APLICACIONES WEB

LENGUAJES DE SCRIPT: JAVASCRIPT

1. INTRODUCCIÓN

JavaScript es un lenguaje **interpretado por el navegador**, basado en objetos y dirigido a proporcionar una mayor flexibilidad a los documentos web que, diseñados solamente con HTML, resultan estáticos. El **código acompaña al HTML** y permite una programación destinada a manipular los elementos de dichos documentos. Su primer nombre fue "LiveScript" y fué diseñado por Netscape. Microsoft contraatacó con su propia versión que denominó "JScript".

JavaScript fue diseñado **para dar cierto dinamismo al comportamiento de los documentos web**, ya que éstos resultan absolutamente estáticos si son desarrollados exclusivamente con HTML. JavaScript permite variar la apariencia o el comportamiento de los diferentes elementos que lo componen en función de la detección de lo que ocurre en ella, ya bien sea por haberlo provocado el usuario, o bien, el propio código del programa. Así, algunos de los usos más típicos que a JavaScript se le dan, son:

- Control del comportamiento del navegador
- Control de elementos obligatorios en un formulario
- Control de formatos de entrada en formularios
- Cambio de imágenes en un documento

La posibilidad de que se puedan escribir códigos maliciosos con intención de dañar o "invadir" los ordenadores que se conecten, es muy alta. Por esto, hay una serie de limitaciones en cuanto a la funcionalidad del lenguaje que intentan poner a salvo de estas malas prácticas al ordenador del usuario que se conecta:

- JavaScript no tiene capacidades gráficas
- El lenguaje no puede escribir ni leer de ficheros del disco del ordenador cliente
- No soporta ninguna gestión de transmisión, excepto la posibilidad de conectarse a una URL

Para programar en JavaScript necesitamos, básicamente, lo mismo que para desarrollar páginas web en HTML: **un editor de texto y un navegador compatible con JavaScript**, aunque hay editores específicos que nos pueden ayudar en la programación con códigos de colores, ayudas, ... (notepad ++, html kit, ...).

Veamos la pinta que tiene un código escrito en JavaScript:

```
 var numero1=23, numero2=63;
 if (numero1==numero2) {
    document.write (“Los dos números son iguales”);
}
else {
    if (numero1 > numero2) {
        document.write (“El primer número es mayor”);
    }
    else {
        document.write (“El segundo número es mayor”);
    }
}
```

2. CÓMO INSERTAR CÓDIGO JAVASCRIPT EN HTML

Las instrucciones de JavaScript se pueden incorporar a nuestros documentos de 4 formas diferentes:

- Usando la etiqueta <script>
- Asociadas a la detección de un evento
- Asociadas a un enlace
- En ficheros externos

2.1. Usando la etiqueta <script>

```
<script  type="text/JavaScript" language=”JavaScript”>
        Aqui vendría el código JavaScript
</script>
```

Las instrucciones que se escriban entre estas dos etiquetas **se ejecutan de forma secuencial** y, si estas instrucciones aparecen antes que algún código HTML, se interpretarán antes que éste.

Se pueden escribir en un documento HTML tantas etiquetas <script></script> como se desee, aunque lo normal es que exista solamente una que contenga todo el código necesario; sin embargo es posible encontrar casos en los que es necesario escribir el código de forma separada. Como norma general, se coloca dentro de la etiqueta <head>

EJEMPLO 1

```
<html>
     <head>
       <title>primer ejemplo JS</title>
        <script type="text/JavaScript" language=”JavaScript”>
               alert ('Bienvenido al primer ejemplo');
        </script>
     </head>
     <body>
        Texto inicial<br />

        <script  type="text/JavaScript" language=”JavaScript”>
               document.write('JavaScript dentro de la parte body');
        </script>

        <br />Texto final
     </body>
</html>
```

2.2. JavaScript en los eventos

Los eventos son acciones que realiza el usuario. JavaScript atrapa determinadas acciones realizadas en una página web y realiza acciones como respuesta, de esta manera se pueden realizar programas interactivos, ya que controlamos los movimientos del usuario y respondemos a ellos. Existen muchos eventos distintos, por ejemplo, pulsar un botón del ratón, moverlo o seleccionar texto en una página. El código JavaScript asociado a un evento se indica dentro la etiqueta HTML del elemento sobre el que se desea detectar el evento; concretamente dentro de un atributo llamando **manejador de eventos**. En este ejemplo, ***onClick*** se dispara cada vez que el usuario pulsa el botón. Una vez que se dispara, se ejecuta el código que aparece a su derecha entre comillas, después del símbolo "=".

EJEMPLO 2

```
<html>
    <head>
        <title>Eventos</title>
    </head>
    <body>
        <hr /><center>
        Esto sería una página con información y un formulario
      Cada vez que se pulse el botón, se ejecuta la instrucción de JavaScript
        <hr />
        <form name='datos' action=''>
           <input type='button' value='Pulsa'
                onClick="alert('Bienvenido al primer ejemplo')">
        </form>
        <hr /></center>
    </body>
</html>
```

OJO CON LAS COMILLAS: las comillas que se utilizan en JavaScript son, indistintamente la “ ” y la ‘ ’. Si hay que anidarlas en una misma instrucción, se alternan como se ha visto en el ejemplo.

2.3. JavaScript en los enlaces

Al igual que se puede asociar código a un evento, también se puede asociar código JavaScript a un hipervínculo, de tal manera que se ejecute al pulsar sobre el texto del enlace en vez de llevarnos a un URL.

