Capítulo 3
JavaScript, jQuery, JSON y Ajax

Objetivo

Al finalizar el capítulo, el alumno:

- Comprende el modelo DOM de una página HTML.
- Construye páginas básicas con HTML.
- Desarrolla funciones javascript para manipular los elementos HTML.
- Hace uso de JQuery,Ajax y JSON para enviar información al servidor.

Temas

1. JavaScript
   - Tipos de variables
   - Operadores
   - Instrucciones de control - Condicionales
   - Instrucciones de control – Bucles
   - Creación de objetos
   - Arrays
   - Document Object Model (DOM)
   - Manejo de eventos
   - Local y Session Storage
2. jQuery
   - Selectores jQuery
   - Eventos
   - Ajax con jQuery
3. JSON: creación y uso
4. AJAX con jquery

1. JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. También existe una forma de JavaScript del lado del servidor, por ejemplo, su uso en documentos PDF y aplicaciones de escritorio (mayoritariamente widgets) es significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

JavaScript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a LiveScript, para finalmente, quedar como JavaScript. El cambio de nombre coincidió aproximadamente con el momento en que Netscape agregó soporte para la tecnología Java en su navegador web Netscape en diciembre de 1995.

La denominación produjo confusión, dando la impresión de que el lenguaje es una prolongación de Java, y se ha caracterizado, por muchos, como una estrategia de mercadotecnia de Netscape para obtener prestigio e innovar en lo que eran los nuevos lenguajes de programación web.

Referencia: http://es.wikipedia.org/wiki/JavaScript

1.1 Tipo de Variables

Aunque todas las variables de JavaScript se crean de la misma forma (mediante la palabra reservada var), la manera de asignarle un valor depende del tipo de dato que se quiere almacenar (números, textos, etc.).

- **Numéricas**
  Se utilizan para almacenar valores numéricos enteros (integer) o decimales (float). En este caso, el valor se asigna indicando directamente el número entero o decimal. Los números decimales utilizan el caracter punto, en vez de coma, para separar la parte entera y la parte decimal.

```
var igv = 18;         // variable tipo entero
var total = 134.46;   // variable tipo decimal
```

- **Cadenas de texto**
  Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final.

```
var mensaje = "Bienvenido a DAT";
var nombreProducto = 'Visual Studio 2012';
var letraSeleccionada = 'a';
```

El texto que se almacena en las variables, en ocasiones no es tan sencillo. Si por ejemplo, el propio texto contiene comillas simples o dobles, la estrategia que se sigue es la de encerrar el texto con las comillas (simples o dobles) que no utilice el texto.

```
/* El contenido de texto1 tiene comillas simples, por lo que
se encierra con comillas dobles */
var texto1 = "El tema de 'Javascript' es demasiado sencillo";

/* El contenido de texto2 tiene comillas dobles, por lo que
    se encierra con comillas simples */
var texto2 = 'El tema de "Javascript" es demasiado sencillo';
```

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc.). Para resolver estos problemas, JavaScript define un mecanismo para incluir de forma sencilla, caracteres especiales y problemáticos dentro de una cadena de texto. El mecanismo consiste en sustituir el caracter problemático por una combinación simple de caracteres. A continuación, se muestra la tabla de conversión que se debe utilizar:

Si se desea incluir	Se debe incluir
Una nueva línea	\n
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\"
Una barra inclinada	\\

Con esto, el ejemplo anterior que contenía comillas simples y dobles dentro del texto, se puede rehacer de la siguiente forma:

```
var texto1 = 'El tema de \'Javascript\' es demasiado sencillo';
var texto2 = "El tema de \"Javascript\" es demasiado sencillo";
```

Este mecanismo de JavaScript se denomina "mecanismo de escape" de los caracteres problemáticos y es habitual referirse a que los caracteres han sido "escapados".

- **Arrays**

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente.

A veces se les llama vectores, matrices e incluso arreglos. No obstante, el término array es el más utilizado y es una palabra comúnmente aceptada en el entorno de la programación.

Por ejemplo, si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto:

```
var dia1 = "Lunes";
var dia2 = "Martes";

var dia7 = "Domingo";
```

Aunque el código anterior no es incorrecto, es poco eficiente y complica en exceso la programación. Si en vez de los días de la semana, se tuviera que guardar el nombre de los meses del año, el nombre de todos los países del

mundo o las mediciones diarias de temperatura de los últimos 100 años, se tendrían que crear decenas o cientos de variables.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array; con lo que el ejemplo anterior se puede rehacer de la siguiente forma:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
```

Una vez definido un array, es muy sencillo acceder a cada uno de sus elementos. A cada elemento se accede indicando su posición dentro del array, empezando el índice en 0.

```
var diaSeleccionado = dias[0];    // diaSeleccionado = "Lunes"
var otroDia = dias[5];            // otroDia = "Sábado"
```

- **Booleanos**

Las variables de tipo boolean o booleano también se conocen con el nombre de variables de tipo lógico. Este tipo de variable, almacena un tipo especial de valor que solamente puede tomar dos valores: true (verdadero) o false (falso).
No se puede utilizar para almacenar números y tampoco permite guardar cadenas de texto.

```
var flagCliente = true;
var flagRegistro = false;
```

- **Fecha**

Para trabajar con fechas en javascript se utiliza la clase Date. Esta clase permite obtener la fecha y hora actual. Permite indicar una fecha determinada, así como obtener parte de una fecha (día, mes o año).

```
miFecha = new Date() //Permite obtener la fecha y hora actual
```

Obtener la fecha actual.

```
var dt = new Date();

// Display the month, day, and year. getMonth() returns a 0-based number.
var month = dt.getMonth()+1;
var day = dt.getDate();
var year = dt.getFullYear();
document.write(month + '-' + day + '-' + year);

// Output: current month, day, year
```

Puede establecer una fecha concreta pasando una cadena de fecha al constructor.

```
var dt = new Date('8/24/2009');
document.write(dt);

// Output: Mon Aug 24 00:00:00 PDT 2009
```

1.2 Operadores

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables. De esta forma, los operadores permiten a los programas realizar cálculos complejos y tomar decisiones lógicas en función de comparaciones y otros tipos de condiciones.

- **Asignación**

  El operador de asignación es el más utilizado y el más sencillo. Se utiliza para guardar un valor específico en una variable. El símbolo utilizado es =.

```
var numero1 = 3;
```

  A la izquierda del operador, siempre debe indicarse el nombre de una variable. A la derecha del operador, se pueden indicar variables, valores, condiciones lógicas, etc.

```
var numero1 = 3;
var numero2 = 4;

/* Error, la asignación siempre se realiza a una variable,
    por lo que en la izquierda no se puede indicar un número */
5 = numero1;

// Ahora, la variable numero1 vale 5
numero1 = 5;

// Ahora, la variable numero1 vale 4
numero1 = numero2;
```

- **Incremento y decremento**

  Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o disminuir en una unidad, el valor de una variable.
  El operador de incremento se indica mediante el prefijo ++ en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad.

