  
    return numero  
  }  
}
```

Funciones recursivas

La recursividad es confusa y, más aún, peligrosa. No es difícil que una programa recursivo deje sin memoria a la máquina sobre la que se está ejecutando, debido a un algoritmo mal codificado. Incluso cuando el programa es correcto, su ejecución con determinados parámetros de entrada puede requerir tantas llamadas recursivas que llegue a agotar los recursos del sistema (cada llamada implica el almacenamiento de variables de estado y otros parámetros).

Otro inconveniente está en la velocidad. Las sucesivas llamadas a sí misma que realiza la función recursiva ralentizan la ejecución de todo el proceso.

La única gran ventaja de la recursividad está en la reducción, en algunos casos notable, del tamaño del código. Sin embargo, este aspecto nunca pesa más que la creación de un código seguro, rápido y estable.

Funciones – Parámetros y retorno

Ejercicio 7

- Escribir una función que devuelva la suma de dos números. Hacer lo propio con la resta, multiplicación y división.
- El resultado hay que recogerlo en variables y mostrarlo por pantalla.

Ejercicio 8

- Pedir una frase o palabra, y mostrar la longitud de esa frase por pantalla. No es necesario devolver ningún valor.

Ejercicio 9

- Pedir el nombre y el 1er apellido, pasárselo a una función, y que la función imprima en un alert una variable que contenga los dos datos pedidos y separados por un espacio en blanco y pasado todo a mayúsculas

Break y Continue

- Al igual que en Java, estas sentencias permiten cortar o saltar las iteraciones del bucle.
 - **continue**, te puedes saltar alguna iteración.
 - **break** acaba con las iteraciones, finaliza el bucle.

Ejercicio 10

- Tener un array con 3 profes.
- Hacer 2 funciones, una en la que tengas un bucle forEach (for ... in) que muestre el array, pero que pare si encuentra el nombre del 2do profe.
- La otra función se saltará al segundo profe.

Clases de Javascript

JAVASCRIPT

Clase Array

Como sabemos Javascript no posee tipos específicos para datos, y por tanto no tiene un tipo para arreglos; para poder usar arrays podemos usar el objeto primitivo de javascript Array.

Un arreglo es un conjunto de valores organizados mediante índices numéricos.

La sintaxis para la creación de un nuevo array es:

```
var ejemplo_array = new Array(elemento1, elemento2, ..., elemento n);  
var ejemplo_array = new Array(tamaño del array);
```

Para acceder a los elementos de array se usa el operador [], y en su interior un entero que indica la posición en el array. Por ejemplo para acceder al cuarto elemento del Array ejemplo_array

```
ejemplo_array[3]
```

La sentencia anterior devolvería el valor en esa posición del array. Con el operador de asignación podemos modificar el valor en dicha posición. Por ejemplo:

```
ejemplo_array[3] = 128937;
```

Clase Array

Métodos:

- **concat** une dos arrays y retorna un nuevo array con el resultado.

```
var a1 = new Array(1,2,3);  
var a2 = new Array(4,5,6,7);  
var ac = a1.concat(a2); // resultado 1,2,3,4,5,6,7
```

- **join(delimitador = ",")** une todos los elementos del array en una cadena. Si no se le pasa el delimitado a **join** por defecto pone una coma.

```
a1.join("-");//Resultado un string "1-2-3"
```

- **pop()** remueve el último elemento del array y retorna este elemento.

```
a1.pop() // devuelve 3 y deja el array a1 con 1,2
```

- **push(elemento)** Añade un o más elemento al final del array y devuelve el último elemento

```
a1.push(3) // añade un 3 al final del array a1 y devuelve 3
```

Clase Array

Métodos:

- **reverse** invierte el orden de los elementos de un array.

```
a1.reverse();
```

- **shift()** quita el primer elemento de un array y devuelve este elemento.

```
a1.shift();
```

- **slice (inicio, fin)** Extra una sección del array delimitada por los parámetros inicio y fin. Devuelve un array con la sección extraída. El valor indicado por el parámetro **fin** no se incluye en el array.

```
a1.slice(1,3);
```

- **sort()** Ordena el array , por defecto lo hace de menor a mayor (Hemos de tener en cuenta que este método modifica el array).

```
a1.sort();
```

Clase Array

Métodos:

sort también se le puede pasar como parámetro una función que indique la forma en que se debe de ordenar el array. Esta función recibe dos parámetros a y b y ha de devolver tres valores. 1, -1 y 0.

Si devuelve -1 (o cualquier otro número negativo) entiende que a debe de ser considerado menor que b.

Si devuelve 1 (o cualquier número positivo) entiende que a debe de ser considerado mayor que b.

Si devuelve 0 a y b entiende que a y b son iguales.

Esto nos permite crear nuestras propias reglas de ordenación en base a dicha función.