EJEMPLO 3

```
<html>
    <head>
        <title>Enlaces</title>
    </head>
    <body>
        <hr /><center>
        Esto sería una página con información y un enlace
        Cada vez que se pulse el enlace, se ejecuta la instrucción de
        JavaScript<hr />
        <a href="JavaScript:alert('Ha pulsado el enlace')">Pulse aquí</a>
        <hr /></center>
    </body>
</html>
```

2.4. JavaScript en ficheros externos

La idea principal reside en la creación de ficheros que contienen **a modo de "librería"**, el código que se debe ejecutar en más de un documento web. De esta manera, la modificación de parte de este código solamente requiere la edición de un documento (aquel en el que reside la "librería"). La incorporación se realiza usando la etiqueta <script>, aprovechando la existencia del atributo src, que le indica al intérprete de JavaScript cuál es la URL del fichero que se desea incorporar al documento actual. Es habitual que los ficheros que contienen código JavaScript tengan la **extensión .js**. El fichero debe contener, exclusivamente, código JavaScript. No debe contener ninguna etiqueta HTML; ni siquiera <SCRIPT>.

EJEMPLO 4

```
<html>
    <head>
        <title>Librerias</title>
    </head>
    <body>
       <script type="text/JavaScript" language='JavaScript' src='funciones_externas.js'>
       </script>
       Aquí vendría información HTML
    </body>
</html>
```

3. COMENTARIOS

Un comentario es una parte del código que no es interpretada por el navegador y que ayuda al programador o a otro personal a entender lo que se está haciendo en el programa.

Comentarios de una línea: // Aquí va el texto del comentario

Comentarios de varias líneas: /* Esto es un comentario que puede
ocupar varias líneas */

4. JAVASCRIPT EN NAVEGADORES ANTIGUOS

Los navegadores antiguos no incluyen el intérprete de JavaScript; para evitar que se muestre el código JavaScript en la página web, se pone el código entre comentarios html:

```
<script language="JavaScript">
<!--Ocultación en navegadores antiguos
      Aquí vendría el código JavaScript
//-->
</script>
```

Esto además es conveniente hacerlo siempre porque en los navegadores actuales puede estar deshabilitada la opción de interpretar JavaScript (ejemplo: Internet Explorer: Herramientas-Opciones de Internet – Seguridad – Nivel personalizado – automatización de: subprogramas de Java, operaciones de pegado, secuencias de comandos ActiveX).

El cierre del comentario HTML va precedido del símbolo de comentarios de JavaScript, para evitar que el navegador lo interprete como código JavaScript y de un error, ya que no se deben usar etiquetas HTML dentro del código JavaScript.

5. CONSIDERACIONES SOBRE JAVASCRIPT

- Se diferencia entre mayúsculas y minúsculas (*hola* es distinto a *Hola*).
- Los espacios no cuentan.

- Cada instrucción debe terminar con el carácter **;** o con un salto de línea. Es recomendable utilizar siempre el ; .

- Se pueden escribir varias instrucciones por línea. Esto no es recomendable porque dificulta la lectura del código.

- Las cadenas de caracteres van entre comillas simples o dobles.

- JavaScript se ejecuta de forma secuencial, de forma que, hasta que no se termina de interpretar el código, no se continúa con el código HTML.

- Para mostrar por pantalla caracteres especiales, como las comillas, se utiliza el carácter de escape \. En relación a esto, las siguientes líneas serían equivalentes:

  ‘Esta información se escribe con el método \’write\’ ’
  “Esta información se escribe con el método ’write’ ”
  ‘Esta información se escribe con el método “write” ’

6. MÉTODO WRITE

document.write( ) es un método del objeto *document* que permite escribir código html dentro del documento actual.

EJEMPLO 5

```
<html>
     <head>
         <title>Primer ejemplo JS</title>
     </head>
     <body>
          Texto inicial
          <script language="JavaScript">
               document.write('<hr /><b>JavaScript</b> dentro de la parte body');
          </script>
          <br />Texto final
     </body>
</html>
```

La salida en el navegador es:

Texto inicial

JavaScript dentro de la parte body
Texto final

7. ALMACENAMIENTO DE DATOS

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. Cada vez que se utiliza el nombre de una variable, el navegador lo sustituye por su valor en ese momento.

JavaScript es un lenguaje no tipado, es decir, no es necesario declarar las variables indicando el tipo de valores que van a contener, aunque sí **es conveniente declararlas para dar más claridad al código**. Simplemente, **cuando se almacena en una variable un valor, el propio intérprete del navegador, la crea y le asigna dicho valor.**

La declaración de las variables, aunque no es obligatoria, se hace al principio del script y se utiliza la palabra reservada *var*. La declaración de variables consiste en definir e informar al sistema de que se va a utilizar una variable. Los nombres de las variables no pueden tener espacios blancos y deben comenzar por una letra o un guión bajo, y los valores se asignan con el símbolo **= .** También podemos declarar varias variables en una sola línea, separando las variables por comas. Cuando se le asigna un valor por primera vez a una variable se denomina **inicializar una variable.**

```
variable="hola"; //inicialización de variable
var cinco=5;
cuatro=4;
var nombre="Agustín", cadena=""; //declaración múltiple
sumando1=23;
sumando2=33;
suma = sumando1 + sumando2;
```

El nombre de las variables debe ser significativo y hacer referencia a su uso o contenido.

El **ámbito de una variable** **es el lugar donde está disponible**. Por lo general, las variables que declaremos en una página, estarán accesibles dentro de ella. En JavaScript no podremos acceder a variables que hayan sido definidas en otra página. Según su ámbito, tenemos dos tipos de variables**: locales y globales**. Las variables globales son "visibles" desde cualquier sitio del programa; las locales sólo son visibles en el bloque donde han sido declaradas.