```
var numero = 5;
++numero;
alert(numero);  // numero = 6
```

El operador decremento se indica mediante el prefijo -- en el nombre de la variable. El resultado es que el valor de esa variable se decrementa en una unidad:

```
var numero = 5;
--numero;
alert(numero);  // numero = 4
```

Los operadores de incremento y decremento no solamente se pueden indicar como prefijo del nombre de la variable (tal como se muestra en los ejemplos anteriores), sino que también es posible utilizarlos como sufijo. En este caso, su comportamiento es similar.

Puede parecer que es equivalente indicar el operador ++ delante o detrás del identificador de la variable. Sin embargo, el siguiente ejemplo muestra sus diferencias:

```
var numero1 = 5;
var numero2 = 2;
numero3 = numero1++ + numero2;
// numero3 = 7, numero1 = 6

var numero1 = 5;
var numero2 = 2;
numero3 = ++numero1 + numero2;
// numero3 = 8, numero1 = 6
```

Si el operador ++ se indica como prefijo del identificador de la variable, su valor se incrementa antes de realizar cualquier otra operación; pero, si se indica como sufijo del identificador de la variable, su valor se incrementa después de ejecutar la sentencia en la que aparece.

Por tanto, en la instrucción numero3 = numero1++ + numero2; el valor de numero1 se incrementa después de realizar la operación (primero se suma y numero3 vale 7, después se incrementa el valor de numero1 y vale 6). Sin embargo, en la instrucción numero3 = ++numero1 + numero2, en primer lugar, se incrementa el valor de numero1 y después se realiza la suma (primero se incrementa numero1 y vale 6, después se realiza la suma y numero3 vale 8).

- **Lógicos**

Los operadores lógicos son imprescindibles para realizar aplicaciones complejas, ya que se utilizan para tomar decisiones sobre las instrucciones que debería ejecutar el programa en función de ciertas condiciones.
El resultado de cualquier operación que utilice operadores lógicos, siempre es un valor booleano (lógico).

  – **Negación:** Es uno de los operadores lógicos más utilizados. Se utiliza para obtener el valor contrario al valor de la variable:

```
var visible = true;
alert(!visible);  // Muestra "false" y no "true"
```

La negación lógica se obtiene prefijando el símbolo (!) al identificador de la variable. Su funcionamiento se resume en la siguiente tabla:

variable	!variable
true	false
false	true

Si la variable original es de tipo booleano, es muy sencillo obtener su negación. Sin embargo, ¿qué sucede cuando la variable es un número o una cadena de texto? Para obtener la negación en este tipo de variables, se realiza en primer lugar, su conversión a un valor booleano:

1. Si la variable contiene un número, se transforma en false si vale 0 y en true para cualquier otro número (positivo o negativo, decimal o entero).
2. Si la variable contiene una cadena de texto, se transforma en false si la cadena es vacía ("") y en true, en cualquier otro caso.

```
var cantidad = 0;
vacio = !cantidad;  // vacio = true

cantidad = 2;
vacio = !cantidad;  // vacio = false

var mensaje = "";
mensajeVacio = !mensaje;  // mensajeVacio = true

mensaje = "Bienvenido";
mensajeVacio = !mensaje;  // mensajeVacio = false
```

– **AND:** Esta operación lógica obtiene su resultado, combinando dos valores booleanos. El operador se indica mediante el símbolo && y su resultado solamente es true si los dos operandos son true:

variable1	variable2	variable1 && variable2
true	true	true
true	false	false
false	true	false
false	false	false

```
var valor1 = true;
var valor2 = false;
resultado = valor1 && valor2; // resultado = false

valor1 = true;
valor2 = true;
resultado = valor1 && valor2; // resultado = true
```

– **OR:** Esta operación lógica, también combina dos valores booleanos. El operador se indica mediante el símbolo || y su resultado es true si alguno de los dos operandos es true:

variable1	variable2	variable1 || variable2
true	true	true
true	false	true
false	true	true
false	false	false

- **Matemáticos**

  JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas. Los operadores definidos son: suma (+), resta (-), multiplicación (*) y división (/).

```
var numero1 = 10;
var numero2 = 5;

resultado = numero1 / numero2;  // resultado = 2
resultado = 3 + numero1;        // resultado = 13
resultado = numero2 - 4;        // resultado = 1
resultado = numero1 * numero2; // resultado = 50
```

  Además de los cuatro operadores básicos, JavaScript define otro operador matemático que es muy útil en algunas ocasiones. Se trata del operador "módulo", que calcula el resto de la división entera de dos números.

  Si se divide por ejemplo 10 y 5, la división es exacta y da un resultado de 2, por lo que módulo es igual a 0.

  Sin embargo, si se divide 9 y 5, la división no es exacta, el resultado es 1 y el resto 4, por lo que módulo es igual a 4.

El operador módulo en JavaScript se indica mediante el símbolo %, que no debe confundirse con un cálculo del porcentaje:

```
var numero1 = 10;
var numero2 = 5;
resultado = numero1 % numero2; // resultado = 0

numero1 = 9;
numero2 = 5;
resultado = numero1 % numero2; // resultado = 4
```

Los operadores matemáticos también se pueden combinar con el operador de asignación para abreviar su notación:

```
var numero1 = 5;
numero1 += 3;  // numero1 = numero1 + 3 = 8
numero1 -= 1;  // numero1 = numero1 - 1 = 4
numero1 *= 2;   // numero1 = numero1 * 2 = 10
numero1 /= 5;   // numero1 = numero1 / 5 = 1
numero1 %= 4;   // numero1 = numero1 % 4 = 1
```

- **Relacionales**
-

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas: mayor que (>), menor que (<), mayor o igual (>=), menor o igual (<=), igual que (==) y distinto de (!=).

Los operadores que relacionan variables son imprescindibles para realizar cualquier aplicación compleja. El resultado de todos estos operadores siempre es un valor booleano:

```
var numero1 = 3;
var numero2 = 5;
resultado = numero1 > numero2; // resultado = false
resultado = numero1 < numero2; // resultado = true

numero1 = 5;
numero2 = 5;
resultado = numero1 >= numero2; // resultado = true
resultado = numero1 <= numero2; // resultado = true
resultado = numero1 == numero2; // resultado = true
resultado = numero1 != numero2; // resultado = false
```

Se debe tener especial cuidado con el operador de igualdad (==), ya que es el origen de la mayoría de errores de programación, incluso para los usuarios que ya tienen cierta experiencia desarrollando scripts. El operador == se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador =, que se utiliza para asignar un valor a una variable:

```
// El operador "=" asigna valores
var numero1 = 5;
resultado = numero1 = 3;  // numero1 = 3 y resultado = 3

// El operador "==" compara variables
var numero1 = 5;
resultado = numero1 == 3; // numero1 = 5 y resultado = false
```

Los operadores relacionales también se pueden utilizar con variables de tipo cadena de texto:

```
var texto1 = "hola";
var texto2 = "hola";
var texto3 = "adios";

resultado = texto1 == texto3; // resultado = false
resultado = texto1 != texto2; // resultado = false
resultado = texto3 >= texto2; // resultado = false
```

Cuando se utilizan cadenas de texto, los operadores "mayor que" (>) y "menor que" (<) siguen un razonamiento no intuitivo: se compara letra a letra, comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra, las mayúsculas se consideran menores que las minúsculas y las primeras letras del alfabeto son menores que las últimas (a es menor que b, b es menor que c, A es menor que a, etc.).

1.3 Instrucciones de control - Condicionales

Los programas que se pueden realizar utilizando solamente variables y operadores, son una simple sucesión lineal de instrucciones básicas, con lo que no se puede realizar programas que, por ejemplo, muestren un mensaje si el valor de una variable es igual o no a un valor determinado.

Para realizar este tipo de programas, es necesario utilizar las estructuras de control de flujo, que son instrucciones del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

- **Estructura if...else**

  La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Se emplea para tomar decisiones en función de una condición.

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del bloque {...}. Si la condición no se cumple (si su valor es false) no se ejecuta ninguna instrucción contenida en el bloque {...} y el programa continúa ejecutando el resto de instrucciones del script.