```
var a1 = new Array(1,4,2,6,7);  
var ordena= function (a,b) { if (a>b) return 1; if (a<b) return -1; if (a == b) return 0;};  
a1.sort(ordena);// [1, 2, 4, 6, 7]
```

Clase Array

Métodos:

- **unshift** añade un elemento al inicio del array y devuelve el tamaño de este.

```
a1.reverse();
```

Propiedades:

- **length** : devuelve la longitud del array, el número de elementos.
- **prototype**: Nos permite añadir nuevos métodos y propiedades a la clase Array.
- **constructor** : Devuelve la función constructora de la clase Array

Actividad 1

El objetivo es escribir un programa que realice un cuestionario. El usuario en una fase inicial introducirá las preguntas y su respuesta correcta (prompt). Cuando quiera dejar de introducir respuestas escribirá -1.

En la segunda fase, el programa comenzará a preguntar (prompt), en cada pregunta hay tres oportunidades para responder de forma correcta y si no se marcará como incorrecta. El paso a la siguiente pregunta es automático. Al responder una pregunta siempre se dará un feedback indicando si la respuesta ha sido o no correcta. Al final del cuestionario se indicará el número de aciertos.

Requisito fundamental para resolver esta actividad es usar **funciones recursivas** tanto para almacenar las preguntas como para realizar el test.

OPCIONAL: Al final del cuestionario presentar un primer informe que muestre las preguntas fallas, con su texto y su respuesta correcta. Y un segundo informe a continuación de este que informe del número de intentos en cada pregunta.

NOTA: el programa ha de estar en un archivo **js** aparte del **html**.

Utilidades

En el ámbito web es importante optimizar lo más posible aunque nos parezca poca cosa lo que podemos ganar por ejemplo en lo tocante a peso de archivos.

Unos recursos interesante en este sentido y relacionados con javascript son aquellas aplicaciones que permiten «comprimir» y ofuscar nuestro código.

Su funcionamiento, a grandes rasgos, consiste en eliminar espacios en blanco y «complicar» los nombres de las funciones y variables.

Veamos el siguiente enlace:

<http://dean.edwards.name/packer/>

Utilidades

Este tipo de aplicaciones tienen sentido en archivos js grandes. O si queremos hacer difícil que nos copien nuestro código.

Guardar siempre una versión sin comprimir del código.

Otra página de este tipo (también para CSS), es una aplicación más compleja:

<http://yui.github.com/yuicompressor/>

Clase Boolean

No debemos de confundir los true o false «nativos» con esta clase, que no es otra cosa que un envoltorio para el que los valores null, undefined, no numérico (NaN), una cadena vacía o un false son false.

```
var b = new Boolean(0) // sera «false»  
var b = new Boolean(«Hola») // sera «true»
```

Clase Date

Esta clase se utiliza para trabajar fechas, que se almacenan en milisegundos tomando como referencia la fecha 1 de enero de 1970 a las 0:00:00 como referencia.

Para crear un objeto Date usamos new Date(). Veamos posibles formas de inicializarlo.

- 1.- new Date() crea un objeto Date con la fecha y hora actual
- 2.- Se le pasa como parámetro una cadena de texto con el formato siguiente:

"Mes día, año horas:minutos:segundos." El Mes ha de ir en inglés

```
var fecha = new Date("October 25, 1995 13:30:00");
```

- 3.- Valores enteros para año, mes y día

```
var fecha = new Date(1995,10,25);
```

- 4.- Lo mismo que la anterior forma, pero añadiendo horas, minutos y segundos