- **Variables globales**: son accesibles desde cualquier lugar de la página, es decir, desde el script donde se han declarado y todos los demás scripts de la página.

```
<script>
        variableGlobal;
</script>
```

- **Variables locales**: son accesibles sólo en una parte acotada del código, y, fuera de este código, dejan de existir. Son ámbitos locales cualquier lugar acotado por llaves. No hay problema en declarar variables locales con el mismo nombre que otras globales, porque la que tiene validez es la última que se declara (en este caso, la local); aunque es más aconsejable no repetir nombres de variables dentro de una misma página. Es necesario que se utilice la declaración explícita con **var** para las variables locales, ya que, si no se utiliza, la variable será global.

```
<script>
.....
{
var variableLocal;
variableGlobal;   // NO se ha declarado con var
}
.....
</script>
```

Si no se declaran las variables explícitamente, se interpretarán como globales.

Los **tipos básicos** de datos que una variable puede contener son:

- Numéricos: enteros o reales
  - Enteros: decimal (2, -34), octal (02, -034), hexadecimal (0x2, -0x34)
  - Reales: notación decimal (3.11, -0.0011), notación científica (311E-2, 11E-4)

- Cadenas de caracteres: van entre comillas "dobles" o 'simples'.
  Hay una serie de caracteres especiales que sirven para expresar en una cadena de texto determinados controles (salto de línea, tabulador, ...); estos caracteres se llaman **caracteres de escape**, y se escriben siempre con la barra invertida:

  - \b – espacio hacia atrás
  - \n – nueva línea
  - \t - tabulación
  - \' – comilla simple
  - \r – retorno de carro
  - \f - avance de página
  - \\ - barra de escape

- \" – comilla doble

- Booleanos: Hay dos valores booleanos: true y false (verdadero y falso). Los booleanos se utilizan para tomar decisiones, en sentencias de control como la siguiente:

```
if  ( numero > 18 ) {
        // hacer algo
}
```

- Objetos: referencias a un objeto.

- UNDEFINED vs NULL

  Existe cierta confusión entre estos valores. La clave para entender la diferencia radica en la noción de **existencia**. Si una variable es:

  - `undefined`: para JavaScript, **no existe**. O bien no ha sido declarada o jamás se le asignó un valor.
  - `null`: para JavaScript, **la variable existe**. En algún momento, explícitamente, la variable se estableció a `null`. `null` significa nulo. `null` es un valor, es decir, no es lo mismo que vacío. Se trata de un valor que se puede asignar a una variable cuando queremos indicar que una variable no tiene un valor en concreto.

  Así que, a la hora de usar uno u otro debemos ver si estamos preguntando por la **existencia** de una variable. Si es el caso, usaremos **undefined**. En caso de que preguntemos por si la variable contiene un **valor seguro** o no, como ocurriría en Java, C++ o C# usaremos **null**.

```
undefined == null  // es true debido a la conversión de tipos
undefined === null // false
```

EJEMPLO 6

```
<html>
    <head>
          <title>Variables sin valor</title>
    </head>
    <body>
          <script type="text/JavaScript" language='JavaScript'>
            var Edad;
            switch (Edad) {
              case undefined:
                  document.write ("Undefined. La edad no tiene valor inicial.
                        Edad=" + Edad + "<br/>");
                  break;
              case null:
                  document.write ("La edad contiene el valor null.
                        Edad=" + Edad + "<br/>");
                  break;
              default:
                  document.write ("El valor de Edad es: " + Edad + "<br/>");
            }
          </script>
    </body>
</html>
```

Debido a que, como ya hemos dicho, JavaScript no es un lenguaje tipado, una misma variable puede contener valores de distinto tipo:

```
A=2;        //variable numérica
A="dos";    //variable cadena
A=true;     //variable booleana
```

LA FUNCIÓN typeof

Devuelve una cadena que describe el tipo de operando: boolean, number, string o object.

Sintaxis: **`typeof (valor)`**

EJEMPLO 7

```
<html>
    <head>
          <title>Uso del typeof</title>
    </head>
    <body>
      <script type="text/JavaScript" language=”JavaScript”>
            var MiNombre;
            var UnNumero=5, OtroNumero=10;
            var continuar = true;
            MiNombre="Ana";

            document.write('El contenido de MiNombre es ' + MiNombre + ' y es de
            tipo ' + typeof(MiNombre) + '<br />');

            document.write('El contenido de UnNumero es ' + UnNumero + ' y es de
            tipo ' + typeof(UnNumero) + '<br />');

            document.write('El contenido de OtroNumero es ' + OtroNumero + ' y es
            de tipo ' + typeof(OtroNumero) + '<br />');

            document.write('El contenido de continuar es ' + continuar + ' y es de
            tipo ' + typeof(continuar) + '<br />');
      </script>
    </body>
</html>
```

La salida que produce este código es:

El contenido de MiNombre es Ana y es de tipo string
El contenido de UnNumero es 5 y es de tipo number
El contenido de OtroNumero es 10 y es de tipo number
El contenido de continuar es true y es de tipo boolean

8. OPERADORES

Los operadores sirven para hacer los cálculos y operaciones necesarios para llevar a cabo los objetivos del programa.