```
var mostrarMensaje = true;

if (mostrarMensaje) {
    alert("Hola Mundo");
}
```

En el ejemplo anterior, el mensaje se muestra al usuario, ya que la variable mostrarMensaje tiene un valor de true y por tanto, el programa entra en el bloque de instrucciones del if. El ejemplo se podría reescribir también como:

```
var mostrarMensaje = true;

if (mostrarMensaje == true) {
    alert("Hola Mundo");
}
```

En este caso, la condición es una comparación entre el valor de la variable mostrarMensaje y el valor true. Como los dos valores coinciden, la igualdad se cumple y por tanto, la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

La condición que controla el if, puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente:

```
var mostrado = false;

if (!mostrado) {
    alert("Es la primera vez que se muestra el mensaje");
}
```

Los operadores AND y OR permiten encadenar varias condiciones simples para construir condiciones complejas:

```
var mostrado = false;
var usuarioPermiteMensajes = true;

if (!mostrado && usuarioPermiteMensajes) {
    alert("Es la primera vez que se muestra el mensaje");
}
```

La condición anterior está formada por una operación AND sobre dos variables. A su vez, a la primera variable se le aplica el operador de negación antes de realizar la operación AND. De esta forma, como el valor mostrado es false, el valor !mostrado, sería true. Como la variable usuarioPermiteMensajes es true, el resultado sería igual a true && true, por lo que el resultado final de la condición del if, sería true y por tanto, se ejecutan las instrucciones que se encuentran dentro del bloque del if.

En ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro". Para este segundo tipo de decisiones, existe la estructura if...else. Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if. Si la condición no se cumple (si su valor es false) se ejecutan todas las instrucciones contenidas en else.

```
var edad = 18;

if (edad >= 18) {
    alert("Eres mayor de edad");
}
else {
    alert("Todavía eres menor de edad");
}
```

Si el valor de la variable edad es mayor o igual que el valor numérico 18, la condición del if se cumple y por tanto, se ejecutan sus instrucciones y se muestra el mensaje "Eres mayor de edad". Sin embargo, cuando el valor de la variable edad no es igual o mayor que 18, la condición del if no se cumple, por lo que automáticamente se ejecutan todas las instrucciones del bloque else. En este caso, se mostraría el mensaje "Todavía eres menor de edad".

El siguiente ejemplo compara variables de tipo cadena de texto:

```
var nombre = "";

if (nombre == "") {
    alert("Aún no nos has dicho tu nombre");
}
else {
    alert("Hemos guardado tu nombre");
}
```

La condición del if anterior se construye mediante el operador ==, que es el que se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores). En el ejemplo anterior, si la cadena de texto almacenada en la variable nombre es vacía (es decir, es igual a "") se muestra el mensaje definido en el if. En otro caso, se muestra el mensaje definido en el bloque else.

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

```
if (edad < 12) {
    alert("Todavía eres muy pequeño");
}
else if (edad < 19) {
    alert("Eres un adolescente");
}
else if (edad < 35) {
    alert("Aun sigues siendo joven");
}
else {
    alert("Piensa en cuidarte un poco más");
}
```

No es obligatorio que la combinación de estructuras if...else, acabe con la instrucción else, ya que puede terminar con una instrucción de tipo else if.

- **Estructura Switch**

Sirve para tomar decisiones en función de distintos estados de las variables. Esta expresión se utiliza cuando tenemos múltiples posibilidades como resultado de la evaluación de una sentencia:

```
switch (expresión) {
    case valor1:
        ...
        break
    case valor2:
        ...
        break
    case valor3:
        ...
        break
    default:
        ...
}
```

La expresión se evalúa, si vale valor1 se ejecutan las sentencias relacionadas con ese caso. Si la expresión vale valor2 se ejecutan las instrucciones relacionadas con ese valor y así sucesivamente, por tantas opciones como deseemos. Finalmente, para todos los casos no contemplados anteriormente se ejecuta el caso por defecto.

La palabra break es opcional, pero si no la ponemos a partir de que se encuentre coincidencia con un valor se ejecutarán todas las sentencias relacionadas con este y todas las siguientes. Es decir, si en nuestro esquema anterior no hubiese ningún break y la expresión valiese valor1, se ejecutarían las sentencias relacionadas con valor1 y también las relacionadas con valor2, valor3 y default. También es opcional la opción default u opción por defecto.

Supongamos que queremos indicar qué día de la semana es, considerando que tenemos una variable dia_de_la_semana cuyo valor es del 1 al 7 y representa la posición de los días de la semana:

```
switch (dia_de_la_semana) {
    case 1:
       document.write("Es Lunes")
       break
    case 2:
       document.write("Es Martes")
       break
    case 3:
       document.write("Es Miércoles")
       break
    case 4:
       document.write("Es Jueves")
       break
    case 5:
       document.write("Es viernes")
       break
    case 6:
    case 7:
       document.write("Es fin de semana")
       break
    default:
       document.write("Ese día no existe")
}
```

1.4 Instrucciones de control – Bucles

Las estructuras condicionales no son útiles cuando se desea ejecutar de forma repetitiva una instrucción, por ejemplo, si se quiere mostrar un mensaje cinco veces, para esto tenemos que usar las instrucciones de control tipo bucles. Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas inteligentes que pueden tomar decisiones en función del valor de las variables.

- **Estructura for**

  La estructura for permite realizar repeticiones (también llamadas bucles) de una forma muy sencilla. La idea del funcionamiento del for es la siguiente: "mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for.

Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición".

- La inicialización, es la zona en la que se establecen los valores iniciales de las variables que controlan la repetición.
- La condición, es el elemento que decide si continúa o se detiene la repetición.
- La actualización, es el nuevo valor que se asigna después de cada repetición a la variable que controla la repetición.

```
var mensaje = "Hola, estoy dentro de un bucle";

for (var i = 0; i < 5; i++) {
    alert(mensaje);
}
```

La parte de la inicialización del bucle consiste en:

```
var i = 0;
```

Por tanto, en primer lugar, se crea la variable i y se le asigna el valor de 0. Esta zona de inicialización solamente se tiene en consideración justo antes de comenzar a ejecutar el bucle. Las siguientes repeticiones no tienen en cuenta esta parte de inicialización.

La zona de condición del bucle es:

```
i < 5
```

Los bucles se siguen ejecutando, mientras se cumplan las condiciones y se dejan de ejecutar justo después de comprobar que la condición no se cumple. En este caso, mientras la variable i valga menos de 5, el bucle se ejecuta indefinidamente.

Como la variable i se ha inicializado a un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el bucle se repetiría indefinidamente. Por ese motivo, es imprescindible indicar la zona de actualización, en la que se modifica el valor de las variables que controlan el bucle:

```
i++
```

En este caso, el valor de la variable i se incrementa en una unidad después de cada repetición. La zona de actualización se ejecuta después de la ejecución de las instrucciones que incluye el for.

Durante la ejecución de la quinta repetición el valor de i será 4. Después de la quinta ejecución, se actualiza el valor de i, que ahora valdrá 5. Como la condición es que i sea menor que 5, la condición ya no se cumple y las instrucciones del for no se ejecutan una sexta vez.

Normalmente, la variable que controla los bucles for se llama i, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

- **Estructura while**

El propósito de un bucle while es ejecutar una instrucción o bloque de código varias veces, siempre y cuando la expresión sea verdadera. Una vez que la expresión se convierte en falsa, se abandonará el bucle.
El siguiente código:

```
var count = 0;
document.write("Starting Loop" + "<br />");
while (count < 10){
  document.write("Current Count : " + count + "<br />");
  count++;
}
document.write("Loop stopped!");
```

Produce el siguiente resultado:

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

- **Estructura do...while**

El bucle do...while es similar al bucle while, excepto que la verificación de la condición ocurre al final del bucle. Esto significa que el bucle siempre se ejecuta al menos una vez, incluso si la condición es falsa.

El siguiente código:

```
var count = 0;
document.write("Starting Loop" + "<br />");
do{
  document.write("Current Count : " + count + "<br />");
  count++;
}while (count < 0);
document.write("Loop stopped!");
```

Produce el siguiente resultado:

```
Starting Loop
Current Count : 0
Loop stopped!
```

- **Estructura for...in**

  Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos o estructuras tipo arrays.
  Por ejemplo, si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

for (i in dias) {
    alert(dias[i]);
}
```

  La variable que se indica como índice es la que se puede utilizar dentro del bucle for...in para acceder a los elementos del array. De esta forma, en la primera repetición del bucle la variable i vale 0 y en la última, vale 6.

