Diseño Web

Prof. Netz Romero

JavaScript

JavaScript es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web. El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar interactividades, de modo que el mayor recurso, y tal vez el único, con que cuenta este lenguaje es el propio navegador.

JavaScript es el siguiente paso, después del HTML, que puede dar un programador de la web que decida mejorar sus páginas y la potencia de sus proyectos. Es un lenguaje de programación bastante sencillo y pensado para hacer las cosas con rapidez, a veces con ligereza.

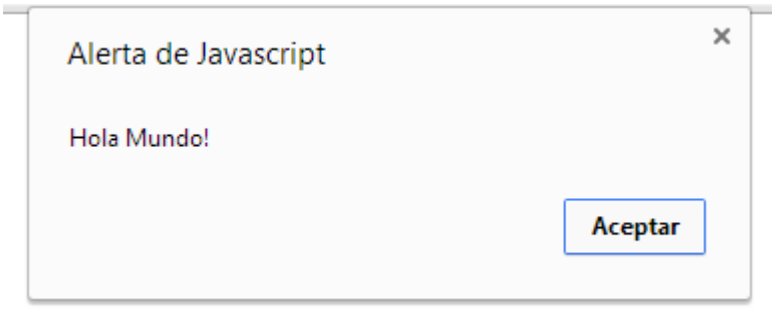
Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A continuación, se muestra un primer script sencillo pero completo:

```
<!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>El primer script</title>

<script type="text/javascript">
  alert("Hola Mundo!");
</script>
</head>

<body>
<p>Esta página contiene el primer script</p>
</body>
</html>
```



La instrucción `alert()` es una de las utilidades que incluye JavaScript y permite mostrar un mensaje en la pantalla del usuario. Si se visualiza la página web de este primer script en cualquier navegador, automáticamente se mostrará una ventana con el mensaje "`Hola Mundo!`".

Programación básica

Variables

Una variable es un espacio en memoria donde se almacena un dato, un espacio donde podemos guardar cualquier tipo de información que necesitemos para realizar las acciones de nuestros programas. También una variable se emplea para hacer referencia a otro valor.

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). Aparte de esta, hay una serie de reglas adicionales para construir nombres para variables. La más importante es que tienen que comenzar por un carácter alfabético o el subrayado. No podemos utilizar caracteres raros como el signo `+`, un espacio o un `$`. Nombres admitidos para las variables podrían ser

```
Edad
paisDeNacimiento
_nombre
```

Las variables en JavaScript se crean mediante la palabra reservada `var`. De esta forma, el ejemplo anterior se puede realizar en JavaScript de la siguiente manera:

```
var res
var operando
```

También se puede asignar un valor a la variable cuando se está declarando

```
var x = 5;
var y = 6;
var z = x + y;
```

También se permite declarar varias variables en la misma línea, siempre que se separen por comas.

```
var precio, total
```

Tipos de variables

Los tipos de datos JavaScript se dividen en dos grupos: tipos primitivos y tipos objeto.

Los tipos primitivos incluyen: cadenas de texto (String), variables booleanas cuyo valor puede ser true o false (Boolean) y números (Number). Además hay dos tipos primitivos especiales que son Null y Undefined.

Los tipos objeto son datos interrelacionados, no ordenados, donde existe un nombre de objeto y un conjunto de propiedades que tienen un valor. Un objeto puede ser creado específicamente por el programador. No obstante, se dice que todo aquello que no es un tipo primitivo es un objeto.

		NOMBRE	DESCRIPCIÓN
TIPOS DE DATOS EN JAVASCRIPT	TIPOS PRIMITIVOS	String	Cadenas de texto
		Number	Valores numéricos
		Boolean	true ó false
		Null	Tipo especial, contiene null
		Undefined	Tipo especial, contiene undefined
	TIPOS OBJETO	Tipos predefinidos de JavaScript	Date (fechas) RegExp (expresiones regulares) Error (datos de error)
		Tipos definidos por el programador / usuario	Funciones simples Clases
		Arrays	Serie de elementos o formación tipo vector o matriz. Lo consideraremos un objeto especial que carece de métodos.
		Objetos especiales	Objeto global
			Objeto prototipo
			Otros

