NOTA:

Todo este material fue seleccionado de páginas web, por lo cual si encuentran contenido de algún sitio web. No se extrañen ya q eh visto conveniente este material para su comprensión y aprendizaje espero les sea útil.

Introducción a JavaScript

JavaScript, es un lenguaje de programación de páginas web de lado del cliente, esto significa, que cuando estamos viendo una página que utiliza JavaScript, hemos descargado el código JavaScript a nuestro navegador y nuestro navegador lo está ejecutando de acuerdo con las acciones realizadas en la página.

Para que sirve y para que no

Gracias a que se ejecuta en el navegador, JavaScript, nos permite responder de manera rápida y eficaz a las acciones del usuario, creando de esta manera aplicaciones interactivas.

Pero el hecho de ejecutarse en el navegador, da lugar a otros problemas, por ejemplo, el código ejecutado es enviado al navegador, por lo que puede ser obtenido pornuestros usuarios, de manera que no podremos utilizar JavaScript para proteger secciones con contraseña...

Además, JavaScript, es ejecutado por el navegador y cada fabricante de este tipo de software interpreta el mismo código javaScript a su manera, por lo que tendremos que tener mucho cuidado si queremos que todo el mundo vea correctamente nuestra página.

Un ejemplo

Un clásico ejemplo de JavaScript es el típico popup o las alertas, en este caso vamos a hacer una alerta que dará el clasico 'hola mundo':

```
000 <script>
001 alert('Hola mundo');
002 </script>
```

Cómo usarlo

Por norma general, podremos utilizar JavaScript en nuestras páginas HTML, incluyendo el código entre las etiquetas <script> y </script> cómo hemos visto en el ejemplo anterior, pero existen otras formas de usarlo cómo los ficheros.js o en los eventos (que veremos más adelante).

Variables JavaScript

Una variable es un dato que nuestra aplicación tiene en memória y que puede modificar y obrener grácias a un nombre de variable único que la identifica, el cual tendrá que ser formado por números y letras y el caracter _ , no pudiendo empezar por número.

Usando variables

A diferéncia de otros lenguajes, en javascript, las variables no tienen muchas más particularidades, veamos un ejemplo y luego lo explicamos con mas calma:

```
000  <script>
001  nombre="Eloi";
002  nombre=nombre+"deSanMartín";
003  alert(nombre);
004  </script>
```

En la primera línea del ejemplo, vemos cómo se asigna a la variable *nombre* la cadena de caracteres 'Eloi', en la siguiente línea, se modifica el valor de ésta variable y se le da el valor actual más la cadena 'de San Martín', seguidamente se muestra una alerta con el resultado 'Eloi de San Martin'.

Como vemos en la segunda línea, obrnemos el valor de la variable ('Eloi'), simplemente escribiendo su nombre (*nombre*), y como vemos en las dos primeras líneas le damos valor con el operador = (que veremos más adelante).

Variables globales y locales

Por defecto, JavaScript, cuando encuentra una variable la define cómo global (que se puede ver desde cualquier parte de la página), pero nosotros podemos hacer que una variable sea local (visible sólo desde la función actual) usando el sufijo var la primera vez que la definimos:

```
000  <script>
001  function m(){
002   var nombre = "Eloi";
003   nombre = nombre + " de San Martín";
004  }
005  m();
006  alert(nombre);
```

```
007 </script>
```

De esta manera, la variable *nombre*, sólo podrá ser vista desde el interior de la función **m** y no será mostrada en la alerta

Tipos de datos en JavaScript

JavaScript dispone de siete tipos de datos, que se definen de maneras distintas, estos datos, se almacenan en variables , y por la naturaleza de JavaScript (lenguaje no tipado), estas variables pueden pasar de contener un tipo de dato a otro tipo y podemos operar con tipos de dato diferentes.