1.5 Creación de objetos

Los objetos en JavaScript, al igual que en muchos otros lenguajes de programación, pueden ser comparados con objetos de la vida real. El concepto de Objetos en JavaScript se puede entender como en la vida real, objetos tangibles.

En JavaScript, un objeto es una entidad independiente con propiedades y tipos. Compárelo con una taza por ejemplo. Una taza es un objeto, con propiedades. Una copa tiene un color, un diseño, peso, un material del que fue hecho, etc. De la misma manera, los objetos de JavaScript pueden tener propiedades, que definen sus características.

- **Objetos y propiedades**

  Un objeto de JavaScript tiene propiedades asociadas. Una propiedad de un objeto puede ser explicada como una variable que se adjunta al objeto. Las propiedades de un objeto son basicamente lo mismo que las variables comunes de JavaScript, excepto por el nexo con el objeto. Las propiedades de un objeto definen las características de un objeto. Tú accedes a las propiedades de un objecto con una simple notación de puntos:

```
1 | nombreObjeto.nombrePropiedad
```

  Como todas las variables de JavaScript, tanto el nombre del objeto (que puede ser una variable normal) y el nombre de propiedad son sensibles a mayúsculas y minúsculas. Puedes definir propiedades asignándoles un

valor. Por ejemplo, vamos a crear el objeto **miAuto** y darle propiedades denominadas **marca**, **modelo**, y **annio** así:

```
var miAuto = new Object();
miAuto.marca = "Ford";
miAuto.modelo = "Mustang";
miAuto.annio = 1969;
```

En JavaScript también se puede acceder o establecer propiedades de objetos mediante la notación de corchetes **[ ]**. Los objetos son llamados a veces *arreglos asociativos*, ya que cada propiedad está asociada con un valor de cadena que puede ser utilizada para acceder a ella. Así, por ejemplo, se puede acceder a las propiedades del objeto **miAuto** de la siguiente manera:

```
miAuto["marca"] = "Ford";
miAuto["modelo"] = "Mustang";
miAuto["annio"] = 1969;
```

El uso de inicializadores de objeto

Además de la creación de objetos utilizando una función constructora, puedes crear objetos utilizando un inicializador de objeto. El uso de los inicializadores de objeto se refiere a veces a cómo crear objetos con la notación literal. "Inicializador de objeto" es consistente con la terminología utilizada por C + +.

La sintaxis para un objeto usando un inicializador de objeto es:

```
var miHonda = {color: "rojo", ruedas: 4, motor: {cilindros: 4, tamanio: 2.2}};
```

- **Definiendo los métodos**

Un **método** es una función asociada a un objeto, o, simplemente, un método es una propiedad de un objeto que es una función. Los métodos se definen normalmente como una función, con excepción de que tienen que ser asignados como la propiedad de un objeto. Ejemplos son:

```
nombreDelObjeto.nombreDelMetodo = nombre_de_la_funcion;

var miObjeto = {
  miMetodo: function(parametros) {
    // ...hacer algo
  }
};
```

Donde **nombreDelObjeto** es un objeto existente, **nombreDelMetodo** es el nombre que se le va a asignar al método, y **nombre_de_la_funcion** es el nombre de la función.

Entonces puede llamar al método en el contexto del objeto de la siguiente manera:

```
object.nombreDelMetodo(parametros);
```

- **Eliminando propiedades**

  Puedes eliminar una propiedad no heredada mediante el operador **delete**. El siguiente código muestra cómo eliminar una propiedad.

```
//Crea un nuevo objeto, miobjeto, con dos propiedades, a y b.
var miobjeto = new Objecto;
miobjeto.a = 5;
miobjeto.b = 12;

//Elimina la propiedad, dejando miobjeto con sólo la propiedad b.
delete myobj.a;
console.log ("a" in myobj) // yields "false"
```

1.6 Document Object Model (DOM)

Cuando se definió el lenguaje XML, surgió la necesidad de procesar y manipular el contenido mediante los lenguajes de programación tradicionales. XML es un lenguaje sencillo de escribir, pero complejo para procesar y manipular de forma eficiente. Por este motivo, surgieron algunas técnicas entre las que se encuentra DOM.

Document Object Model (DOM) es un conjunto de utilidades específicamente diseñadas para manipular documentos XML. Por extensión, DOM también se puede utilizar para manipular documentos XHTML y HTML. Técnicamente, DOM es una API de funciones que se pueden utilizar para manipular las páginas XHTML de forma rápida y eficiente.

Antes de poder utilizar sus funciones, DOM transforma internamente el archivo XML original en una estructura más fácil de manejar, formada por una jerarquía de nodos. De esta forma, DOM transforma el código XML en una serie de nodos interconectados en forma de árbol. Este árbol generado, no solo representa los contenidos del archivo original (mediante los nodos del árbol) sino que también representa sus relaciones (mediante las ramas del árbol que conectan los nodos).

Aunque en ocasiones DOM se asocia con la programación web y con JavaScript, la API de DOM es independiente de cualquier lenguaje de programación. De hecho, DOM está disponible en la mayoría de lenguajes de programación, comúnmente empleados.

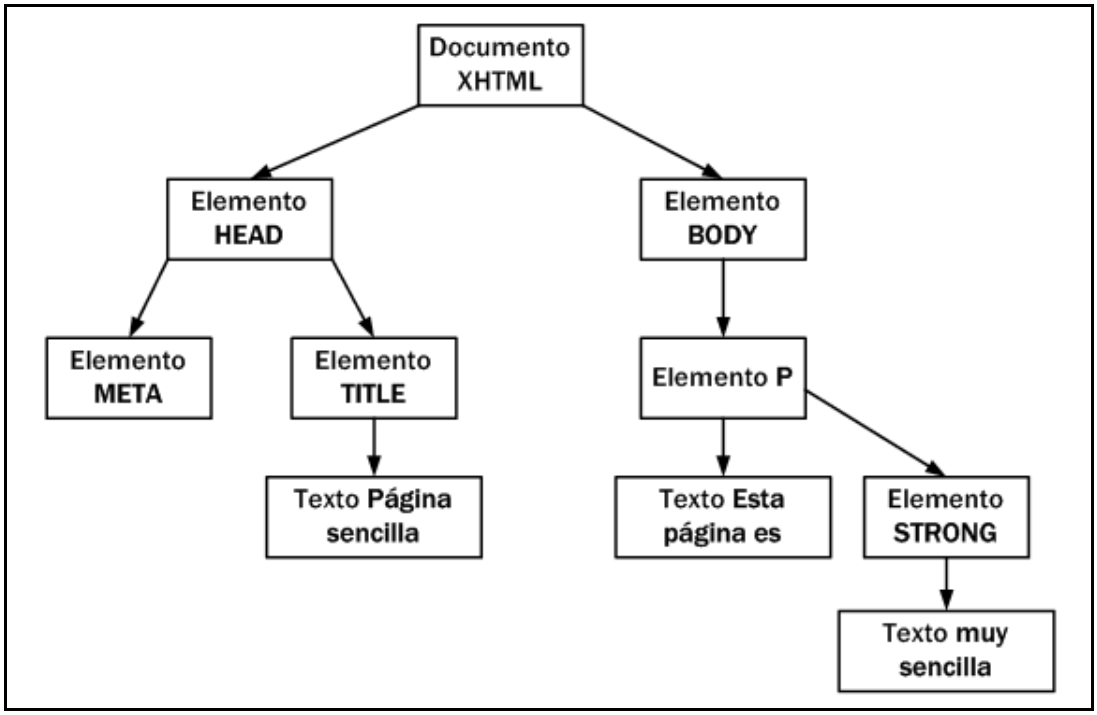
La primera especificación de DOM “DOM Level 1”, se definió en 1998 y permitió homogenizar la implementación del DHTML o HTML dinámico, en los diferentes navegadores, ya que permitía modificar el contenido de las páginas web, sin necesidad de recargar la página entera.