OPERADOR	DESCRIPCIÓN
Operador Aritmético	**Realizan operaciones matemáticas simples.**
+ -	Suma. Resta.
* /	Multiplicación. División de números. `Ejemplo: 7/2 daría 3,5`
\	División entera de números. `Ejemplo: 7\2 daría 3`
%	Resto de la división entera. Es típico calcular si un nº es par haciendo: `num%2 == 0`
++ --	Incremento (suma 1) / Decremento (resta 1) de una variable
Operadores de Relación	**Comparan dos valores devolviendo *true* o *false* en función de que se cumpla la relación indicada.**
==	Igual que
===	Igual que en sentido estricto . Los comparadores estrictos exigen no solo que los valores comparados sean iguales en valor, sino también en tipo. Ejemplo: cadena31 = "31"; numero31 = 31;

	test1 = (cadena31 == numero31); // true
	test2 = (cadena31 === numero31); // false
	alert("(" + cadena31 + " == " + numero31 + ") = " + test1);
	alert("(" + cadena31 + " === " + numero31 + ") = " + test2);

!=	Distinto
!==	Distinto en sentido estricto
> >=	Mayor - Mayor o igual
< <=	Menor - Menor o igual
Operadores Lógicos	**Se aplican sobre operadores booleanos y sirven para tomar decisiones:** **joven = (edad >= 18) && (edad <= 30);** //se deben cumplir las dos condiciones **correcto = !(a < b) || (c == d);**//se debe cumplir al menos una de las dos condiciones
!	Not : transforma true en false y viceversa
&&	And : obliga a que se cumplan todas las condiciones para devolver *true*. Si hay alguna que sea falsa, devuelve *false*.
||	Or : devuelve true si una de las condiciones es *true*. Si todas son falsas, devuelve *false*.
Operadores de Asignación	**Sirven para asignar valores a las variables.**
=	Asigna un valor a una variable
+=	Asignación con suma x += y equivale a x = x + y
-=	Asignación con resta x -= y equivale a x = x - y
*=	Asignación con producto x *= y equivale a x = x * y
/=	Asignación con división x /= y equivale a x = x / y
%=	Asignación con resto x %= y equivale a x = x % y
Operadores de Cadenas	**Operadores que se aplican sobre cadenas de caracteres.**

+	Concatena cadenas de caracteres.
	A=1+2 // Operador aritmético suma. A contiene el valor 3
	A=”1”+2 //A contiene el valor “12”
	A=1+”2” //A contiene el valor “12”
	A=”1”+”2” //A contiene el valor “12”

FUNCIONES DE CONVERSIÓN DE TIPOS:

Al existir un operador que tiene dos funciones en función del tipo de los operandos (+ suma números o concatena cadenas), puede haber cierta confusión si utilizamos este operador con un dato numérico y otro cadena. JavaScript, en este caso, entenderá que queremos concatenar y actuará como si los dos datos fueran de tipo cadena. Existen funciones de conversión de tipos que nos pueden ayudar en estos casos:

- **parseInt (cadena, base)** examina la cadena recibida y trata de convertirla en un número entero en la base especificada como parámetro (decimal si no se indica). Devuelve el valor especial NaN (Not a Number) si no puede obtener un número.

- **parseFloat (cadena)** convierte la cadena que se le pasa en un real. Si no lo consigue, asigna el valor especial NaN.

En JavaScript también existe la función **isNaN(valor)** que comprueba si el valor es NaN. Devuelve true si el valor no es un número (*Not a Number*) y false cuando lo es. Esta función se suele utilizar en combinación con las anteriores para comprobar si la conversión ha funcionado correctamente:

```
x = parseInt(“A3”);
if ( isNaN(x) ) alert(“la conversión ha fallado”);
else alert(“El número convertido es ” + x);
```

LA FUNCIÓN EVAL

eval(cadena); devuelve el resultado de ejecutar el código JavaScript contenido en la cadena.

```
x=5;
eval("document.write(" + x + ")"); /* la función eval reconstruye la
instrucción document.write y la ejecuta: document.write(x); */

eval("x=10;y=20;document.write(x*y)"); // escribe 200
document.write("<br />" + eval("2+2"));  // escribe 4
```

EL ORDEN DE PRECEDENCIA DE LOS OPERADORES:

En principio, todos los operadores se evalúan de izquierda a derecha, pero hay una serie de reglas llamadas **orden de precedencia** que indican el orden en que se deben ejecutar determinados operadores. De menor a mayor son las siguientes:

```
Asignación = += -= *= /= %=
Condicional ?:
"o" lógica ||
"y" lógica &&
Igualdad == !=
Comparación < <= > >=
Suma y Resta + -
Multiplicación, División y Módulo * / %
Negación, Incremento ! ~ ++ --
Paréntesis ( ) []
```

El orden de precedencia se puede alterar usando los paréntesis.

9. INTERACCIÓN CON EL USUARIO

Existen tres funciones que permiten interaccionar con el usuario:

- **alert ("***texto***");** Aparece una ventana con un botón Aceptar.

EJEMPLO 8

```
<html>
    <head><title>alert y prompt</title></head>
    <body>
       <script type='text/JavaScript' language='JavaScript'>
         var Edad;
         Edad = prompt ('Dame tu edad');
         alert ('Tu edad es ' + Edad);
       </script>
     </body>
</html>
```

- variable = **confirm("***texto***");** Aparece una ventana con dos botones (Aceptar y Cancelar). El botón que se pulsa es el que se almacena en la variable (true o false respectivamente).

EJEMPLO 9

```
<html>
    <head><title>confirm</title></head>
    <body>
        <script type='text/JavaScript' language='JavaScript'>
           var confirmación;
           confirmacion = confirm ("Mensaje de confirmación");
           alert ("Has pulsado " + confirmacion);
        </script>
```

```
    </body>
</html>
```

- variable = **prompt(“***texto***”,”***cadena que aparece por defecto***”);** Aparece una ventana en la que se pide un dato al usuario con dos botones (OK y Cancel). Si se pulsa OK el valor tecleado se almacena en la variable.