Variables numéricas

Se utilizan para almacenar valores numéricos enteros (llamados integer en inglés) o decimales (llamados float en inglés), ejemplos:

```
var suma = 16;          // variable tipo entero
var total = 234.65;     // variable tipo decimal
```

Variables de 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 javaScript";
var nombreProducto = 'Producto ABC';
```

Toma en cuenta las siguientes expresiones para los caracteres especiales:

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

Variables booleanas

Una variable de tipo boolean almacena un tipo especial de valor que solamente puede tomar dos valores: `true` (verdadero) o `false` (falso). A continuación se muestra un par de variables de tipo booleano:

```
var tienesNovia = false;

var debesTarjeta = true;
```

La palabra reservada null representa a una variable que contiene una propiedad especial que se utiliza para indicar "ningún valor". Si el valor de una variable es `null`, la variable no contiene ningún valor de tipo objeto o primitivo. Ejemplo:

```
var numero1 = null;

var numero2;
```

En vista de que la variable "`numero2`" no tiene un valor asignado, ¿de qué tipo es? Dado que todas las variables han de tener siempre un tipo, en este caso Javascript considera la variable "`numero2`" de tipo undefined.

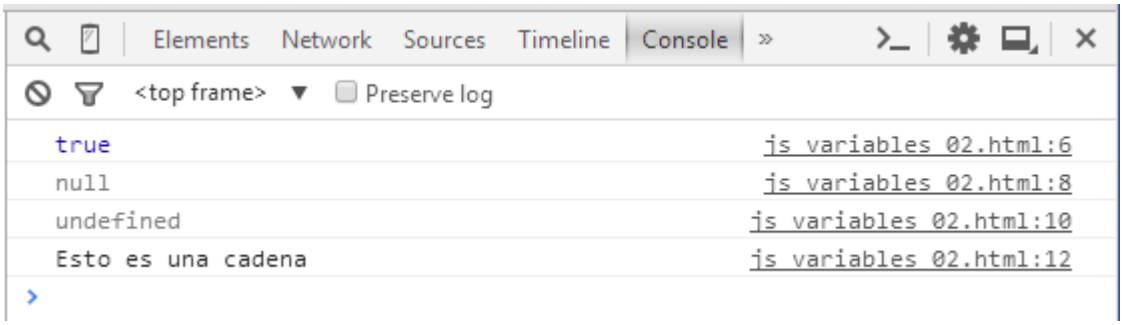
Un ejemplo de variables en JavaScript:

```
<html>
<head><title>Variables 1 en JavaScript</title></head>
<body>
<script type="text/javascript">
var entero = 10;
alert("La variable entero vale: " + entero);
var flotante = 123.456;
alert("La variable flotante vale: " + flotante);
var precio = 22.55;
precio = precio + 10;
alert("La variable precio vale: " + precio);
var cadena = "Esto es una cadena";
alert("La variable cadena vale: " + cadena);
</script>
</body>
</html>
```

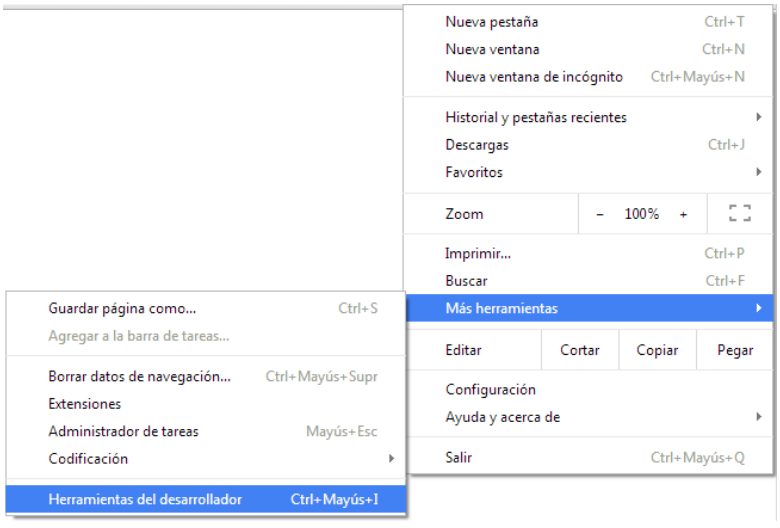
Un ejemplo de variables en JavaScript:

```
<html>
<head><title>Variables 2 en JavaScript</title></head>
<body>
<script type="text/javascript">
var booleano = true;
console.log(booleano);
var cad = null;
console.log(cad);
var precio;
console.log(precio);
var cadena = "Esto es una cadena";
console.log(cadena);
</script>
</body>
</html>
```

`Console` es el objeto JavaScript que representa a la consola y con el método JavaScript `console.log` podemos volcar el contenido sobre la consola.



Para encontrar la consola en Google Chrome:



Arrays

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. La declaración de un array se hace de misma forma que se declara cualquier variable:

```
var nombreDelArray;
```

La variable anterior podría considerarse como un array vacío, el cual se puede inicializar de la siguiente forma:

```
nombreDelArray = ['Uno', 'Dos', 'Tres', 'Cuatro', 'Cinco'];
```

También el array adquiere condición de tal cuando la variable se inicializa con forma de array:

```
var nombreDelArray = ['Uno', 'Dos', 'Tres', 'Cuatro', 'Cinco'];
```

Una vez definido un array, es muy sencillo acceder a cada uno de sus elementos. Cada elemento se accede indicando su posición dentro del array. Y las posiciones de los elementos empiezan a contarse en el 0.

```
var numero = nombreDelArray [0];     // numero = "Uno"

var otroNumero = dias[4];           // otroNumero = "Cinco"
```

También se puede crear un array utilizando un objeto Javascript ya implementado en el navegador. En este caso el objeto global de JavaScript Array es un constructor para matrices unidimensionales (también llamadas vectores o arreglos), que son objetos tipo lista de alto nivel. Esta es la sentencia para crear un objeto array:

```
var miArray = new Array()
```

Esto crea un array en la página que esta ejecutándose. El array se crea sin ningún contenido, es decir, no tendrá ninguna casilla o compartimiento creado. También podemos crear el array Javascript especificando el número de compartimentos que va a tener.

```
var miArray = new Array(10)
```

En este caso indicamos que el array va a tener 10 posiciones, es decir, 10 casillas donde guardar datos. En las casillas de los arrays podemos guardar datos de cualquier tipo. Podemos ver un array donde introducimos datos de tipo carácter.

```
miArray[0] = "Hola"

miArray[1] = "a"

miArray[2] = "todos"
```

Incluso, en Javascript podemos guardar distintos tipos de datos en las casillas de un mismo array. Es decir, podemos introducir números en unas casillas, textos en otras, boleanos o cualquier otra cosa que deseemos.

```
miArray[0] = "desarrolloweb.com"

miArray[1] = 1275

miArray[1] = 0.78

miArray[2] = true
```

Operadores

Los operadores permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar diferentes variables.

Los operadores se dividen en:

1. operadores de asignación;
2. operadores aritméticos;
3. operadores relacionales;
4. operadores lógicos;
5. operadores cadenas.
6. operadores especiales.

Operadores de Asignación

Los operadores de asignación permiten asignar un valor a una variable, los cuales son:

Operador	Descripción
=	Asigna el valor del operando de la derecha a la variable de la izquierda. Ejemplo: inttotal=100;
+= (tambien - =, * =, / =)	Añade el valor del operando de la derecha a la variable de la izquierda. Ejemplo: inttotal +=100
&= (también |=)	Asigna el resultado de (operando de la izquierda & operando de la derecha) al operando de la izquierda

```
<html>
<head><title>Operadores 1</title></head>
<body>
<script language="JavaScript">
   var a = 8;
   document.write(a);document.write("<br>");
   a += 3;
   document.write(a);document.write("<br>");
   a -= 2;
   document.write(a);document.write("<br>");
   var b = 3;
   b *= 2;
   document.write(b);document.write("<br>");
   var c = true;
   c &= false;
   document.write(c);
</script>
</body>
</html>
```

Operadores de Aritméticos

JavaScript permite realizar manipulaciones matemáticas sobre el valor de las variables numéricas y los operadores aritméticos normales son:

Operador	Nombre	Ejemplo	Descripción
+	Suma	5 + 6	Suma dos números
-	Substracción	7 - 9	Resta dos números
*	Multiplicación	6 * 3	Multiplica dos números
/	División	4 / 8	Divide dos números
%	Módulo: el resto después de la división	7 % 2	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	a++	Suma 1 al contenido de una variable.
--	Decremento.	a--	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	-a	Invierte el signo de un operando.