Consideremos la siguiente página HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
    <title>Página sencilla</title>
  </head>

  <body>
    <p>Esta página es <strong>muy sencilla</strong></p>
  </body>
</html>
```

Antes de poder utilizar las funciones de DOM, los navegadores convierten automáticamente la página HTML en una estructura de árbol, con nodos que corresponden a los elementos de la página HTML:



Antes de poder utilizar la API de DOM, se construye de forma automática el árbol. De este modo, para utilizar DOM es imprescindible que la página web se haya cargado por completo, caso contrario no existe el árbol de nodos y las funciones DOM no pueden funcionar correctamente.

La ventaja de emplear DOM es que permite a los programadores disponer de un control muy preciso sobre la estructura del documento HTML o XML que están manipulando. Las funciones que proporciona DOM permiten añadir, eliminar, modificar y reemplazar cualquier nodo de cualquier documento de forma sencilla. Sobre este DOM, javascript realiza su trabajo y nos permite manipular los elementos de las páginas y los eventos que se generen.

1.7 Manejo de eventos

En la programación tradicional, las aplicaciones se ejecutan secuencialmente, de principio a fin, para producir sus resultados. Sin embargo, en la actualidad el modelo predominante es el de la programación basada en eventos. Los scripts y programas esperan, sin realizar ninguna tarea, hasta que se produzca un evento. Una vez producido, ejecutan alguna tarea asociada a la aparición de ese evento, y cuando concluye, el script o programa vuelve al estado de espera.

JavaScript permite realizar scripts con ambos métodos de programación: secuencial y basada en eventos. Los eventos de JavaScript permiten la interacción entre las aplicaciones JavaScript y los usuarios. Cada vez que se pulsa un botón, se produce un evento. Asimismo, cada vez que se pulsa una tecla, también se produce un evento. No obstante, para que se produzca un evento no es obligatorio que intervenga el usuario, ya que por ejemplo, cuando se carga una página, también se produce un evento.

El nivel 1 de DOM, no incluye especificaciones relativas a los eventos JavaScript. El nivel 2, incluye ciertos aspectos relacionados con los eventos. Y el nivel 3, incluye la especificación completa de los eventos de JavaScript.

Desafortunadamente, la especificación de nivel 3 de DOM se publicó en el año 2004, por este motivo, muchas de las propiedades y métodos actuales relacionados con los eventos son incompatibles con los de DOM. De hecho, navegadores como Internet Explorer tratan los eventos, siguiendo su propio modelo incompatible con el estándar.

A continuación, se detalla la clasificación de los eventos:

- **Eventos de mouse**

  Se originan cuando el usuario emplea el mouse para realizar algunas acciones. Son los más empleados en las aplicaciones web.

Evento	Descripción
click	Se produce cuando se pulsa el botón izquierdo del ratón. También se produce cuando el foco de la aplicación está situado en un botón y se pulsa la tecla ENTER.
dblclick	Se produce cuando se pulsa dos veces el botón izquierdo del ratón.
mousedown	Se produce cuando se pulsa cualquier botón del ratón.
mouseout	Se produce cuando el puntero del ratón se encuentra en el interior de un elemento y el usuario, mueve el puntero a un lugar fuera de ese elemento.
mouseover	Se produce cuando el puntero del ratón se encuentra fuera de un elemento y el usuario mueve el puntero hacia un lugar en el interior del elemento.
mouseup	Se produce cuando se suelta cualquier botón del ratón que haya sido pulsado.
mousemove	Se produce (de forma continua) cuando el puntero del ratón se encuentra sobre un elemento.

Todos los elementos de las páginas soportan los eventos de la tabla anterior.

El objeto event contiene las siguientes propiedades para estos eventos:

- Las coordenadas del ratón (todas las coordenadas diferentes relativas a los distintos elementos).
- La propiedad type.
- La propiedad srcElement (Internet Explorer) o target (DOM).
- Las propiedades shiftKey, ctrlKey, altKey y metaKey (solo DOM).
- La propiedad button (solo en los eventos mousedown, mousemove, mouseout, mouseover y mouseup).

Los eventos mouseover y mouseout tienen propiedades adicionales. Internet Explorer define la propiedad fromElement, que hace referencia al elemento desde el que el puntero del ratón se ha movido y toElement que es el elemento al que el puntero del ratón se ha movido. De esta forma, en el evento mouseover, la propiedad toElement es idéntica a n y en el evento mouseout, la propiedad fromElement es idéntica a srcElement.

En los navegadores que soportan el estándar DOM, solamente existe una propiedad denominada relatedTarget. En el evento mouseout, relatedTarget apunta al elemento al que se ha movido el ratón; por su parte, en el evento mouseover, relatedTarget apunta al elemento desde el que se ha movido el puntero del ratón.

Cuando se pulsa un botón del ratón, la secuencia de eventos que se produce es la siguiente: mousedown, mouseup, click. Por tanto, la secuencia de eventos necesaria para llegar al doble click llega a ser tan compleja como la siguiente: mousedown, mouseup, click, mousedown, mouseup, click, dblclick.

- **Eventos de teclado**

  Se originan cuando el usuario pulsa sobre cualquier tecla del teclado.

Evento	Descripción
keydown	Se produce cuando se pulsa cualquier tecla. También se produce de forma continua si se mantiene pulsada la tecla.
keypress	Se produce cuando se pulsa una tecla correspondiente a un caracter alfanumérico (no se tienen en cuenta: SHIFT, ALT, etc.). También se produce de forma continua si se mantiene pulsada la tecla.
keyup	Se produce cuando se suelta cualquier tecla pulsada.

  El objeto event contiene las siguientes propiedades:

  - La propiedad keyCode
  - La propiedad charCode (solo DOM)
  - La propiedad srcElement (Internet Explorer) o target (DOM)
  - Las propiedades shiftKey, ctrlKey, altKey y metaKey (solo DOM)

  Cuando se pulsa una tecla correspondiente a un caracter alfanumérico, se produce la siguiente secuencia de eventos: keydown, keypress y keyup. Cuando se pulsa otro tipo de tecla, se produce la siguiente secuencia de eventos: keydown, keyup. Si se mantiene pulsada la tecla, en el primer caso se repiten de forma continua los eventos keydown y keypress y en el segundo caso, se repite el evento keydown de forma continua.

- **Eventos HTML**

  Se originan cuando se producen cambios en la ventana del navegador o cuando se producen ciertas interacciones entre el cliente y el servidor.

Evento	Descripción
load	Se produce en el objeto window cuando la página se carga por completo. En el elemento <img> cuando se carga por completo la imagen. En el elemento <object> cuando se carga el objeto.
unload	Se produce en el objeto window cuando la página desaparece por completo (al cerrar la ventana del navegador, por ejemplo). En el elemento <object> cuando desaparece el objeto.
abort	Se produce en un elemento <object> cuando el usuario detiene la descarga del elemento antes de que haya terminado.
error	Se produce en el objeto window cuando se produce un error de JavaScript. En el elemento <img> cuando la imagen no se ha podido cargar por completo y en el elemento <object> cuando el elemento no se carga correctamente.
select	Se produce cuando se seleccionan varios caracteres de un cuadro de texto (<input> y <textarea>).
change	Se produce cuando un cuadro de texto (<input> y <textarea>) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento <select>.
submit	Se produce cuando se pulsa sobre un botón de tipo submit (<input type="submit">).
reset	Se produce cuando se pulsa sobre un botón de tipo reset (<input type="reset">).
resize	Se produce en el objeto window cuando se redimensiona la ventana del navegador.
scroll	Se produce en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento <body> contiene la barra de scroll de la página completa.
focus	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento obtiene el foco.
blur	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento pierde el foco.

Uno de los eventos más utilizados es el evento load, ya que todas las manipulaciones que se realizan mediante DOM requieren que la página esté cargada por completo, y por tanto el árbol DOM se haya construido completamente.

El elemento <body> define las propiedades scrollLeft y scrollTop, que se pueden emplear junto con el evento scroll.

1.8 Local y Session Storage

Para el servidor, una petición es indistinta de otra inmediata y el ciclo comienza nuevamente. Es por esto que se dice que el protocolo HTTP no tiene estado, o sea, es stateless. Es nuestra responsabilidad como desarrolladores solventar esta limitación y asegurarnos de mantener el estado entre peticiones.

Para esta tarea, era muy habitual utilizar cookies, estos pequeños pedacitos de información que viajan en cada petición entre el cliente y el servidor. Entonces, si tenemos el problema resuelto, ¿para qué necesitamos otro método nuevo? Bueno, resulta que las cookies no son tan buena solución:

- Aumentan el peso de cada petición al servidor, ya que toda la información guardada en las cookies debe viajar al servidor y volver.
- Tienen una limitación de 4kb de espacio disponible.
- No todo el mundo tiene las cookies habilitadas, sobre todo luego de que se publicaran cómo ciertas empresas utilizan las cookies para registrar nuestro comportamiento en la web.

Por suerte, una de las nuevas específicaciones de la W3C y WHATWG, indica cómo podemos utilizar Javascript para guardar información en los navegadores de nuestros usuarios que expire al finalizar la sesión, o que no expire en lo absoluto a menos que el usuario lo indique.

En HTML5 se tienen dos mecanismos para guardar información en el cliente sin tener que ser enviado en cada momento al servidor, estos son: Local Storage y Session Storage

- **LocalStorage**

  Permite guardar información en el cliente y no tiene un tiempo de expiración.

- **SessionStorage**

  Permite guardar información en el cliente y la información se borra cuando cierra el browser.

  Ambos tienen los mismos métodos:

  - **getItem** (*key*): permite obtener un valor del Storage.
  - **setItem** (*key* , *value*): permite establecer un valor en el Storage.
  - **removeItem** (*key*): permite remover el elemento del Storage.

Ejemplos:

Local Storage