Vamos a ver que tipos de dato existen:

Números

Son datos numéricos que se asignan con el número en cuestión, veamos un ejemplo:

```
000 <script>
001 var = 33;
002 </script>
```

Cadenas de caracteres

Son datos de texto y deben estar delimitados por comillas simples o dobles, pueden incluir caracteres especiales cómo \' (comilla) \" (comilla doble), \n (salto de línea), \t (tabulador)...

```
000 <script>
001 var = "Hola\namigo";
002 </script>
```

Booleanos

Son datos de bool, sus valores significan verdadero (true) y falso (false), se escriben sin comillas

```
000 <script>
001 var = true;
002 </script>
```

Objetos

Son conjuntos de variables y funciones definidos previamente por el lenguaje (objetos predefinidos) o por el usuario, son ejemplos de objetos predefinidos los objetos Image y Array, para definir-las, usaremos el prefijo new y por el uso de paréntesis:

```
000 <script>
001 var = new Image();
002 </script>
```

Nulos

Són datos vacios, se producen cuando se ha definido una variable como null para borrarla:

```
000 <script>
001 var = null;
002 </script>
```

Indefinidas

Són variables que ahún no han sido asignadas con el operador de asignación (=)

```
000 <script>
001 var;
002 </script>
```

No numérico

Se producen cuando no podemos realizar una operación determinada porque los datos no son compatibles, su valor es NaN (not a number):

```
000 <script>
001 var = 'a'*1;
002 </script>
```

Estructuras de control

Para controlar la ejecución de una acción o bloque de acciones una o mas veces disponemos de lo que se llama estructuras de control, a continuación explicaré las estructuras de control que usaremos más comunmente:

El condicional IF

Realiza un bloque de acciones si se cumple una condición y si no se cumple realiza el bloque de acciones de else.

```
000 <script>
001 if(2 == 2){
002 alert('2 es igual a 2');
003 }
004 else {
005 alert('2 es distinto de 2');
006 }
007 </script>
```

El ciclo FOR

Realiza un bloque de acciones desde una acción inicial a una condición.

```
000 <script>
001 for(i=0;i<10;i=i+1){
002 alert(i);
003 }
004 </script>
```

El primer parámetro de for (i=0) indica el estado inicial, es decir, cuando empiece el ciclo i será 0. La segunda es la condición final, se repetirá alert(i) hasta que i sea 9. La tercera es una operación que se ejecuta cada vez que se repite.

El ciclo WHILE

Repite un bloque de acciones mientras se repite la condición entre paréntesis.

Cuando el bloque de acciones modifica la situación que hacía cumplir la condición y la condición no se cumple, se deja de repetir el bloque de acciones.

```
000 <script>
001 i = 0;
002 while(i < 10){
003 alert(i);
004 i=i+1;
005 }
006 </script>
```

Repetirá alert(i) y i=i+1 mientras i sea menor que 10.

Operaciones

Javascript dispone de una gran cantidad de operaciones que se pueden efectuar sobretodo sobre valores numéricos, pero las más importantes y las que se usan más a menudo són las siguientes:

Operaciones aritméticas

Las operaciones aritméticas són aquellas que nos permiten el trabajo con números tales como la suma, la resta,...

Los operadores mas comunes són el de suma (+), el de resta (-), el de multiplicación (*) y el de división (/).

```
000 <script>
001 a = 3 - 5 // a valdrá -2
002 a = 3; b = 4 // a valdrá 3 y b 4
003 c = a + b // c valdrá 7
004 </script>
```

Además de estos, contamos con otros operadores tales como resto de la division (%) y los operadores de incremento y decremento (++ y --)

```
000 <script>
001 a = 4 // a valdrá 4
002 a++ // a valdrá 5
003 a-- // a volverá a valer 4
004 </script>
```

El operador (+) lo podremos usar también para la concatenación de cadenas:

```
000 <script>
001 a = "java"
002 b = "script"
003 c = a+b // c valdrá "javascript"
004 </script>
```

Operaciones comparativas

Para hacer una comparacion para manejar una estructura de control, debemos indicar una expresión a evaluar.

Para hacerla tenemos 8 operadores de comparación de los que explicaremos la igualdad (==), la diferéncia (!=), el menor (<), mayor (>), menor o igual (<=) y

mayor o igual (>=).

Estos comparadores devolverán true o false según si se cumple o no la expresión:

```
000 <script>
001 a = (2 == 2) // a valdrá true
002 b = (3 != 3) // b valdrá false
003 </script>
```

Operaciones lógicas
Nos permiten complicar las expresiones a evaluar, insertando mas de una comparación en la misma expresión.

Los operadores de los que disponemos son and (&&), que devuelve true si se cumplen ambas comparaciones, or (||) que lo hace si se cumple alguna y not (!) que nega una expresión.

```
000 <script>
001 a = ((1 == 1) && (2 == 5)) // a valdrá false
002 </script>
```

Como vemos, también podemos usar paréntesis para indicar la prioridad en las operaciones.