```
var fecha = new Date(1995,10,25, 15,01,33);
```

- 5.- Finalmente le podemos pasar el «tiempo» en milisegundos.

Clase Date

Métodos:

| getDate() | Devuelve el día del mes (de 1 a 31) |
|-------------------|--|
| getDay() | Devuelve el día de la semana (de 0 a 6) |
| getFullYear() | Devuelve el año en formato de 4 dígitos |
| getHours() | Devuelve la hora (de 0 a 23) |
| getMilliseconds() | Devuelve milisegundos de la fecha |
| getMinutes() | Devuelve minutos (de 0 a 59) |
| getMonth() | Devuelve el número del mes (de 0 a 11) |
| getSeconds() | Devuelve los segundos de la fecha (entre 0 y 59) |
| getTime() | Devuelve los milisegundos entre la fecha guardada y el 1 de Enero de 1970. |

A cada uno de estos métodos **get** les corresponde un **set** para modificar el objeto Date en el mismo parámetro que devuelve el **get**.

Clase Date

Métodos:

| toString() | Convierte el objeto date en una cadena de texto. |
|------------|--|

Propiedades:

- **prototype**: Nos permite añadir nuevos métodos y propiedades a la clase Date.
- **constructor** : Devuelve la función constructora de la clase Date.

Actividad 2

Escribir un programa que muestre en que cuatrimestre del año nos encontramos y la fecha y hora actuales.



Clase Math

Esta clase nos permite hacer operaciones matemáticas a través de los métodos y constantes que implementa.

Entre los cálculos que permiten está los trigonométricos, que tienen la «particularidad» de usar los ángulos expresados en radianes. ($[\text{grados}] * (\pi/180)$).

La clase Math se usa del modo siguiente:

```
var h = Math.sin(3);
```

Clase Math

Métodos:

| abs(x) | Devuelve el valor absoluto de un número |
|-------------------------|---|
| sin, cos, tan | Funciones trigonométricas estandar |
| acos, asin, atan, atan2 | Funciones trigonométricas inversas, devuelve radianes |
| exp, log | Logaritmos exponencial y natural, base e |
| ceil(x) | Retorna un entero mayor o igual al número que se le pasa como argumento |
| floor(x) | Retorna un entero menor o igual al número que se le pasa como argumento |
| min (n1,n2,...nn); | Devuelve el número con el menor valor de entre todos los que se pasen como argumentos |
| max (n1,n2,...nn); | Devuelve el número con el mayor valor de entre todos los que se pasen como argumentos |
| pow(x,y) | x elevado a y |
| random() | Devuelve un valor aleatorio entre 0 y 1 |

Clase Math

Métodos:

| round(x) | Redondea al entero más próximo. |
|----------|---------------------------------|
| sqrt(x) | Raíz cuadrada de x |

Propiedades:

| E | Contante de euler |
|----|-------------------|
| PI | Número PI |

Actividad 3

- 1) Escribir un programa que pida un número y nos muestre dicho valor elevado a la tercera potencia.
- 2) Escribir un programa que pida un valor y luego mostrar la raíz cuadrada de dicho valor.

Clase String

Esta clase nos permite manejar cadenas de texto.

Se puede crear un objeto de este tipo del modo siguiente:

```
var texto = new String("Cadena de texto")
```

O simplemente

```
var texto = "Cadena de texto"
```

Clase String

Métodos:

| charAt(x) | Devuelve el carácter en la posición que se le pasa como parámetro. |
|-----------------------------|--|
| indexOf(buscar) | Devuelve la posición de la primera ocurrencia de la cadena que se le pasa como parámetro. |
| lastIndexOf(buscar) | Devuelve la última posición de la primera ocurrencia de la cadena que se le pasa como parámetro. |
| replace(buscar, cambiarPor) | Reemplaza el texto buscar por el que se le pasa en cambiarPor. |
| ceil(x) | Retorna un entero mayor o igual al número que se le pasa como argumento |
| split(separador) | Retorna un array de subcadenas de la cadena principal, usando como criterio para «cortarla» el patrón que se le pasa como parámetro. |
| substr(x,y) | Extrae una subcadena de texto desde el punto de la cadena indicado con x con la longitud indicada en y. |
| substring(x,y) | Extrae una subcadena de texto de la principal entre las posiciones indicadas con x e y |

Clase String

Métodos:

| toLowerCase() | Pasa a minúsculas todos los caracteres de la cadena |
|---------------|---|
| toUpperCase() | Pasa a mayúsculas todos los caracteres de la cadena |

Propiedades:

- **Length** : número de caracteres de la cadena
- **Prototype**: Permite añadir nuevos métodos y propiedades a la clase String.

Actividad 4

- 1) Realizar la búsqueda de un string clave en un string fuente. Se deberá meter por prompt una frase o texto (fuente) y luego la clave a buscar. En caso de encontrarla, imprimir la posición, de lo contrario informa del fracaso en la búsqueda.
- 2) Meter una palabra o texto por teclado y determinar si es o no una palabra palíndromo. (Palabra que se lee de igual manera de adelante hacia atrás, que de atrás hacia delante).
- 3) Realizar un programa que permita cargar una dirección de mail e implementar una función que verifique si el String tiene cargado el carácter @.
- 4) Cargar un String por teclado e implementar las siguientes funciones:
 - a) Imprimir la primera mitad de los caracteres de la cadena.
 - b) Imprimir el último carácter.
 - c) Imprimirlo en forma inversa.
 - d) Imprimir cada carácter del String separado con un guión.
 - e) Imprimir la cantidad de vocales almacenadas.

Clase Number

Este objeto es un «envoltorio» para números. A continuación se indican los métodos de Number.

| toExponential(x) | Convierte un número a formato exponencial |
|------------------|---|
| toFixed(x) | Indica cuantos digitos en la parte decimal se han de mostrar. |
| toPrecision(x) | Indica la longitud del numero |
| toString() | Convierto el objeto a una cadena |

