

CONSTRUCCIÓN DE PÁGINAS WEB

HTML



CSS



JAVASCRIPT



mardeasa



Integración de componentes Software en páginas web

Javascript



Introducción a JavaScript (1)

¿Qué es JavaScript?



- JavaScript es un lenguaje de programación utilizado para crear pequeños programitas encargados de realizar acciones dentro del ámbito de una página web dinámica.
- Una página web dinámica es aquella que incorpora efectos, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.
- JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos.
- El navegador del cliente es el encargado de interpretar las instrucciones Javascript y ejecutarlas para realizar estos efectos e interactividades.

¿Qué es JavaScript?

- A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems, como se puede ver en <http://www.sun.com/suntrademarks/>.



JAVASCRIPT



HISTORIA

Breve historia

- A principios de los años 90 empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.
- Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario.
- **Brendan Eich**, un programador que trabajaba en Netscape, pensó que podría solucionar este problema...
- Curiosidad: *es también uno de los fundadores de Mozilla.*



Breve historia



- **Brendan Eich** adaptó otras tecnologías existentes (como *ScriptEase*) al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje *LiveScript*.
- Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación.
- Justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript.
- ***La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.***



Breve historia



- La primera versión de JavaScript fue un completo éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1.
- Al mismo tiempo, Microsoft lanzó **JScript** con su navegador Internet Explorer 3.
- **JScript** era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.



Internet Explorer 3.0



Breve Historia



- Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript.
- De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA (*European Computer Manufacturers Association*).



- ECMA creó el comité TC39 con el objetivo de *"estandarizar de un lenguaje de script **multiplataforma e independiente** de cualquier empresa"*.
- El primer estándar que creó el comité TC39 se denominó **ECMA-262**, en el que se definió por primera vez el lenguaje **ECMAScript**.

Breve Historia



- Por este motivo, algunos programadores prefieren la denominación *ECMAScript* para referirse al lenguaje JavaScript.
- De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.
- En Junio de **1997** se publicó la primera edición del estándar ECMA-262.
- Un año después, en Junio **de 1998** se realizaron pequeñas modificaciones para adaptarlo al estandar ISO/IEC-16262 y se creó la **segunda edición**.
- La **tercera edición del estándar ECMA-262** (*publicada en Diciembre de 1999*) es la versión que utilizan los navegadores actuales.



USOS DE JAVASCRIPT

Usos de Javascript



- En realidad se pueden ver ejemplos de Javascript dentro de cualquier página un poco compleja. Algunos que habremos visto en innumerables ocasiones son calendarios dinámicos para seleccionar fechas, calculadoras o convertidores de divisas, editores de texto enriquecido, navegadores dinámicos, etc.
- Es mucho más habitual encontrar Javascript para realizar efectos simples sobre páginas web, o no tan simples, como pueden ser navegadores dinámicos, apertura de ventanas secundarias, validación de formularios, etc. Nos atrevemos a decir que este lenguaje es realmente útil para estos casos, pues estos típicos efectos tienen la complejidad justa para ser implementados en cuestión de minutos sin posibilidad de errores. Sin embargo, aparte de esos unos simples ejemplos, podemos encontrar dentro de Internet muchas aplicaciones que basan parte de su funcionamiento en Javascript, que hacen que una página web se convierta en un verdadero programa interactivo de gestión de cualquier recurso. Ejemplos claros son las aplicaciones de ofimática online, como Google Docs, Office Online o Google Calendar.

Efectos rápidos con Javascript



Abrir una ventana secundaria

Primero vamos a ver que con una línea de Javascript podemos hacer cosas bastante atractivas. Por ejemplo podemos ver cómo abrir una ventana secundaria sin barras de menús que muestre el buscador Google. El código sería el siguiente:

```
<script>  
window.open("http://www.google.com","", "width=550,height  
=420,menubar=no")  
</script>
```

Efectos rápidos con Javascript



Un mensaje de bienvenida

Podemos mostrar una caja de texto emergente al terminarse de cargar la portada de nuestro sitio web, que podría dar la bienvenida a los visitantes:

```
<script>  
window.alert("Bienvenido a mi sitio web. Gracias...")  
</script>
```

Efectos rápidos con Javascript

Fecha actual

Veamos ahora un sencillo script para mostrar la fecha de hoy. A veces es muy interesante mostrarla en las webs para dar un efecto de que la página está al "al día", es decir, está actualizada:

```
<script> document.write(new Date()) </script>
```

Estas líneas deberían introducirse dentro del cuerpo de la página en el lugar donde queramos que aparezca la fecha de última actualización.