Vectores y matrices

Un vector (array, cadena, arreglo...) es un conjunto ordenado de variables que se almacenan bajo un mismo nombre y se distinguen por un indice.

¿Como creo un vector?

Los vectores en JavaScript son objetos del lenguaje (objetos Array) y se definen como tales.

```
000 <script>
001 dias = new Array ('lunes', 'martes', 'miercoles', 'jueves', 'viernes', 'sabado', 'domingo');
    </script>
```

En el ejemplo creamos un vector con los nombres de los días de la semana, el primero tendría indice 0, el segundo indice 1 y asi sucesivamente...

¿Como accedo a un elemento?

Si quedemos acceder al primer elemento del vector (lunes), escrivbiremos el nombre del vector y entre parentesis cuadrados pondremos el indice del elemento:

```
000 <script>
001 dias = new Array ('lunes', 'martes', 'miercoles', 'jueves', 'viernes', 'sabado', 'domingo');
002 alert(dias[0]);
    </script>
```

Esta forma de acceder a los datos, también sirve para modificar y crear nuevos elementos, veamos un ejemplo:

```
000 <script>
001 dias = new Array ('lunes', 'martes', 'miercoles', 'jueves', 'viernes', 'sabado', 'domingo');
002 dias[0] = 'Monday';
    alert(dias[0]);
    </script>
```

Hemos modificado el primer elemento (0) y pasa de valer 'lunes' a valer 'Monday', si el elemento lunes no existiera, se crea dinámicamente.

¿Y que hay de las matrices?

Las matrices són vectores de dos dimensiones, si las queremos representar gráficamente tendrían el siguiente aspecto:

e00 e01 e02
e10 e11 e12
e20 e21 e22

La forma de crear una matriz es crear un vector y hacer que cada elemento de este vector sea otro vector, para hacerlo de forma ordenada así:

```
000 <script>
001 matriz = new Array ();
002 matriz[0] = new Array ('e00', 'e01', 'e02');
003 matriz[1] = new Array ('e10', 'e11', 'e12');
004 matriz[2] = new array ('e20', 'e21', 'e22');
005 </script>
```

Y sabiendo que matriz[0] es new Array ('e00', 'e01', 'e02'), tenemos que si queremos acceder a e01, haremos matriz[0][1]

Funciones

Por lo general, cuando programamos en cualquier lenguaje hay partes de código a las que recurrimos constantemente y no tiene sentido escribirlas en todas las partes en las que vamos a utilizarlas, es este caso es extremadamente útil escribirlas una vez y luego utilizar repetidas veces lo que hemos escrito.

Para esto, existen lo que llamaremos funciones y que realmente son una parte de código delimitada que podemos llamar cuando nos parezca necesario.

Para aumentar sus posibilidades, las funciones incorporan lo que llamamos parámetros (datos que son recibidos por la función al ser llamada) y valores de retorno (datos que són enviados por la función al finalizar).

Como es una función

La sintaxis de las funciones en JavaScript es muy sencilla, se identifican con el nombre **function** seguido de el nombre de la función, los parámetros entre paréntesis y separados por comas (son nombres de variables que estarán disponibles dentro de la función) y luego todo su código entre corchetes ({) y (}), veamos un ejemplo:

```
000 <script>
001 function saludar(texto, numero) {
002  alert(texto);
003  return numero;
004 }
005 </script>
```

En este caso, la función se llama 'saludar', tiene los parámetros 'texto' y 'numero', su contenido es un alert(texto)' y su valor de retorno' es 'numero'.

Como llamo a una función

Una vez hemos creado esa parcela de código, debemos poder ejecutarla, esto es lo que se llama *llamar a la función*, para hacerlo escribiremos el nombre de la función y entre paréntesis los valores de los parámetros que queremos enviarle.

Para utilizar sus valores de retorno, usaremos la función cómo si fuera una variable, vamos a ver un ejemplo que llame a la función anterior:

```
000 <script>
001 num = saludar("hola pepe", 4);
002 alert(num); // nos alertará el numero 4
003 </script>
```

En este ejemplo, llamamos a la función 'saludar' con los valores "hola pepe" y 4 para los parámetros definidos 'texto' y 'numero', la función alertará texto (que será "hola pepe") y devolverá numero (que será 4). Se almacenará el valor devuelto en 'num' y se alertará (// nos alertará el numero 4)

Funciones predefinidas

Todos los lenguajes (y JavaScript no es una excepción), además de permitir definir funciones, pero también llevan funciones ya definidas, un ejemplo de ello es **alert**, esta función que hemos utilizado en los ejemplos anteriores es una función predefinida.