```
<html>
<head><title>Operadores 2</title></head>
<body>
<script language="JavaScript">
   var a = 8;
   var b = 3;
   document.write(a + b);
   document.write("<br>");
   document.write(a - b);
   document.write("<br>");
   document.write( a * b);
   document.write("<br>");
   document.write(a / b);
   document.write("<br>");
   a++;
   document.write(a);
   document.write("<br>");
   b--;
   document.write(b);
</script>
</body>
</html>
```

El operador de división aplicado a variables de tipo entero produce un resultado truncado en la parte decimal. Si se divide el módulo entre cero, se tendrá una excepción en la ejecución: si la operación excede del límite inferior (underflow), el resultado será cero; si excede del límite superior (overflow), se obtendrá una aproximación.

Operadores de Relación

Los operadores relacionales se llegan a utilizar en las expresiones condicionales para tomas de decisiones. Se basan en el concepto de verdadero o falso por lo que devuelve un valor lógico basado en si la comparación es verdad o no. Los operando pueden ser numéricos o cadenas.

Operador	Descripción
==	" Igual a" devuelve true si los operandos son iguales
===	Estrictamente "igual a" (JavaScript 1.3)
!=	" No igual a" devuelve true si los operandos no son iguales
!==	Estrictamente " No igual a" (JavaScript 1.3)
>	" Mayor que" devuelve true si el operador de la izquierda es mayor que el de la derecha.
>=	" Mayor o igual que " devuelve true si el operador de la izquierda es mayor o igual que el de la derecha.
<	" Menor que" devuelve true si el operador de la izquierda es menor que el de la derecha.
<=	"Menor o igual que" devuelve true si el operador de la izquierda es menor o igual que el de la derecha.

```
<html>
<head><title>Operadores 3</title></head>
<body>
<script language="JavaScript">
   a = 8;
   b = 3;
   c = 3;
   document.write(a == b);document.write("<br>");
   document.write(a != b);document.write("<br>");
   document.write(a < b);document.write("<br>");
   document.write(a > b);document.write("<br>");
   document.write(a >= c);document.write("<br>");
   document.write(b <= c);document.write("<br><br>");
   document.write(3 == "3");document.write("<br>");
   document.write(3 === "3");document.write("<br>");
</script>
</body>
</html>
```

JavaScript 1.2 no realizaba 'conversiones de tipo', por eso si dos operadores eran de tipos distintos no se realizaba la comparación. Finalmente, en las últimas versiones de JavaScript se añaden los operadores de 'comparación estricta', los cuales realizarán la comparación si los dos operandos son del mismo tipo.

Operadores Lógicos

Los operadores lógicos son muy parecidos a los relacionales, en el sentido de que dan también como output sólo dos valores que, en este caso, son: 0 si la expresión lógica es verdadera, 1 si es falsa.

Operador	Descripción
&&	" Y " Devuelve true si ambos operadores son true.
||	" O " Devuelve true si uno de los operadores es true.
!	"No" Devuelve true si la negación del operando es true.

```
<html>
<head><title>Operadores 3</title></head>
<body>
<script language="JavaScript">
   a = 8;
   b = 3;
   c = 3;
   document.write( (a == b) && (c > b) );document.write("<br>");
   document.write( (a == b) || (b == c) );document.write("<br>");
   document.write( !(b <= c) );document.write("<br>");
</script>
</body>
</html>
```

Operadores de Cadena

Los valores cadena pueden compararse usando los operadores de comparación. Adicionalmente, usted puede concatenar cadenas usando el operador +.

```
<html>
<head><title>Operadores 3</title></head>
<body>
<script language="JavaScript">
   Nombre = "Jose"
   document.write( "Hola " + Nombre + "." );
</script>
</body>
</html>
```

Operadores Especiales

Varios operadores de JavaScript, es difícil clasificarlos en una categoría en particular. Estos operadores se resumen a continuación.

Operador	Descripción
(condición) ? trueVal : falseVal	Asigna un valor especificado a una variable si la condición es true, por otra parte asigna un valor alternativo si la condición es false.
New	El operador new crea una instancia de un objeto.
This	La palabra clave 'this' se refiere al objeto actual.
,	El operador ',' evalúa los dos operados.
Delete	El operador delete borra un objeto, una propiedad de un objeto, o un elemento especificado de un vector.
Void	El operador Void especifica una expresión que será evaluada sin devolver ningún valor.
Typeof	Devuelve el tipo de dato de un operando.

Estructuras de control

Las estructuras de control en javascript y en la mayoría de los lenguajes de programación se utilizan en los para definir el flujo de instrucciones que se van ejecutando. A continuación veremos la sintaxis de las mismas.

Estructura if

Se emplea para tomar decisiones en función de una condición. Su sintáxis es:

```
if (expresión) {

   ...
}
```

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

```
<html>
<head><title>Operadores 3</title></head>
<body>
<script language="JavaScript">
var mostrarMensaje = true;
 if(mostrarMensaje) {
   document.write("Condicion verdadera");
}
</script>
</body>
</html>
```

Otro ejemplo:

```
<html>
<head><title>If</title></head>
<body>
<h3>If</h3>
Dame un valor mayor que 10<br>
<textarea id="myTextarea" rows="1" cols="10"></textarea>
<br>
<button type="button" onclick="miFuncion()">OK</button>
<script>
function miFuncion() {
  var x = document.getElementById("myTextarea").value;
  if(x > 10) {
    document.write( "Valor mayor que 10");
  }
}
</script>
</body>
</html>
```

Estructura if - else

Ahora se toman dos tipos de decisiones, "si se cumple la condición, hazlo; si no se cumple la condición, haz esto otro". Su definición formal es la siguiente:

```
if(condicion) {
 ...
} 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 (es decir, si su valor es `false`) se ejecutan todas las instrucciones contenidas en `else { }`. Ejemplo:

```
<html>
<head><title>If else</title></head>
<body>
<script language="JavaScript">
var edad = 5.9;
if(edad >= 6) {
  alert("Aprobado");
} else {
  alert("Reprobado");
} </script>
</body>
</html>
```

Otro ejemplo:

```
<html>
<head><title>If else</title></head>
<body>
<h3>If</h3>
Dame un valor mayor que 10<br>
<textarea id="myTextarea" rows="1" cols="10"></textarea>
<br>
<button type="button" onclick="miFuncion()">OK</button>
<script>
function miFuncion() {
  var x = document.getElementById("myTextarea").value;
  if(x > 10) {
    document.write( "Valor mayor que 10");
  } else {
    document.write( "Valor menor o igual que 10");
  }
}
</script>
</body>
</html>
```

Estructura switch

La estructura `switch` es muy útil cuando la condición que evaluamos puede tomar muchos valores, es el caso típico de los menús de elección de opciones. En función de la opción elegida por el usuario nosotros debemos hacer lo que nos pide. La sintaxis es:

```
switch (expresion) {
 case valor1: sentencia1;
      break;
 case valor2: sentencia2;
      break;
 ...
 case valorN: sentenciaN;
      break;
 default: sentenciaFinal;
      break;
}
```

Ejemplo:

```
<html>
<head><title>Switch</title></head>
<body>
<script language="JavaScript">
var dia = 8;
switch(dia) {
  case 1: console.log("Hoy es lunes."); break;
  case 2: console.log("Hoy es martes."); break;
  case 3: console.log("Hoy es miércoles."); break;
  case 4: console.log("Hoy es jueves."); break;
  case 5: console.log("Hoy es viernes."); break;
  case 6: console.log("Hoy es sábado."); break;
  case 7: console.log("Hoy es domingo."); break;
  default: console.log("Dia incorrecto."); break;
} </script>
</body>
</html>
```

Estructura while

La estructura `while` ejecuta un simple bucle, mientras se cumpla la condición. Su definición formal es la siguiente:

```
while (condición){

    //sentencias a ejecutar

}
```

Ejemplo:

```
<html>
<head><title>While</title></head>
<body>
<script language="JavaScript">
 var nb = prompt("Introduzca un número", 15);  // teclear un número
 var fact = 1;      // valor factorial inicial
 var i = nb;
 while (i > 1) {   //bucle mientras sea mayor que la unidad
   fact*=i;        // factorial = factorial * i
   i-=1;           // i = i - 1
 }
 alert ("Factorial de " + nb + " = " + fact);  // mostrar resultado
</script>
</body>
</html>
```

Una característica a tener muy en cuenta a la hora de decidirse a utilizar este tipo de bucles es que la condición es lo primero que se evalúa**.**

Estructura do - while

Este bucle es exactamente igual que el anterior pero con la diferencia de que la condición se comprueba al final.