EJEMPLO 10

```
<html>
    <head><title>prompt</title></head>
    <body>
        <script type='text/JavaScript' language='JavaScript'>
            var Edad;
            Edad = prompt ('Dame tu edad');
            alert ('Tu edad es ' + Edad);
        </script>
    </body>
</html>
```

Hay que observar que en la ventana que abre esta función como valor por defecto aparece *undefined*. Si no queremos que le aparezca este valor al usuario, podemos evitarlo: prompt acepta un segundo parámetro que será la cadena que debe aparecer por defecto. Si el valor que le pasamos a este segundo parámetro es una cadena vacía, conseguimos que no aparezca el indeseado valor: **prompt ('Dame tu edad', “”)**

Pero ¿qué ocurre si introducimos en la caja de texto un valor que no sea numérico?: esto no provoca un error del programa pero sí que es un error de funcionamiento. Para evitarlo podemos ejecutar la función parseInt para convertir una cadena de caracteres en número.

10. ESTRUCTURAS DE CONTROL

Dentro de la programación estructurada, tenemos 2 tipos distintos de instrucciones:

- **alternativas**: if, switch. Sirven para tomar decisiones sobre las acciones a realizar en función del estado de las variables.
- **repetitivas:** for, while, do-while. Se utilizan para realizar ciertas acciones repetidamente.

Todas las palabras clave de las estructuras de control se escriben siempre en **minúsculas**.

10.1 LA INSTRUCCIÓN *IF*

```
if (condición) {
      Bloque 1;
}

else {
      Bloque 2;
}
```

Esta instrucción evalúa una condición: si es verdadera, ejecuta las instrucciones del primer bloque, y si es falsa, se ejecutan las instrucciones del bloque *else* (si se ha incluido puesto que es opcional).

Si el bloque de instrucciones que se deben ejecutar está compuesto por una única instrucción, entonces las llaves son optativas.

El contenido de cada uno de los bloques debe escribirse con un sangrado a la derecha, ya que esto da más claridad al código, sobre todo, cuando varios programadores se encarguen de trabajar con el mismo código.

EJEMPLO 11

```
<html>
    <head>  <title>if</title>  </head>
    <body>
        <script type='text/JavaScript' language='JavaScript'>
            var Edad;
            Edad = prompt ('Dame tu edad',"");
            if ( Edad < 18 )
                alert ('Eres muy joven');
            else
                alert ('Eres mayor de edad');
        </script>
    </body>
</html>
```

Hay una **versión corta** para la instrucción if que se usa mucho para comprobar pequeñas condiciones:

```
variable = (condicion)? valor1 : valor2;
```

es equivalente a:

```
if (condicion)
     variable = valor1;
else
     variable = valor2;
```

Las instrucciones IF se pueden anidar, es decir, podemos colocar sentencias IF dentro de otras sentencias IF, dentro de la parte IF o dentro de la parte ELSE.

EJEMPLO 12

```
<html>
    <head>  <title>if anidado</title>  </head>
    <body>
        <script type='text/JavaScript' language='JavaScript'>
            var numero1=23, numero2=63;
            if (numero1==numero2)
                document.write (“Los dos números son iguales”);
            else {
                 if (numero1 > numero2)
                    document.write (“El primer número es mayor”);
                 else
                    document.write (“El segundo número es mayor”);
            }
        </script>
    </body>
</html>
```

10.2 LA INSTRUCCIÓN SWITCH

La sentencia **switch** permite la ejecución de un bloque de instrucciones en función del valor que tome una variable. La sintaxis es:

```
switch (expresión)
   {
   case valor1:
      Bloque de instrucciones resultado1;
      break;
   case valor2:
      Bloque de instrucciones resultado 2;
      break;
   ......
   default:
      Bloque de instrucciones por defecto;
   }
```

La expresión se evalúa; si vale valor1 se ejecutan las sentencias del bloque1, si vale valor2 se ejecutan las sentencias del bloque2, y así sucesivamente. En caso de que no coincida ninguno de los valores escritos se ejecuta el bloque de instrucciones especificados en la cláusula default si existe (es optativa). La sentencia break sirve para que se ejecuten solo las instrucciones correspondientes a un único valor del switch. Si no se pone break, se ejecutarían todas las sentencias que existan hasta el final del switch.

Esta instrucción se utiliza normalmente para sustituir sentencias if anidadas, de tal manera que se mejora la legibilidad del programa.

```
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");
}
```

Notar que los valores 6 y 7 tienen el mismo bloque de instrucciones asociado; por eso y el valor es 6, al no haber break; se continúa la ejecución y se ejecutan las instrucciones del valor 7.

10.3 LA INSTRUCCIÓN *FOR*

Se utiliza para repetir una serie de instrucciones un número determinado de veces. Su sintaxis es:

```
for (inicialización; condición; actualización) {
       Bloque de instrucciones;
}
```

El paréntesis tiene 3 partes:

- inicialización: se ejecuta **solamente** la primera vez que se ejecuta el bucle, y se utiliza para asignar el primer valor que va a tomar la variable que controla el bucle.
- Condición: se evaluará cada vez que se repite el bucle. Cuando la condición no se cumpla, dejará de dar vueltas el bucle.
- Actualización: indica los cambios que queremos que se produzcan en la variable que controla el bucle cada vez que se da una vuelta.

El funcionamiento de la instrucción es:

1. Se evalúa la expresión inicialización.
2. Se evalúa la expresión condición. Si el resultado de dicha evaluación es falso, se abandona el bucle sin ejecutar ninguna de las instrucciones del bloque.
3. Si el resultado de la evaluación anterior es verdadero, se ejecutan las instrucciones del bloque.
4. Se ejecuta la expresión actualización.
5. Se vuelve al paso 2.