Las funciones predefinidas, son como las que podemos escribir nosotros, pero ya vienen hechas con el JavaScript, de esta manera nos ahorramos escribirlas nosotros mismos

JavaScript orientado a objetos

No todos los desarrolladores están de acuerdo sobre si JavaScript es un lenguaje de programación orientado a objetos o no, pero es indiscutible que es una herramienta excelente para el desarrollo de aplicaciones que trabajen con trabajar con clases y objetos.

Tabla de contenido

Clases y Objetos

Para entender la programacion orientada a objetos debemos primero entender lo que es una clase, pero primero diremos que un objeto es un conjunto de atributos y metodos agrupados.

Una clase es un grupo de objetos que comparten los mismos atributos y métodos, veamos como podemos crear una clase llamada *popup*, como observareis no hay ninguna diferéncia con una funcion solo que al crearlo usamos **new**:

```
000  <script language="javascript">
001  // Creamos la clase
002  function popup ( ) {
003   // ...
004  }
```

```
005  ventana = new popup ();
006  </script>
```

Los atributos

La clase que acabamos de crear tendrá sus atributos, que podrán ser publicos (accesibles desde fuera del objeto) o privados (solo accesibles desde el código del objeto).

Vamos a añadir un atributo público a la clase que hemos creado para indicar la URL del popup:

```
000  <script language="javascript">
001  // Creamos la clase
002  function popup ( ) {
003   // Atributo público inicializado a about:blank
004   this.url = 'about:blank';
005   // ...
006  }
007  ventana = new popup ();
008  </script>
```

Y le añadimos también una clase privada para almacenar el objeto window de la ventana abierta.

```
000  <script language="javascript">
001  // Creamos la clase
002  function popup ( ) {
003   // Atributo público inicializado a about:blank
004   this.url = 'about:blank';
005   // Atributo privado para el objeto window
006   var ventana = null;
007   // ...
008  }
009  ventana = new popup ();
010  ventana.url = 'http://www.educasis.net/';
011  </script>
```

Los métodos

Al crear un objeto de tipo imagen se ejecutará el metodo **constructor** de la imagen que será en el caso de nuestro ejemplo la funcion *popup*, cuando se ejecuta, este crea el objeto y todas sus variables (atributos) y funciones (metodos).

En el interior de esta función constructor, podemos crear nuevas funciones públicas y privadas que a su vez, podrían ser constructor de un objeto 'hijo', vamos a completar

el ejemplo anterior añadiendo un par de metodos.

```
000 <script language="javascript">
001 function popup ( ) {
002  // Atributo público inicializado a about:blank
003  this.url = 'about:blank';
004  // Atributo privado para el objeto window
005  var ventana = null;
006  // Metodo público para abrir el popup
007  this.abrir = function ( ) {
008  // Generamos la ventana
009  ventana = window.open ( this.url );
010  // Si no hay ventana llamamos al error
011  if ( ! ventana ) error ( 'El popup ha sido bloqueado' );
012  }
013  // Metodo privado para alertar un mensaje en caso de error
014  var error = function ( texto ) {
015  // Mostramos el error
016  alert ( texto );
017  }
018 }
019 ventana = new popup ();
020 ventana.url = 'http://www.educasis.net /';
021 ventana.abrir ();
022 </script>
```

La visibilidad de los elementos

Los elementos de un objeto (metodos y clases) són accesibles desde el exterior del mismo objeto cuando los creamos con **this.** tal y como hemos observado en el ejemplo con *url* y *abrir*:

```
000 <script language="JavaScript">
001 // ...
002 ventana.url = 'http://www.educasis.net /';
003 ventana.abrir ();
004 </script>
```

Por otro lado tambien hemos creado los objetos *ventana* y *error*, pero al crearlos con **var** solo serán accesibles desde el interior de la clase y sus subclases y funciones:

```
000 <script language="JavaScript">
001 // ...
002 ventana.url = 'http://www.educasis.net /';
003 // Llamamos a un metodo privado
004 // Error: ventana.error is not a function
005 ventana.error ( 'Hola amigo' );
006 </script>
```

Objeto Math

El Objeto Math te permite realizar tareas matematicas comunes. Incluye varias funciones y constantes Matematicas.