```
var n = new Number(3)
```

Clase History

Este objeto (del Browser) contiene las url visitadas por el usuario (desde esa ventana). Con sus métodos podemos recorrerlo

| back() | Carga la URL anterior a la que se este viendo en ese momento |
|-----------|--|
| forward() | Carga la URL siguiente a la que se este viendo en ese momento. |
| go() | Carga una URL específica de la lista de history |

```
history.back()
```

Clase Location

Este objeto (del Browser) contiene información de la url en que se encuentre el navegador.

Nos podemos encontrar con diferencias entre navegadores a la hora de manejar esta clase.

Métodos:

| assign(url); | Carga la URL anterior indicada como parámetro |
|--------------|---|
| reload() | Recargar la URL actual. |
| replace(url) | Carga la URL anterior indicada como parámetro |

```
location.assign("http://www.google.com")
```

Con replace la nueva url no se almacena en el objeto history mientras que con assign sí.

Clase Location

Propiedades:

| host | Devuelve el hostname y el puerto de la URL |
|----------|---|
| hostname | Devuelve el hostname |
| href | Devuelve la URL completa. |
| pathname | Devuelve en path de la URL |
| port | Devuelve al número del puerto del servidor. |
| protocol | Devuelve el protocolo de la URL |
| search | Devuelve el query de la URL |

Clase Navigator

Esta clase contiene información acerca del navegador. Hay que usarla poniendo cuidado en el hecho de que puede tener comportamientos «raros» en algunos navegadores, que además pueden implementar sus propios métodos y propiedades. Veamos a continuación algunas propiedades.

| appCodeName | Cadena que contiene el nombre del código del cliente. |
|-------------|--|
| appName | Cadena que contiene el nombre del cliente. |
| appVersion | Cadena que contiene información sobre la versión del cliente. |
| language | Cadena de dos caracteres que contiene información sobre el idioma de la versión del cliente. |

Clase Navigator

Ejemplo de uso:

```
navigator.appName;
```

Propiedades:

| platform | Cadena con la plataforma sobre la que se está ejecutando el programa cliente. |
|-----------|--|
| plugins | Array que contiene todos los plug-ins soportados por el cliente. |
| userAgent | Cadena que contiene la cabecera completa del agente enviada en una petición HTTP. Contiene la información de las propiedades appName y appVersion. |

Clase Navigator

Métodos:

| <code>javaEnabled()</code> | Devuelve true si el cliente permite la utilización de Java, en caso contrario, devuelve false. |
|----------------------------|--|

Clase Screen

Contiene información acerca de la pantalla del cliente.

```
screen.height;
```

Propiedades:

| availHeight | Devuelve la altura de la ventana, no se incluye la altura de la barra de herramientas |
|-------------|---|
| availWidth | Devuelve el ancho de la ventana, no se incluye el ancho de la barra de scroll |
| colorDepth | Devuelve la profundidad de color |
| height | Devuelve al altura total de la pantalla |
| pixelDepth | Devuelve la resolución de color en bits por pixel |
| width | Devuelve el ancho total de la pantalla |

Actividad 5

Escribe un programa que pregunte al usuario los siguientes datos de 4 clientes:

- Nombre y apellidos (se almacenaran con todas las letras en mayúsculas. String)
- Edad
- DNI (validar entre 0 y 99 millones y que tiene letra al final, si es incorrecto volver a preguntar)
- Numero de teléfono.

Una vez tengamos los datos almacenados(Array) mostrará todos los clientes en pantalla ordenados por edades y de menor a mayor. Esto podemos hacerlo mediante sucesivos alerts o generando de forma «dinámica» el código html necesario con **document.write**.

Este mismo programa y mediante una función **buscar()** que llamaremos desde la consola nos permitirá encontrar un usuario usando su DNI (un prompt nos preguntará ese dato). Si la búsqueda tiene éxito mostraremos en pantalla los datos del usuario pero en este caso el nombre y los apellidos han de ir todas las letras en minúsculas. Si la búsqueda no tiene éxito volverá a preguntar por un DNI.

NOTA: el programa ha de estar en un archivo js aparte del html.

Clase RegExp

Sirve para instanciar un objeto de tipo «expresión regular» para encontrar texto a partir de un patrón.

Veamos un ejemplo:

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
  var valor=prompt('Escribe un numero entero positivo de 3 dígitos','');
  var patron=new RegExp('^[0-9]{3}$');
  //var patron=/^[0-9]{3}$/; // También se puede definir así una expresión regular
  if (patron.test(valor)){
    alert('Ok, valor entero positivo de 3 dígitos');
  } else{
    alert('No es un valor entero positivo de 3 dígitos');
  }
</script>
</body>
</html>
```