EJEMPLO 13

```
<html>
    <head> <title>for</title> </head>
    <body>
        <script type='text/JavaScript' language='JavaScript'>
            var x, indice;

            x = prompt ('Introduce el número de vueltas: ', "0");
            for ( indice=0; indice<x; indice++ )
                document.write('Vuelta número ' + indice + '<br/>');

            document.write('<hr />');
            for ( indice=x; indice>0; indice-- )
                document.write('Vuelta número ' + indice + '<br/>');
        </script>
    </body>
</html>
```

Un ejemplo de uso de un bucle for en una página web es el siguiente:

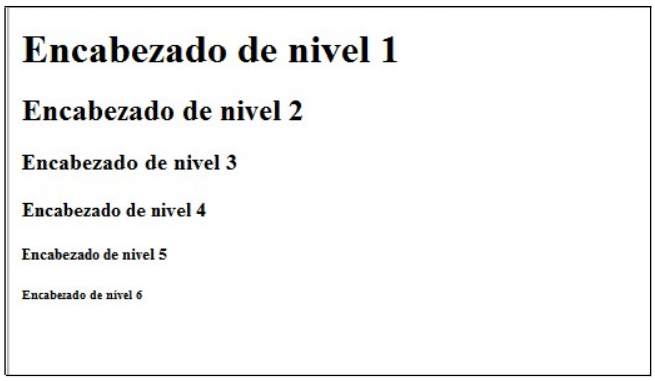
EJEMPLO 14

```
<html>
   <head>  <title>Ejemplo bucle FOR</title>  </head>
   <body>
      <script>
         var i;
         for (i=1; i<=6; i++) {
          document.write("<H" + i + ">Encabezado de nivel " + i + "</H" + i + ">");
         }
      </script>
   </body>
</html>
```

El script escrito en negrita sería equivalente a escribir el siguiente código HTML:

```
<h1>Encabezado de nivel 1</h1>
<h2>Encabezado de nivel 2</h2>
<h3>Encabezado de nivel 3</h3>
<h4>Encabezado de nivel 4</h4>
<h5>Encabezado de nivel 5</h5>
<h6>Encabezado de nivel 6</h6>
```

Y la salida que se produce sería:



10.4 LA INSTRUCCIÓN *WHILE*

Este bucle se utiliza cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición.

```
while (condición){
       Bloque de instrucciones
}
```

El bucle se repite mientas la condición sea verdadera.

EJEMPLO 15

```
<html>
   <head> <title>Ejemplo de while</title> </head>
   <body>
      <script type="text/JavaScript" language='JavaScript'>
          var x, índice, Edad;
          x = prompt ('Introduce el número de vueltas: ', "0");
          indice=0;
          while ( indice < x ) {
             document.write('Vuelta número ' + indice + '<br />');
             indice++;
          }

     /* el siguiente bucle se repite hasta que se introduzca un
     valor de edad */

          Edad = parseInt(prompt ('<br />Dame tu edad: ', '0'));
          while ( (Edad == 0) || isNaN(Edad) )
          Edad = prompt ('Dame tu edad',"");

          alert('La edad introducida es: ' + Edad);
      </script>
   </body>
</html>
```

10.5 LA INSTRUCCIÓN *DO-WHILE*

Este bucle es una variación del buce while. Se utiliza cuando no sabemos cuantas veces se tendrá que ejecutar el bucle, con la diferencia de que sabemos seguro que el contenido se va a ejecutar una vez. Este bucle se introdujo en JavaScript 1.2, por lo que no todos los navegadores lo soportan. En cualquier caso, todo lo que queramos hacer con do-while se puede hacer con while.

```
do {
       Bloque de instrucciones
} while (condición);
```

El bucle se repite mientas la condición sea verdadera. Este tipo de bucles **se ejecutan, al menos, una vez.**

EJEMPLO 16

```
<html>
    <head> <title>Bucles while</title> </head>
    <body>
       <script type="text/JavaScript" language='JavaScript'>
          var Edad;

/*el bucle se repite hasta que se introduzca un valor edad*/

          do
             Edad = prompt ('Dame tu edad',"");
          while ( (Edad == 0) || isNaN(Edad) );

          alert ('La edad que me has dado es: ' + Edad);
       </script>
    </body>
</html>
```

10.6 LA INSTRUCCIÓN *BREAK*

La instrucción BREAK se utiliza dentro de los bucles para forzar el fin de las iteraciones, independientemente de que la condición que controla el bucle siga siendo verdadera. Significa *detener la ejecución del bucle y salir de él*.

EJEMPLO 17

```
<html>
   <head> <title>break</title> </head>
   <body>
      <center>
      <script  type="text/JavaScript" >
         <!--
         var i;
         for (i=0; i< 10; i++) {
             document.write ('iteración: ' + i + '<br />');
             if ( i == 3) break;
         }
         document.write ('Fin del bucle en la tercera iteración');
         //-->
      </script>
      </center>
   </body>
</html>
```

10.7 LA INSTRUCCIÓN *CONTINUE*

La instrucción CONTINUE se utiliza dentro de los bucles para forzar la vuelta al principio del bucle para ejecutar la siguiente iteración sin que se terminen de ejecutar las demás instrucciones del bucle. Significa *detener la ejecución y volver al principio del código del bucle*.