Efectos rápidos con Javascript

Botón de volver

Otro ejemplo rápido se puede ver a continuación. Se trata de un botón para volver hacia atrás, como el que tenemos en la barra de herramientas del navegador. Ahora veremos una línea de código que mezcla HTML y Javascript para crear este botón que muestra la página anterior en el historial, si es que la hubiera.

```
<input type=button value=Atrás onclick="history.go(-1)">
```

Como diferencia con los ejemplos anteriores, hay que destacar que en este caso la instrucción Javascript se encuentra dentro de un atributo de HTML, onclick, que indica que esa instrucción se tiene que ejecutar como respuesta a la pulsación del botón.

JAVASCRIPT



CÓMO TRABAJAR CON JAVASCRIPT

Incluir JavaScript en documentos XHTML



En el mismo
documento
XHTML

En un archivo
externo

En los
elementos
XHTML

Incluir JavaScript en el mismo documento XHTML

- El código JavaScript se encierra entre etiquetas `<script>` y se incluye en cualquier parte del documento. Aunque es *correcto incluir cualquier bloque de código en cualquier zona* de la página, se recomienda definir el código JavaScript dentro de la cabecera del documento (*dentro de la etiqueta `<head>`*):

`<head>`

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

```
<title>Ejemplo de código JavaScript en el propio documento</title>
```

```
<script type="text/javascript"> alert("Un mensaje de prueba");
```

```
</script>
```

`</head>`



Ejemplo
Ejemplo

Definir JavaScript en un archivo externo

- Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos XHTML enlazan mediante la etiqueta <script>.
- Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite.

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

- Archivo codigo.js:

```
alert("Un mensaje de prueba");
```



Incluir JavaScript en los elementos XHTML

- Este último método es el menos utilizado, ya que consiste en incluir trozos de JavaScript dentro del código XHTML de la página:

```
<body>
```

```
    <p onclick="alert('Un mensaje de prueba')">
```

Haga click aquí.

```
    </p>
```

```
</body>
```

El mayor inconveniente de este método es que *ensucia* innecesariamente el código XHTML de la página y complica el mantenimiento del código JavaScript. En general, este método sólo se utiliza para definir algunos eventos y en algunos otros casos especiales, como se verá más adelante.



Ejemplo

Ελεωβιο

Etiqueta noscript

El lenguaje HTML define la etiqueta `<noscript>` para mostrar un mensaje al usuario cuando su navegador no puede ejecutar JavaScript o no lo tiene activado.

```
<body>  
  <noscript>  
    <p>Bienvenido a Mi Sitio</p>  
    <p>La página que estás viendo requiere para su  
funcionamiento el uso de JavaScript. Si lo has deshabilitado  
intencionadamente, por favor vuelve a activarlo.</p>  
  </noscript>  
</body>
```

La etiqueta `<noscript>` se debe incluir en el interior de la etiqueta `<body>` (normalmente se incluye al principio de `<body>`). El mensaje que muestra `<noscript>` puede incluir cualquier elemento o etiqueta XHTML.



Ejemplo

ΕΙεωβjo

JAVASCRIPT



SINTAXIS JAVASCRIPT

Sintaxis en Javascript

- No se tienen en cuenta los espacios en blanco y las nuevas líneas
- Se distinguen las mayúsculas y minúsculas
- No se define el tipo de las variables
- No es necesario terminar cada sentencia con el carácter de punto y coma (;) pero si recomendable.
- Se pueden incluir comentarios:

// Ejemplo de comentario de una sola línea

/*Ejemplo de

comentario de

varias líneas: */

Posibilidades y limitaciones

- JavaScript fue diseñado de forma que se ejecutara en un entorno muy limitado
- Los scripts tampoco pueden cerrar ventanas que no hayan abierto esos mismos scripts.
- Los scripts no pueden acceder a los archivos del ordenador del usuario (ni en modo lectura ni en modo escritura) y tampoco pueden leer o modificar las preferencias del navegador.
- Si la ejecución de un script dura demasiado tiempo (por ejemplo por un error de programación) el navegador informa al usuario de que un script está consumiendo demasiados recursos y le da la posibilidad de detener su ejecución.