Para usar estas funciones y constantes, es necesario usar "**Math.**", para especificar que se encuentran dentro del objeto Math (Por ejemplo Math.abs() o Math.PI).

Funciones:

abs(x)
Devuelve el valor absoluto de x

acos(x)
Devuelve el arco-coseno de x

asin(x)
Devuelve el arco-seno de x

atan(x)
Devuelve el arco-tangente de x

atan2(y,x)
Devuelve el angulo de un punto (x,y)

ceil(x)
Devuelve el valor de x redondeado al entero superior

cos(x)
Devuelve el coseno de x

exp(x)
Devuelve el valor de E elevado a la x potencia

floor(x)
Devuelve el valor de x redondeado al entero inferior

log(x)
Devuelve el logaritmo natural de x

max(x,y)
Devuelve el valor del mayor de sus argumentos, x o y

min(x,y)
Devuelve el valor del menor de sus argumentos, x o y

pow(x,y)
Devuelve el valor de x elevado a la y potencia

random()
Devuelve un numero aleatorio entre 0 y 1

round(x)
Devuelve x redondeado al entero mas cercano

sin(x)
Devuelve el seno de x

sqrt(x)
Devuelve la raiz cuadrada de x

tan(x)
Devuelve la tangente del angulo x

Constantes:

E
Devuelve el valor de e (constante de euler) (2.718281828459045)

LN2
Devuelve el logaritmo natural de 2 (0.6931471805599453)

LN10
Devuelve el logaritmo natural de 10 (2.302585092994046)

LOG2E
Devuelve el logaritmo base-2 de E (1.4426950408889634)

LOG10E
Devuelve el logaritmo base-10 de E (0.4342944819032518)

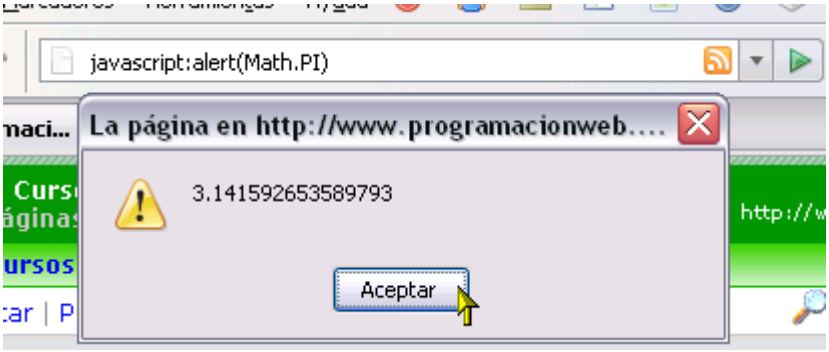
PI
Devuelve PI (3.141592653589793)

SQRT1_2
Devuelve la raiz cuadrada de 1/2 (0.7071067811865476)

SQRT2

Devuelve la raiz cuadrada de 2 (1.4142135623730951)

Debido a que es posible ejecutar codigo JavaScript desde la barra de direcciones, una manera rapida de usar estas funciones y constantes es la siguiente:



Claro, tambien las puedes usar en tu codigo html, dentro de:

```
000 <script type="text/javascript">
001
002 numero = -7
003 alert( Math.abs(numero) )
004 //El resultado seria "7"
005
006 </script>
```

Codigos de ejemplo.... POO.

Código Script del constructor de clases

```
//Construimos el objeto
function constructorObjeto(idObj) {
//Como variable publica del objeto definimos el innerTexto, que recibira
el argumento que se le pasa a la funcion que es el id del link pulsado y
extraemos el innerHTML.
```

```
this.innerTexto = document.getElementById(idObj).innerHTML;
 //Me creo un metodo público que me saca por medio de un alert el string
que le pase
 this.sacarInnerTexto =function(strTexto) {
 alert(strTexto);
 }
}
function LlamarAlObjeto(paramValue) {// Definimos un nuevo objeto por
medio de nuestro constructor, asi en la siguiente linea podemos acceder
a todos sus metodos y variables
var miObjeto = new constructorObjeto(paramValue);
//Ahora llamamos al metodo sacarInnerTexto de nuestro objeto y le
pasamos la variable innerTexto que hemos definido en el interior del
objeto miObjeto.sacarInnerTexto(miObjeto.innerTexto);
}
```