EJEMPLO 18

```
<html>
   <head> <title>continue</title> </head>
   <body>
      <center>
      <script type="text/JavaScript">
         <!--
         var i;
         for (i=0; i< 5; i++) {
             if (i == 2) continue;
      /*el mensaje de la iteración 2 no se mostrará*/
             document.write ('iteración: ' + i + '<br />');
         }
         document.write ('Se ha terminado el bucle');
         //-->
      </script>
      </center>
   </body>
</html>
```

10.8 BUCLES ANIDADOS

Anidar un bucle consiste en meter ese bucle dentro de otro. Un bucle anidado tiene una estructura similar a la siguiente:

```
for (i=0; i<10; i++) {
      for (j=0; j<10; j++) {
            document.write (i + “-” + j);
      }
}
```

En este ejemplo las llaves de apertura y cierre no harían falta, se han puesto por claridad. Su funcionamiento es el siguiente: para cada valor de i, se ejecuta todo el bucle j. Su salida sería:

```
0-0
0-1
0-2
0-3
…
0-8
0-9
1-0
1-1
…
1-8
1-9
…
9-9
```

11. FUNCIONES

Una función es un trozo de código que permite desarrollar una tarea concreta y bien definida, que se encuentra separado del resto de instrucciones del programa y al que se le ha dado un nombre para que, posteriormente, pueda ser referenciado. A través del nombre se pueden ejecutar las instrucciones contenidas en él tantas veces como sea necesario.

El uso de funciones evita la repetición de código innecesario y da mayor legibilidad al código.

La sintaxis general para la declaración de una función es:

```
function nombre_función (parámetros){
        Instrucciones de la función;
}
```

Para ejecutar una función utilizamos su nombre con los paréntesis.

El uso de funciones es generalizado, se utilizan constantemente. De hecho, muchas están ya definidas en el sistema (por ejemplo, eval, parseInt, isNaN, ...).

EJEMPLO 19

```
<html>
    <head>  <title>funciones</title>  </head>
    <body>
        <script type="text/JavaScript">
            <!--
            function Tabladel3( ){     //declaración de la función
                with (document)
                {
                  write ('<table border=1>');
                  write ('<tr>');
                  for (i=0; i < 10; i ++)
                      write ('<td> 3*'+i+'='+3*i+'<\/td>');
                  write ('<\/tr>');
                  write ('<\/table>');
                  /* podemos encontrar el carácter "\" para evitar que los
                  navegadores antiguos se confundan con /t */
               }
            }

            Tabladel3( );  // llamada y ejecución de la función
            //-->
        </script>
    </body>
</html>
```

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre las etiquetas SCRIPT. No obstante existe una limitación a la hora de colocarlas con relación a los lugares desde donde se llame. Tenemos dos opciones:

- **Colocar la función en el mismo bloque de script desde el que se la va a llamar**. En este caso no importa colocar la declaración después de la llamada, aunque no es recomendable.
- **Colocar la función en otro bloque de script**. En este caso es obligatorio que el SCRIPT en el que se declara la función esté antes que el SCRIPT en el que se llama a la función. En otro caso, daría error.

Lo más recomendable es declarar todas las funciones de la página en la parte HEAD, para, después, hacer las llamadas correspondientes desde la parte BODY.

Las funciones, al ejecutarse, pueden devolver valores con la sentencia return que, a su vez, pueden ser asignados a variables. De hecho ésta es la gran utilidad de las funciones.

EJEMPLO 20

```
<html>
    <head>  <title>funciones</title>  </head>
    <body>
        <script type="text/JavaScript" language="JavaScript">
            <!--
            function Mifuncion( ) {
                var devolver = prompt("Escribe algo","");
                return devolver;
            }
            var escrito = Mifuncion( );
            document.write(escrito);
            //-->
         </script>
    </body>
</html>
```

Las funciones sólo pueden devolver un valor, aunque en su cuerpo puede haber colocados varios **return**. Una vez que se ejecuta un return, la función termina su ejecución, aunque tenga más código debajo.

```
function multipleReturn( ) {
    numero=8;
    if ((numero % 2) == 0) return 0;
    else return 1;
    document.write (“este mensaje no se vería nunca”);
}
```

La función anterior sería equivalente a la siguiente. ¿Sabrías explicar porqué?

```
function multipleReturn( ) {
    numero=8;
    if ((numero % 2) == 0) return 0;
    return 1;
    document.write (“este mensaje no se vería nunca”);
 }
```

Las funciones, además de salida, también pueden tener entrada de datos mediante el paso de parámetros. Los parámetros se utilizan para enviar datos a las funciones. Por ejemplo, una función que suma dos números tendría estos dos números como parámetros de entrada y la suma sería el valor de salida de la función. Los parámetros hay que declararlos en la cabecera de la función y, por supuesto, ésta debe ser invocada con el número correcto de parámetros que se han declarado. Los parámetros pueden ser de cualquier tipo de datos.

Los parámetros pueden ser comparados con variables a los que se les asigna un determinado valor en el momento de la llamada a la función.

EJEMPLO 21

```
<html>
    <head>
        <title>funciones</title>

        <script type="text/JavaScript" language="JavaScript">
            <!--
            function escribirBienvenida(nombre){
                document.write ("<h1>Hola " + nombre + "<\/h1>");
            }
            //-->
        </script>
    </head>
    <body>
        <script type="text/JavaScript" language="JavaScript">
             <!--
             escribirBienvenida("Juan García");
             //-->
        </script>
    </body>
</html>
```

Una función puede recibir todos los parámetros que consideremos necesario, y debemos tener cuidado en poner los valores de estos parámetros en el orden correcto a la hora de llamar a la función para que los trate correctamente.