Actividad 6

En base a lo que acabos de ver escribe un programa que pida un texto (prompt) y comprueba si se contiene la palabra «java». Informa de ello mediante un alert.



Clase RegExp

Carácteres especiales en expresiones regulares

Tienen un significado especial en la definición del lenguaje de expresiones regulares, estos tipos de caracteres suelen llamarse metacaracteres.

| ^ | <p>Principio de cadena.</p> <p>Cuando queremos controlar si un string comienza con un determinado carácter o conjunto de caracteres podemos implementar una expresión regular antecediendo al o los caracteres el símbolo ^.</p> |
|----|--|
| \$ | <p>Cuando queremos verificar si una cadena finaliza con un determinado carácter o conjunto de caracteres debemos poner al final el carácter \$ y antecederle el o los caracteres a verificar.</p> |

Clase RegExp

Caracteres especiales en expresiones regulares

| [] | <p>Conjunto de caracteres opcionales. Se emplea estos caracteres para encerrar los caracteres permitidos. Por ejemplo palabra que empieza por mayúsculas: <code>^[A-Z]</code>. Si necesitamos usar uno de los caracteres especiales debemos de escaparlos (<code>\</code>).</p> |
|-----|---|

Clase RegExp

Caracteres especiales en expresiones regulares

| $\{x\}$ $\{x,y\}$ $\{x,\}$ | <p>Cuantificadores o repeticiones . Indican el numero de veces que se ha de repetir lo que antecede al cuantificador.</p> <ul style="list-style-type: none">- Contiene 3 dígitos : $^{[0-9]\{3\}}$- Contiene entre 3 y 5 dígitos : $^{[0-9]\{3,5\}}$- Contiene 3 o más dígitos : $^{[0-9]\{3,\}}$ |
|----------------------------|--|

Clase RegExp

Caracteres especiales en expresiones regulares

| * | Equivale a $\{0,\}$ |
|---|---|
| + | Equivale a $\{1,\}$ |
| ? | Equivale a $\{0,1\}$ |
| | Alternacia. Permite analizar entre varias opciones posibles. Por ejemplo si una palabra debe de empezar por H o por h $/^{[H h]}/$ |

Clase RegExp

Metodos:

| test(cadena) | Devuelve true o false en función de que el patrón coincida o no. |
|--------------|---|
| exec(cadena) | Devuelve un array si el patrón coincide o null en caso de no coincidir. |

Actividad 7

- 1) Validar un número de 3 dígitos enteros, el carácter punto y 2 dígitos decimales.
- 2) Solicitar la coordenada de un punto. El formato a ingresar por teclado es:
(999,999) Los números pueden tener entre 1 y 3 dígitos.

Eventos

JAVASCRIPT

Eventos

Hasta ahora todo lo que hemos hecho en JavaScript se ejecuta de principio a fin en el momento que se carga nuestro documento HTML de forma secuencial salvo por las estructuras de control de flujo.

Este tipo de programación no se hace cargo de la interacción del usuario con nuestra página, algo fundamental en las aplicaciones web. El recurso que tenemos que utilizar para resolver esto es la «programación orientada a eventos», con este tipo de programación nuestros scripts esperaran determinados eventos que especificaremos para desencadenar las acciones que establezcamos.

Así podemos registrar un evento en un enlace (<a>) con el uso combinado de la selección de nodos vista anteriormente y asignarle a ese evento un manejador, una función que se ejecutará cuando ocurra el evento registrado.