La sintaxis es la siguiente.

```
do {
    //sentencias del bucle
} while (condición);
```

El bucle se ejecuta siempre una vez y al final se evalúa la condición para decir si se ejecuta otra vez el bucle o se termina su ejecución. Veamos el ejemplo que escribimos para un bucle `while`:

```
<html>
<head><title>Do while</title></head>
<body>
<script language="JavaScript">
 var color
 do {
    color = prompt("dame un color (escribe rojo para salir)","")
 } while (color != "rojo");
</script>
</body>
</html>
```

Estructura for

El bucle FOR se utiliza preferentemente cuando deseamos repetir una o más instrucciones un determinado número de veces. La sintaxis es:

```
for (inicialización; condición; actualización) {

   //sentencias a ejecutar en cada iteración

}
```

El bucle `for` tiene tres partes incluidas entre los paréntesis, que nos sirven para definir cómo deseamos que se realicen las repeticiones. La primera parte es la inicialización, que se ejecuta solamente al comenzar la primera iteración del bucle. En esta parte se suele colocar la variable que utilizaremos para llevar la cuenta de las veces que se ejecuta el bucle.

La segunda parte es la condición, que se evaluará cada vez que comience una iteración del bucle. Contiene una expresión para decidir cuándo se ha de detener el bucle, o mejor dicho, la condición que se debe cumplir para que continúe la ejecución del bucle.

Por último tenemos la actualización, que sirve para indicar los cambios que queramos ejecutar en las variables cada vez que termina la iteración del bucle, antes de comprobar si se debe seguir ejecutando.

Un ejemplo de utilización de este bucle lo podemos ver a continuación, donde se imprimirán los números del 0 al 10.

```
<html>
<head><title>For</title></head>
<body>
<script language="JavaScript">
 var i
 for (i = 0; i <= 10; i++) {
    document.write(i)
    document.write("<br>")
 }
</script>
</body>
</html>
```

Estructura for…in

Una estructura de control derivada de `for` es la estructura `for...in`. Su definición exacta implica el uso de objetos, permitiendo recorrer las propiedades de un objeto. Es posible adaptar este comportamiento a los arrays:

```
for(indice in array) {

  ...

}
```

Ejemplo:

```
<html>
<head><title>For in</title></head>
<body>
<script language="JavaScript">
 var dias = ["Lunes","Martes","Miércoles","Jueves","Viernes","Sábado","Domingo"];
 for(i in dias) {
   console.log(dias[i]);
 }
</script>
</body>
</html>
```

Estructura try

La estructura try consiste en un bloque de código que se ejecuta de manera normal, y captura cualquier excepción que se pueda producir en ese bloque de sentencias. Su definición formal es la siguiente:

```
try {
    funcion_que_no_existe();
} catch(ex) {
    console.log("Error detectado: " + ex.description);
}
```

En este ejemplo, llamamos a una función que no está definida, y por lo tanto provoca una excepción en JavaScript. Este error es capturado por la cláusula `catch`, que contiene una serie de sentencias que indican que acciones realizar con esa excepción que acaba de producirse. Si no se produce ninguna excepción en el bloque `try`, no se ejecuta el bloque dentro de `catch`.

La cláusula `finally` contiene las sentencias a ejecutar después de los bloques `try` y `catch`. Las sentencias incluidas en este bloque se ejecutan siempre, se haya producido una excepción o no. Un ejemplo clásico de utilización de la cláusula `finally`, es la de liberar recursos que el script ha solicitado.

```
abrirFichero()
try {
   escribirFichero(datos);
} catch(ex) {
    // Tratar la excepción
} finally {
   cerrarFichero(); // siempre se cierra el recurso
}
```

Referencias:

http://www.desarrolloweb.com/articulos/508.php

http://codigomaldito.blogspot.mx/2011/05/arrays-en-javascript-ejemplos.html

http://www.jesusda.com/docs/ebooks/introduccion_javascript.pdf

http://www.webestilo.com/javascript/js08.phtml

http://jlmorgon.byethost4.com/tutojs/tutojs11.html?ckattempt=1

http://www.arkaitzgarro.com/javascript/capitulo-5.html