Código HTML del constructor de clases

```
<div class="center">
<div class="contenedor">
<h1>Ejemplo de constructor de un Objeto</h1>
<a href="#" onclick="LlamarAlObjeto(this.id) ;return false"
id="link1">Llamar al Objeto</a>
</div>
</div>
```

Código Script del creador de métodos y variables públicas

```
//Construimos el objeto
function constructorObjeto(idObj) {
//Como variable publica del objeto definimos el innerTexto, que recibira
el argumento que se le pasa a la funcion que es el id del link pulsado y
extraemos el innerHTML.
this.innerTexto = document.getElementById(idObj).innerHTML;
//Me creo un metodo público que me saca por medio de un alert el string
que le pase
this.sacarInnerTexto = function(strTexto) {
alert(strTexto);
}
```

```
}
function LlamarAlObjeto(paramValue) {
// Definimos un nuevo objeto por medio de nuestro constructor, asi en la
siguiente linea podemos acceder a todos sus metodos y variables
var miObjeto = new constructorObjeto(paramValue);
//Ahora llamamos al metodo sacarInnerTexto de nuestro objeto y le
pasamos la variable innerTexto que hemos definido en el interior del
objeto miObjeto.sacarInnerTexto(miObjeto.innerTexto);
}
```

Código HTML del creador de métodos y variables públicas

```
<div class="center">
<div class="contenedor">
<h1>Ejemplo de creación de metodo y variables
públicas</h1>
<a href="#" onclick="LlamarAlObjeto(this.id) ;return false"
id="link1">Llamar al Objeto</a>
</div>
</div>
```

Código Script del creador de métodos y variables privadas

```
//Construimos el objeto
function constructorObjeto(idObj) {
//Defino metodos y variables privadas que no pueden ser accesibles
desde fuera del objeto
var texto = "Texto privado";
var metodoPrivado = function () {
alert(texto+"----Esto es un metodo privado que utiliza una variable
privada y que se accede a el solo desde dentro del objeto")
}
metodoPrivado();
}
function LlamarAlObjeto(paramValue) {
// Definimos un nuevo objeto por medio de nuestro constructor, asi en la
siguiente linea podemos acceder a todos sus metodos y variables
var miObjeto = new constructorObjeto(paramValue);
}
```

Código HTML del creador de métodos y variables privadas

```
<div class="center">
<div class="contenedor">
<h1>Ejemplo de creación de metodo y variables privadas</h1>
<a href="#" onclick="LlamarAlObjeto(this.id) ;return false"
id="link1">Llamada correcta a un metodo privado</a>
</div>
</div>
```

constructor de un objeto creador de herencias

```
function creaHerencia(objDestino, objOrigen) {
var sConstructor = objOrigen.toString();
var aMatch = sConstructor.match( /\s*function (.*)\(/ );
if ( aMatch != null ) {
objDestino.prototype[aMatch[1]] = objOrigen;
}
for (var m in objOrigen.prototype) {
objDestino.prototype[m] =objOrigen.prototype[m];
}
};
```

Código Script del creador de herencias

```
//Funcion que copia las clases
function creaHerencia(objDestino, objOrigen) {
var sConstructor = objOrigen.toString();
var aMatch = sConstructor.match( /\s*function(.*)\(/ );
if ( aMatch != null ) {objDestino.prototype[aMatch[1]] = objOrigen; }
for (var m in objOrigen.prototype) {
objDestino.prototype[m] =objOrigen.prototype[m];
}
};
// Funcion que crea las variablespropias de cada objeto
function creaciondeIdentificadores(obj) {
this.id = document.getElementById(obj)};
// metodo del objeto "creaciondeIdentificadores" que saca el mensaje de
```

```
alerta
creaciondeIdentificadores.prototype.identificador = function() {
alert("el valor del campo es:"+this.id.value);
};
function datosCampo(obj) {
// Llamamos al objeto de referencia
this.creaciondeIdentificadores(obj);
};
// Llamamos a la funcion que hara que datosCampo herede de
creaciondeIdentificadores creaHerencia(datosCampo,
creaciondeIdentificadores); function verValordeCampos(obj) {
var obj = new datosCampo(obj.id);
obj.identificador();
}
```

Código HTML del creador de herencias

```
<div class="center">
<div class="contenedor">
<h1>Ejemplo de creación de herencias</h1>
< input type="text" value="" name="campo1" id="campo1"
onblur="verValordeCampos(this)">
</div>
</div>
```