Eventos

En el siguiente ejemplo, preguntaremos al usuario si está seguro de querer ir a la url del enlace que tiene el documento:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      window.onload = function(){
        var enlace = document.getElementById('ejemplo');
        enlace.onclick=function (){          //le asignamos el manejador del evento clic

          /*La función confirm de JavaScript, lanza un dialogo con dos botones
          * uno para aceptar y otro para cancelar. La función devuelve true o false
          * según se pulse aceptar o cancelar respectivamente.
          * Por otro lado, un return false en este manejador evita que se haga el
          * salto a la página
          */
          return confirm('¿Está seguro de ir a esta página?');

        };
      }
    </script>
  </head>
  <body>
    <a id="ejemplo" href="http://www.google.com">Enlace</a>
  </body>
</html>>
```

Eventos

Hay gran cantidad de eventos, y varias maneras de aplicarles manejadores.

Se pueden aplicar, además de la forma que hemos visto en el ejemplo anterior denominada «de manejadores semánticos» basada en asignar un id único al elemento al que le queremos asignar el manejador, directamente en el código html:

```
<a href='#' onclick='function (){alert('Ha hecho clic')} '>
```

Esta forma «ensucia» el HTML y es recomendable evitarla en la medida de lo posible.

Finalmente, hay otra forma aunque en el momento de escribir este texto en IE9 no funciona (usar attachEvent):

```
elemento.addEventListener(tipo, manejador);
```

```
enlace.addEventListener('click', function(){  
    return confirm('¿Está seguro de ir a esta página?');  
});
```

Listado de eventos

Fuente: Introducción a JavaScript (Javier Eguiluz Pérez)

| Evento | Descripción | Elementos para los que está definido |
|-------------|---|---|
| onblur | Deseleccionar el elemento | <button>, <input>, <label>, <select>,<textarea>, <body> |
| onchange | Deseleccionar un elemento que se ha modificado | <input>, <select>, <textarea> |
| onclick | Pinchar y soltar el ratón | Todos los elementos |
| ondblclick | Pinchar dos veces seguidas con el ratón | Todos los elementos |
| onfocus | Seleccionar un elemento | <button>, <input>, <label>, <select>,<textarea>, <body> |
| onkeydown | Pulsar una tecla (sin soltar) | Elementos de formulario y <body> |
| onkeypress | Pulsar una tecla | Elementos de formulario y <body> |
| onkeyup | Soltar una tecla pulsada | Elementos de formulario y <body> |
| onload | La página se ha cargado completamente | <body> |
| onmousedown | Pulsar (sin soltar) un botón del ratón | Todos los elementos |
| onmousemove | Mover el ratón | Todos los elementos |
| onmouseout | El ratón "sale" del elemento (pasa por encima de otro elemento) | Todos los elementos |

| Evento | Descripción | Elementos para los que está definido |
|-------------|--|--------------------------------------|
| onmouseover | El ratón "entra" en el elemento (pasa por encima del elemento) | Todos los elementos |
| onmouseup | Soltar el botón que estaba pulsado en el ratón | Todos los elementos |
| onreset | Inicializar el formulario (borrar todos sus datos) | <form> |
| onresize | Se ha modificado el tamaño de la ventana del navegador | <body> |
| onselect | Seleccionar un texto | <input>, <textarea> |
| onsubmit | Enviar el formulario | <form> |
| onunload | Se abandona la página (por ejemplo al cerrar el navegador) | <body> |

Observaciones

Acabamos de ver que la asignación de eventos en algunos casos puede plantear problemas de compatibilidad con algunos navegadores, una forma de evitar estos problemas es usar **jQuery**.

Una librería de JavaScript que veremos en el siguiente bloque del curso.

jQuery (I) - Introducción

JAVASCRIPT

jQuery

Nos encontramos con una librería de Javascript que puede resultarnos muy útil a la hora de realizar aplicaciones web.

La mejor forma de describirla es la que hacen sus autores en la página web de jQuery (<http://jquery.com/>), y que se puede resumir como una librería que nos facilita el trabajo de programación de forma notable; y que ahora con la nueva versión para desarrollo de aplicaciones para móviles(<http://jquerymobile.com/>) nos abre la puerta a este «nuevo mundo» manteniendo la sencillez en su uso.

NOTA: Hay otras librerías a nuestra disposición, algunas más complejas y potentes, como es el caso de la desarrollada por Sencha, Ext JS (<http://www.sencha.com/products/extjs3/>). Sencha también tiene una versión de Ext JS para móviles, Sencha Touch (<http://www.sencha.com/products/touch/>)

¿Porqué usar jQuery?

Las aplicaciones web están ganando en complejidad tanto desde el punto de vista del qué hacen y cómo, como desde el punto de vista del cómo muestran el resultado de lo que hacen.

Por otro lado es fundamental que una aplicación funcione en los navegadores más utilizados (Chrome, Firefox, IE, Opera, Safari).

Y finalmente, los tiempos de producción de las aplicaciones se hacen más pequeños. jQuery nos aporta soluciones a estas necesidades y es bastante rápido de aprender.

Empezando a utilizar jQuery

jQuery está en un archivo js y para tenerlo a nuestra disposición basta con enlazarlo a nuestros documentos html:

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

Veamos como lo podemos descargar.

Descargando jQuery

En <http://jquery.com/> podemos descargar la librería, hay dos versiones, una de desarrollo y otra de producción. Ambas hacen lo mismo pero tiene pesos distintos. Sin entrar en aspectos técnicos de porque hay dos versiones, nosotros usaremos la versión de desarrollo. Pero tal como indican sus nombres la de desarrollo es la más adecuada para el desarrollo de aplicaciones y la de producción es la adecuada para poner en el servidor de producción la aplicación una vez terminada.

Una opción a la descarga es cargar jquery desde un servidor remoto, como el caso de google, se hace del modo siguiente :

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.0/jquery.min.js" type="text/javascript"></script>
```