```
if (window.localStorage) {

  localStorage.setItem("nombre", "pepe");

  var nombre = localStorage.getItem("nombre");

  localStorage.removeItem("nombre");
}
else {
  throw new Error('Tu Browser no soporta LocalStorage!');
}
```

Session Storage

```
// Almacena la información en sessionStorage
sessionStorage.setItem('key', 'value');

// Obtiene la información almacenada desde sessionStorage
var data = sessionStorage.getItem('key');
```

¿Cuánto podemos almacenar?

Ddepende mucho del navegador, pero la cantidad oscila entre 2.5 y 5 Mb (2.5 Mb en la mayoría de los navegadores).
Pueden ver la tabla completa y probar hasta cuanto se puede guardar en el navegador en el que están con Web Storage Test

¿Cómo guardar JSON?

Primero se tiene que serializar (convertirlo en cadena) el objeto con **JSON.stringify** y para deserializarlo (convertirlo nuevamente a objeto) se debe usar **JSON.parse**

```
var personaAGuardar = JSON.stringify(persona);
localStorage.setItem("persona", personaAGuardar);
var personaGuardada = localStorage.getItem("persona");
console.log(typeof persona); //object
console.log(typeof personaGuardada); //string
var personaGuardada = JSON.parse(personaGuardada);
console.log(personaGuardada.locura); //true
```

.

2. jQuery

jQuery

Es un javascript library que permite simplificar la manera de interactuar con los documentos HTML, manipular el DOM, manejar eventos, desarrollar animaciones y agregar interacciones con AJAX en aplicaciones web.

http://jquery.com/

---

jQuery es una biblioteca o framework de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC.

Además, jQuery es libre y de código abierto. Posee un doble licenciamiento bajo la licencia MIT y la licencia pública general de GNU v2, permitiendo su uso en proyectos libres y privativos. jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript, que de otra manera, requerirían de mucho código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y esfuerzo.
Microsoft incluye la biblioteca en su IDE Visual Studio, y la usa en las aplicaciones ASP.NET.

Para mayor información, visitar el siguiente enlace: http://jquery.com/

Es importante identificar todas las características y funcionalidades que brinda jQuery, y al mismo tiempo, poder agruparlas según el tipo de funcionalidad. Para ello, existen sitios web con el resumen general de toda la librería. Por ejemplo, una muy completa se puede encontrar en el siguiente enlace: http://jqapi.ru/

2.1 Selectores jQuery

Los selectores son un mecanismo para seleccionar determinados elementos de una página o documento HTML. jQuery utiliza los selectores para acceder de una manera rápida y sencilla a un elemento o grupo de elementos del DOM y luego poder aplicarle cualquier tipo de instrucción, evento, animación, etc.
También tenemos a los filtros, que suelen acompañar a los selectores para realizar selecciones más específicas.

2.1.1 Selectores

- **Selectores Básicos**

```
// Selecciona un elemento por ID.
// Selecciona un elemento: <span id="footer"><span>
$('#footer');

// Selecciona todos los elementos por nombre.
// Selecciona elementos: <table>
$('table');

// Selecciona todos los elementos por CLASS.
// Selecciona los elementos: <span class="box" />
$('.box');

// Selecciona todos los elementos por las clases especificadas.
// Selecciona los elementos:<span class="box error"/>
$('.box.error');

// Selecciona todos los elementos.
// Selecciona todos los elementos.
$('*');

// Selecciona los elementos usando varios selectores.
// Cada selector selector es separado por coma.
$('#header, div, .footer');
```

- **Selectores Jerárquicos:**

```
// Selecciona todos los elementos descendiente de otro elemento
// especificado.
// Selecciona los IMG descendiente de un DIV.
$('div img');

// Selecciona los elementos descendientes directos de otros.
// Esto quiere decir que solo se seleccionarán los primeros
// dentro de otros.
// Selecciona los IMG descendientes directos de un DIV.
$('div > img');

// Selecciona los elementos que le siguen a otro elemento
// especificado.
// Selecciona todos los IMG después de un DIV.
$('div + img');

// Selecciona todos los elementos primos que le siguen a otro
// elemento especificado.
// Selecciona todos los IMG primos después de un DIV.
$('div ~ img');
```

- **Selectores de elementos de formulario:**

```
$(':input');// Selecciona todos los elementos INPUT, TEXTAREA, SELECT
// y BUTTON.