EJEMPLO 22

```
<html>
    <head> <title>funciones</title>
        <script type="text/JavaScript" language="JavaScript">
            <!--
            function escribirBienvenida(nombre, color){
                document.write ('<font color="' + color + '">');
                document.write ("<h1>Hola " + nombre + "</h1>");
                document.write ("</font>");
            }
            //-->
        </script>
    </head>

    <body>
        <script type="text/JavaScript" language="JavaScript">
            <!--
            escribirBienvenida("Juan García en rojo", "#FF0000");
            escribirBienvenida("Juan García en verde", "#00FF00");
            escribirBienvenida("Juan García en azul", "#0000FF");
            escribirBienvenida("Juan García en amarillo", "yellow");
            //-->
        </script>
    </body>
</html>
```

En JavaScript los parámetros se pasan por valor; es decir, a la función se pasan sólo los valores, no las variables. Esto significa que cualquier modificación que haga la función sobre los parámetros de entrada no tendrá efecto fuera de la función.

Las variables declaradas con var dentro de las funciones son siempre locales a éstas, lo que quiere decir que no se tendrá acceso a sus valores desde fuera de dichas funciones. Si dentro de una función se declaran variables sin var o se usan sin declararlas explicitamente, serán variables globales y “vivirán” fuera de la función.

EJEMPLO 23

```
<html>
    <head>
        <title>funciones</title>
         <script type="text/JavaScript" language="JavaScript">
             <!--
             function pasoPorValor(parametro){
                    parametro=35;
                    document.write ('Dentro de la función, el valor del
                parámetro es '+ parametro + '<br />');
             }
             //-->
         </script>
    </head>
    <body>
        <script type="text/JavaScript" language="JavaScript">
             <!--
             var variable=5;
             document.write ('El valor de la variable antes de la
                              función es '+ variable + '<br />');
             pasoPorValor(variable);
             document.write ('El valor de la variable después de la
                              función es '+ variable + '<br />');
             //-->
        </script>
    </body>
</html>
```

La salida es:

> El valor de la variable antes de la función es 5
> Dentro de la función, el valor del parámetro es 35
> El valor de la variable después de la función es 5

EJEMPLO 24

```
<html>
   <head> <title>funciones</title>  </head>
   <body>
   <script type="text/JavaScript">
   <!--
   varglobal1 = 5;
   varglobal2 = 8;

   function MuestraContenido(parametro){
     var varlocal1 = 3;
     document.write('Dentro de función, parametro es '+ parametro + '<br />');
     document.write('Dentro de función, varglobal2 es '+ varglobal2 + '<br />');
     document.write('Dentro de función, varlocal1 es '+ varlocal1 + '<br />');
   }

   function Suma(parametro){
     var varlocal1 = 2;
     document.write('Dentro de función, parametro es '+ parametro + '<br />');
     parametro = parametro + varlocal1;
     document.write('Dentro de función, parametro ahora es '+parametro+'<br />');
     document.write('Dentro de función, varglobal2 es '+ varglobal2 + '<br />');
     document.write('Dentro de función, varlocal1 es '+ varlocal1 + '<br />');
   }

   document.write('Fuera de la función, varglobal1 es '+ varglobal1 + '<br />');
   document.write('Fuera de la función, varglobal2 es '+ varglobal2 + '<br />');
   // La siguiente instrucción está comentada porque aquí no existe varlocal1
```

```
   // document.write('Fuera de la función, varlocal1 es '+ varlocal1 + '<br />');

   MuestraContenido(varglobal1);
   Suma(varglobal1);
   document.write('Fuera de la función, varglobal1 es '+ varglobal1 + '<br />');
   //-->
   </script>
   </body>
</html>
```

La salida es:

Fuera de la función, varglobal1 es 5
Fuera de la función, varglobal2 es 8
Dentro de la función, parametro es 5
Dentro de la función, varglobal2 es 8
Dentro de la función, varlocal1 es 3
Dentro de la función, parametro es 5
Dentro de la función, parametro ahora es 7
Dentro de la función, varglobal2 es 8
Dentro de la función, varlocal1 es 2
Fuera de la función, varglobal1 es 5

12. ERRORES COMUNES Y CONSEJOS DE PROGRAMACIÓN

Los típicos errores comunes que se comenten al programar en JavaScript son:

- Utilizar = en expresiones condicionales en lugar de ==
- No conocer la precedencia de operadores y no utilizar paréntesis para agrupar operaciones a realizar.
- Usar las comillas simples y dobles erróneamente: dentro de las comillas dobles se deben utilizar comillas simples y viceversa.
- Olvidarse de cerrar una llave o cerrar una llave de más.
- Colocar varias sentencias en una misma línea sin separarlas por punto y coma.
- Utilizar una variable antes de inicializarla o no declarar las variables con var antes de utilizarlas. Si no se declaran las variables, se interpretarán como globales.
- Contar las posiciones de un vector o cadena desde el 1 en lugar de comenzar en 0.

La depuración de errores en JavaScript no es una tarea fácil. JavaScript nos informará de errores de sintaxis o de aquellos que tienen lugar en el momento de la ejecución al escribir mal los nombres de variables o funciones. Para otro tipo de errores no disponemos de herramientas potentes de depuración, sino que tendremos que aplicar una técnica rudimentaria que consiste en utilizar *alert( )* en puntos estratégicos del programa para controlar los valores que van tomando las variables.

Algunos consejos que se deberían seguir a la hora de crear código JavaScript son:

- Las funciones deben ser razonablemente pequeñas, para, así, poder reutilizarlas con facilidad.
- Utilizar nombres de variables y funciones que representen su contenido.
- Comentar el código.
- Utilizar sangrados para colocar los distintos bloques de código dentro del programa.

13. BIBLIOGRAFÍA

- Apuntes extraídos del material elaborado por los profesores de la Escuela Universitaria de Informática de la Universidad Politécnica de Madrid (Santiago Alonso Villaverde y otros)
- Programación en JavaScript I y II, Miguel Ángel Álvarez y Manu Gutiérrez. Desarrolloweb.com