Vemos un ejemplo de uso de jQuery descargando el archivo js.

Hemos descargado ya el archivo js que contiene la librería, lo hemos llamado «jquery.js» y lo hemos guardado en la misma carpeta en la que crearemos los documentos html.

Ahora creamos un nuevo documento html, el nombre será prac1.html y el código es el siguiente:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery</title>
  <script type="text/javascript" src="jquery.js"></script>
  <script type="text/javascript">
    // Aquí escribiremos nuestro código
  </script>
</head>
<body>
  <!-- HTML aquí-->
</body>
</html>
```

Si abrimos este documento en un navegador no veremos nada ya que ni hay contenidos en el cuerpo del documento, ni hay programación alguna en Javascript. Este ejemplo lo usaremos como plantilla.

Probemos a escribir lo siguiente en el lugar de `//Aquí escribiremos nuestro código` :

```
$(document).ready(function (){  
    alert('Hola con jQuery');  
});
```

Esto es equivalente, si no usáramos jQuery , a :

```
window.onload = function (){alert('Hola sin jQuery');};
```

Analicemos este código coloreado.

`$()` nos permite hacer referencia a uno o más elementos del DOM en base al selector que le pasemos como parámetro, en este caso el objeto document.

`ready()` le asignamos el listener onload al resultado de la función `$()` y le pasamos como parámetro un manejador para este evento, en este caso una función que lanza un alert .

Ejercicio (Ref. jquery_001)

Con jQuery lanzar un alert con un texto una vez se han cargado todos los elementos del DOM.

OBJETIVOS:

Iniciar jQuery.

Selectores de jQuery

Los selectores, entendiendo como tales los que le pasamos a la función `$()` como parámetros, son de dos tipos, un objeto (algo que ya hemos visto en ejemplo anterior con `document`) o una combinación de selectores CSS y Xpath.

Selectores sencillos:

`$('a')` → selecciona todas las etiquetas tipo `<a>`

`$('.negrita')` → selecciona todos los elementos que tiene el atributo «class» y cuyo valor es «negrita»

`$('#titulo')` → selecciona el elemento que tiene el atributo «id» y cuyo valor es «titulo»

Selectores de jQuery

Selectores avanzados:

Permiten selecciones más precisas.

Selección por atributo

Sintaxis:

`$(elemento["nombre_atributo='valor']")` → Recupera los elementos del tipo indicado, con el atributo especificado en el parámetro `nombre_atributo` y cuyo valor coincida con lo indicado.

`$("a[rel='slide'] ")`

En este ejemplo seleccionamos todos los elemento de tipo ancla `<a>` que tienen al tributo `rel` con el valor «slide».

Dentro de este selector hay varios tipos:

`$(elemento["nombre_atributo!='valor']")` → Recupera los que no tiene el atributo indicado, o aunque lo tengan su valor es diferente del especificado

Selectores de jQuery

`$(elemento["nombre_atributo$='valor'])` → Recupera los elementos indicados con el atributo especificado cuyo valor termina con lo indicado en el parámetro valor.

`$(elemento["nombre_atributo*='valor'])` → Recupera los elementos indicados con el atributo especificado cuyo valor contiene la cadena indicada en el parámetro valor.

Puede sernos útil una vez hemos hecho la selección filtrar el resultado, o bien movernos por el DOM a partir de un elemento concreto dentro de la selección.



Selectores de jQuery

Filtros

Los filtros se aplican sobre la selección para aumentar la precisión del resultado de la selección. Veamos la sintaxis de algunos de ellos y lo que hacen.

\$(selector:filtro)

\$(div:first) → selecciona de todos los div del documento, el primero.

:first → Devuelve el primer elemento de la selección

:last → Devuelve el último elemento de la selección

:gt(número) → Devuelve todos los elementos cuyo índice en el resultado de la selección sea mayor al indicado en el parámetro número.

:lt(número) → Devuelve todos los elementos cuyo índice en el resultado de la selección sea menor al indicado en el parámetro número.

:eq(número) → Devuelve el elemento cuyo índice en el resultado de la selección sea igual al indicado en el parámetro número.

:not → Devuelve todos los elementos que no coinciden con el selector.

Modificación de estilos CSS mediante jQuery

El método `css()` sirve tanto para obtener el valor de una propiedad CSS como para asignarle un nuevo valor. La sintaxis es la siguiente:

```
$('.selector').css('nombre propiedad','nuevo valor');
```

Si no indicamos un nuevo valor, la expresión :

```
$('.selector').css('nombre propiedad');
```

Devolvería en valor de la propiedad.

Modificación de estilos CSS mediante jQuery

Un ejemplo del cambio de un estilo podría ser el siguiente , en el que se cambiar a color blanco el color del texto de todos los enlaces del documento:

```
$('.a').css('color','#ffffff');
```

Es muy probable que tengamos que cambiar más de una propiedad a la vez, esto lo podemos hacer con la siguiente sintaxis:

```
$('.a').css('color':'#ffffff',  
           'text-decoration':'none',  
           'font-size':'10px');
```

Efectos

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery</title>
  <script type="text/javascript" src="jquery.js"></script>
  <script type="text/javascript">
    $(document).ready(function (){
      $(".e1").delay(500).hide();
      $(".e2") .hide().delay(1000).show("slow");
      $(".e4").delay(2000).slideToggle("fast");
      $(".e5").delay(4000).fadeOut();
    });
  </script>
</head>
<body>
  <ul>
    <li class="e1">Mira aquí 2</li>
    <li class="e2"><a href="#">Efecto hide</a></li>
    <li class="e3"><a href="#">Efecto show()</a></li>
    <li class="e4"><a href="#">Efecto slideToggle</a></li>
    <li class="e5"><a href="#">Efecto fadeOut</a></li>
  </ul>
</body>
</html>
```

Efectos

Analicemos el código de uno de los efectos:

```
$("#e1").delay(500).hide();
```

En este caso se selecciona el elemento con el atributo class igual a «e1» y le aplicamos el efecto «hide» que lo hace desaparecer. El método delay retrasa 500 milisegundos hide().

Cada uno de los efectos se puede parametrizar para ajustar ese efecto a nuestra necesidades.

Ejercicio (Ref. jquery_002)

Usando el ejemplo anterior, al hacer click en los enlaces de la lista que desencadenen los efectos programados.

Retira el delay para evitar esperas.



Cómo recorrer el resultado de una selección con jQuery

La forma es utilizando el método de jQuery **each()**, este método aplica una función a cada uno de los elementos de la selección, esta función tiene como parámetros en índice de ese elemento en la selección y una referencia al propio elemento.

Veamos un ejemplo.

```
$('li').each(function(indice, elemento) {  
    // this, dentro de estas funciones, hace referencia también al elemento  
    console.log(  
        'El elemento ' + indice +  
        'contiene el siguiente HTML: ' +  
        $(elemento).html()  
    );  
});
```

Cómo recorrer el resultado de una selección con jQuery

Otras formas:

```
$('#h1').next('p');           // seleccionar el inmediato y más próximo elemento <p> con respecto a H1  
$('#miDiv').parent();        // seleccionar el elemento contenedor de #miDiv  
$('input[name=first_name]').closest('form'); // seleccionar el elemento <form> más cercano a un input  
$('#myList').children();      // seleccionar todos los elementos hijos de #myList  
$('li').siblings();           // seleccionar todos los items hermanos del elemento <li>
```


Ejercicio (Ref. jquery_004)

Utilizando el html que se puede encontrar en el archivo jquery_004.zip que acompañan a este documento y mediante jQuery modificar la altura de los elementos de lista () para que todos ellos tengan la misma altura.

Para resolver este ejercicio es útil el método .height() que devuelve el valor de la altura de un elemento y al que se le pasa un valor numérico como parámetro le asigna ese valor como altura al elemento.

```
$(elemento).height(20); // el elemento pasará a medir 20px de alto
```