$(':text');// Selecciona todos los elementos INPUT de tipo "text".

$(':password');// Selecciona todos los elementos INPUT de tipo "password".

$(':radio');// Selecciona todos los elementos INPUT de tipo "radio".

$(':checkbox');// Selecciona todos los elementos INPUT de tipo "checkbox".

$(':password');// Selecciona todos los elementos INPUT de tipo "submit".

$(':image');// Selecciona todos los elementos INPUT de tipo "image".

$(':reset');// Selecciona todos los elementos INPUT de tipo "reset".

$(':button');// Selecciona todos los elementos INPUT de tipo "button".

$(':file');// Selecciona todos los elementos INPUT de tipo "file".

$(':hidden');// Selecciona todos los elementos INPUT de tipo "hidden".
```

2.1.2 Filtros

- **Filtros Básicos**

```
$(':first'); // Selecciona el primer elemento.
$('div:first'); // Selecciona el primer DIV.

$(':last'); // Selecciona el último elemento.
$('div:last'); // Selecciona el último DIV.

$(':not(X)'); // Filtra los elementos usando un selector X.
$('div:not(#box)'); // Selecciona los DIV que no tenga un id="box".

$(':even'); // Selecciona los elementos pares.
$('div:even'); // Selecciona los DIV pares.

$(':odd'); // Selecciona los elementos impares.
$('div:odd'); // Selecciona los DIV impares.

$(':eq(X)'); // Selecciona un elemento de índice X.
$('div:eq(4)'); // Selecciona el quinto índice, contado de 0 a 4.

$(':gt(X)'); // Selecciona los elementos con índice mayor a X.
$('div:gt(2)'); // Selecciona todos los DIV después del tercero.

$(':lt(X)'); // Selecciona los elementos con índice menor a X.
$('div:lt(2)'); // Selecciona todos los DIV antes del tercero.

$(':header'); // Selecciona los elementos: h1, h2, h3 etc...

$(':animated'); // Selecciona todos los elementos animados
```

- **Filtros de Contenido**

```
// Selecciona los elementos con el texto especificado
$(':contains(texto)');
$('div:contains(texto)');

// Selecciona los elementos que no tienen ni texto ni elementos.
$(':empty');
$('div:empty');

// Selecciona los elementos que tenga por lo menos un elemento
// especificado por un selector.
$(':has(selector)');
// Selecciona todos los DIV que tenga un IMG de clase 'box'.
$('div:has(img.box)');

// Selecciona los elementos que tienen texto o elementos.
$(':parent');
```

- **Filtros de Visibilidad**

```
$(':hidden'); // Selecciona los elementos no visibles.
$('div:hidden'); // Selecciona los DIV no visibles.

$(':visible'); // Selecciona los elementos visibles.
$('div:visible'); // Selecciona los DIV visibles.
```

- **Filtros por Atributos**

```
$('[href]');// Selecciona los elementos con el atributo HREF.

$('[href=blank.html]');// Selecciona los elementos con el atributo HREF="blank.html".

$('[href!=blank.html]');// Selecciona los elementos con el atributo HREF que no sea igual a "blank.html".

$('[href^=blan]');// Selecciona los elementos con el atributo HREF que comience con "blan".

// Selecciona los elementos con el atributo HREF que
// termine en ".html".
$('[href$=.html]');

// Selecciona los elementos con el atributo HREF que
// tenga en cualquier parte el valor "k.h".
$('[href*=k.h]');

// Selecciona los elementos con el atributo HREF que
// comience por "bla" y termine en ".html".
$('[href^=bla][href$=.html]');
```

- **Filtros Descendientes**

```
// Selecciona los descendientes de un elemento por índice, pares,
// impares o ecuación.
$(':nth-child(índice/par/impar/ecuación)');
// Selecciona todos los IMG descendientes pares de un DIV.
$(':nth-child(even)');
// Selecciona todos los IMG descendientes de tres en tres
// con un desfase de 1 de un DIV.
$(':nth-child(3n + 1)');

// Selecciona los primeros elementos descendientes de un
// elemento especificado.
$(':first-child');

// Selecciona los últimos elementos descendientes de un
// elemento especificado.
$(':last-child');

// Selecciona los elementos descendientes de un
// elemento especificado.
$(':only-child');
```

- **Filtros de Formularios**
-

```
// Selecciona los elementos de formularios que están activos.
$(':enabled');

// Selecciona los elementos de formularios que no están activos.
$(':disabled');

// Selecciona los CHECKBOXs que están chequeados.
$(':checked');

// Selecciona los OPTIONs seleccionado de un SELECT.
$(':selected');
```

2.2 Eventos

Los eventos son un tipo de métodos en jQuery que se usan para saber cuándo el usuario interactúa con el navegador.

2.2.1 Eventos producidos por el mouse

- click(): sirve para generar un evento cuando se produce un clic en un elemento de la página.
- dblclick(): sirve para generar un evento cuando se produce un doble clic sobre un elemento.
- hover(): esta función sirve para manejar dos eventos: cuando el mouse entra y sale de encima de un elemento. Por tanto, espera recibir dos funciones en vez de una que se envía a la mayoría de los eventos.

- mousedown(): sirve para generar un evento cuando el usuario hace clic, en el momento que presiona el botón e independientemente, de si lo suelta o no. Sirve tanto para el botón derecho como el izquierdo del mouse.
- mouseup(): sirve para generar un evento cuando el usuario ha hecho clic y luego suelta un botón del ratón. Este evento se produce en el momento de soltar el botón.
- mouseenter(): este evento se produce al situar el mouse, encima de un elemento de la página.
- mouseleave(): este evento se desata cuando el mouse sale de encima de un elemento de la página.
- mousemove(): se produce al mover el mouse sobre un elemento de la página.
- mouseout(): cumple la misma función que el evento mouseout de JavaScript. Se desata cuando el usuario sale con el mouse de la superficie de un elemento.
- mouseover(): cumple la misma función que el evento mouseover de Javascript.
  Se produce cuando el mouse está sobre un elemento, pero tiene como particularidad que puede producirse varias veces, mientras se mueve el mouse sobre el elemento, sin necesidad de haber salido.
- toggle(): sirve para indicar dos o más funciones, para ejecutar cosas cuando el usuario realiza clics; con la particularidad, que esas funciones se van alternando a medida que el usuario hace clics.

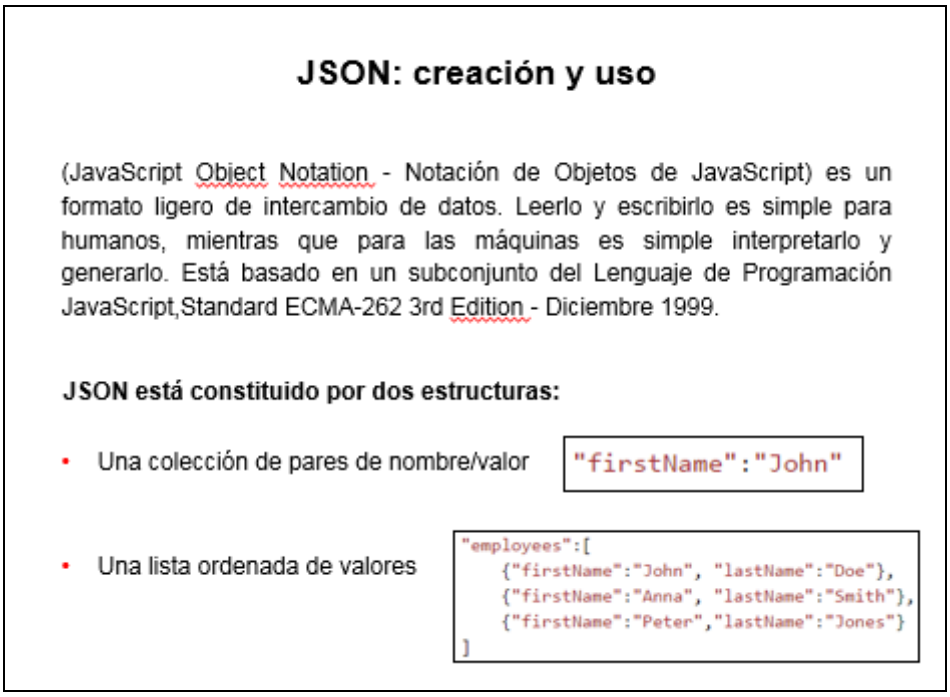
2.2.2 Eventos producidos por el teclado

- keydown(): este evento se produce en el momento que se presiona una tecla del teclado, independientemente de si se libera la presión o se mantiene. Se produce una única vez en el momento exacto de la presión.
- keypress(): ocurre cuando se digita un caracter o se presiona otro tipo de tecla. Es como el evento keypress de Javascript, por lo que se entiende que keypress() se ejecuta una vez, como respuesta a una pulsación e inmediata liberación de la tecla, o varias veces si se pulsa una tecla y se mantiene pulsada.
- keyup(): se ejecuta en el momento de liberar una tecla, es decir, al dejar de presionar una tecla que teníamos pulsada.

2.2.3 Eventos producidos por el teclado o mouse

- focusin(): se produce cuando el elemento gana el foco de la aplicación, que puede producirse al hacer clic sobre un elemento, o al presionar el tabulador y situar el foco en dicho elemento.
- focusout(): ocurre cuando el elemento pierde el foco de la aplicación, que puede ocurrir cuando el foco está en ese elemento y se pulsa el tabulador, o al mover a otro elemento con el mouse.
- focus(): sirve para definir acciones cuando se produce el evento focus de Javascript, cuando el elemento gana el foco de la aplicación.

3. JSON: creación y uso



JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript,Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituído por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.

- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

En JSON, se presentan de estas formas:

- Un *objeto* es un conjunto desordenado de pares nombre/valor. Un objeto comienza con **{** (llave de apertura) y termine con **}** (llave de cierre). Cada nombre es seguido por **:** (dos puntos) y los pares nombre/valor están separados por **,** (coma).

3.1 Nombre y Valor

JSON data es escrito como un par de nombre/valor.
Un par de nombre/valor consiste en el nombre del campo (entre doble comillas), seguido por una coma y el valor:

```
"firstName":"John"
```

3.2 JSON Objects

JSON object debe de escribirse entre llaves ({}).

```
{"firstName":"John", "lastName":"Doe"}
```

3.3 JSON Arrays

JSON array debe ser escrito entre corchetes ([]).

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName":"Jones"}
]
```

3.4 Uso de JSON

```
var employees = [
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter","lastName": "Jones"}
];
```

La primera entrada en el array puede **ser accessible** de la siguiente manera:

```
// returns John Doe
employees[0].firstName + " " + employees[0].lastName;
```

La primera entrada en el array puede **ser modificado** de la siguiente manera:

```
employees[0].firstName = "Gilbert";
```

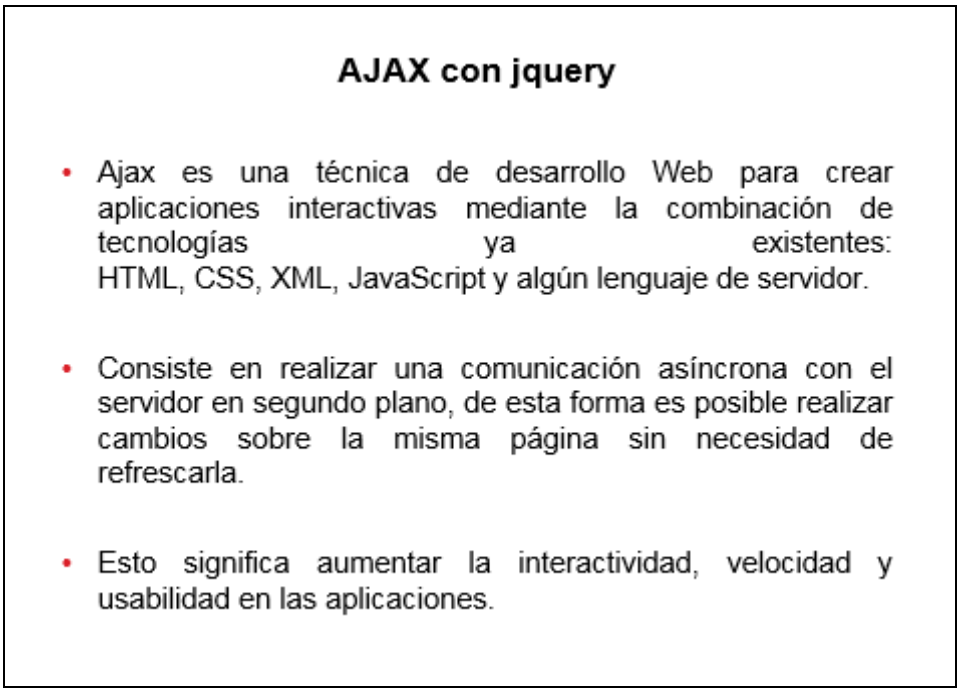
Si se necesita agregar un objeto al array se tiene que hacer lo siguiente:

```
employees.push({
    "firstName":"Pedro",
    "lastName":"Diaz"
    });
```

Ejemplo de un objeto más complejo en JSON:

```
var persona = {
    'nombre': 'Cesar',
    'edad': 25,
    'estudios': ['primaria', 'secundaria','universidad']
};
```

4. AJAX con jquery



Ajax (Asynchronous JavaScript and XML) es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios, mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma, es posible realizar cambios sobre las páginas sin necesidad de refrescarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. JavaScript es el lenguaje interpretado en el que normalmente se efectúan las funciones de llamada de Ajax, mientras que el acceso a los datos se realiza mediante el objeto XMLHttpRequest, disponible en los navegadores actuales.

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como JavaScript y Document Object Model (DOM).

Cuando se usa Ajax, no es necesario que el contenido asíncrono esté formateado en XML, hay un formato más ligero, y el más usado, que se llama JSON (Java Script Object Notation).

A continuación, se detallan las formas en que se pueden realizar peticiones Ajax con jQuery:

1. $.get()

Se trata de una función sencilla con la que se puede lanzar peticiones GET al servidor mediante Ajax.

```
//Utilizando $.get():
$.get("/Demo/Mantenimiento", { nombre: "Cesar", edad: "25" },
  function (data) {
      alert("Información Cargada: " + data);
  });
```

Mediante el paso de 3 opciones, se puede lanzar la petición al url (1º) con los parámetros (2º) y tratar la respuesta mediante un callback (3º).

2. $.post()

Similar al anterior, pero en este caso permite enviar peticiones POST:

```
//Utilizando $.post():
$.post("/Demo/Mantenimiento", { nombre: "Juan", edad: "18" },
  function (data) {
      alert("Información Cargada: " + data);
  });
```

3. $.getJSON()

Aunque los anteriores tienen la posibilidad de especificar el tipo de retorno, la opción más cómoda es la de usar este método que permite obtener la respuesta JSON evaluada en la función callback.:

```
//Utilizando $.getJSON():
$.getJSON('/Demo/ProfesoresListar', { busqueda: filtro }, function (dataLista) {

    alert(dataLista[1].nombre);
});
```