Conceptos básicos



Por operatividad, para entender y realizar ejercicios, debemos conocer algunas funciones básicas de Javascript:

Función alert

Esta función es la encargada de mostrar una pequeña ventana de aviso en la pantalla, así, si se requiere que aparezca un mensaje cuando ocurra determinada acción en el programa, podemos hacer uso de esta función.

```
alert(mensaje a mostrar);
```

Función write

Esta función es un método del objeto document y lo que hace es escribir en la página el texto que se ingresa como parámetro.

```
document.write(mensaje);
```

Función prompt

Con esta función aparece una ventana en la pantalla, con un espacio para el valor que se debe introducir y un botón aceptar para que la información sea guardada. Esta función recibe dos parámetros, el primero es el mensaje que se muestra en la ventana y el segundo es el valor inicial del área de texto.

```
variable = prompt(mensaje, valor inicial a mostrar);
```

JAVASCRIPT



VARIABLES

Variables en Javascript



Las variables son uno de los elementos fundamentales a la hora de realizar los programas, en Javascript y en la mayoría de los lenguajes de programación existentes.

Variables en Javascript

Concepto de variable

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. Por ejemplo, si nuestro programa realiza sumas, será muy normal que guardemos en variables los distintos sumandos que participan en la operación y el resultado de la suma. El efecto sería algo parecido a esto.

```
sumando1 = 23
```

```
sumando2 = 33
```

```
suma = sumando1 + sumando2
```

En este ejemplo tenemos tres variables, sumando1, sumando2 y suma, donde guardamos el resultado. Vemos que su uso para nosotros es como si tuviésemos un apartado donde guardar un dato y que se pueden acceder a ellos con sólo poner su nombre.

Variables en Javascript

Los nombres de las variables han de construirse con caracteres alfanuméricos y el carácter subrayado (_). Aparte de ésta, 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

También hay que evitar utilizar nombres reservados como variables, por ejemplo no podremos llamar a nuestra variable palabras como return o for, que ya veremos que son utilizadas para estructuras del propio lenguaje.

Veamos ahora algunos nombres de variables que no está permitido utilizar

- 12meses
- tu nombre
- return
- pe%pe

Variables en Javascript

Declaración de variables en Javascript

Declarar variables consiste en definir y de paso informar al sistema de que vas a utilizar una variable. Es una costumbre habitual en los lenguajes de programación el definir las variables que se van a usar en los programas y para ello, se siguen unas reglas estrictas. Pero Javascript se salta muchas reglas por ser un lenguaje un tanto libre a la hora de programar y uno de los casos en los que otorga un poco de libertad es a la hora de declarar las variables, ya que no estamos obligados a hacerlo, al contrario de lo que pasa en la mayoría de los lenguajes de programación.

Javascript cuenta con la palabra "var" que utilizaremos cuando queramos declarar una o varias variables. Como es lógico, se utiliza esa palabra para definir la variable antes de utilizarla.

Nota: Aunque Javascript no nos obligue a declarar explícitamente las variables, es aconsejable declararlas antes de utilizarlas y veremos en adelante que se trata también de una buena costumbre. Además, en algunos casos especiales, no producirá exactamente los mismos resultados un script en el que hemos declarado una variable y otro en el que no lo hagamos.

Ámbito de las variables en Javascript

Concepto de ámbito de variables

Se le llama ámbito de las variables al lugar donde éstas están disponibles. Por lo general, cuando declaramos una variable hacemos que esté disponible en el lugar donde se ha declarado, esto ocurre en todos los lenguajes de programación y como Javascript se define dentro de una página web, **las variables que declaremos en la página estarán accesibles dentro de ella.**

En Javascript no podremos acceder a variables que hayan sido definidas en otra página. Por tanto, la propia página donde se define es el ámbito más habitual de una variable y le llamaremos a este tipo de **variables globales** a la página. Veremos también se pueden hacer variables con ámbitos distintos del global, es decir, variables que declararemos y tendrán validez en lugares más acotados.

Ámbito de las variables en Javascript

Variables globales

Como hemos dicho, las variables globales son las que están declaradas en el ámbito más amplio posible, que en Javascript es una página web. Para declarar una variable global a la página simplemente lo haremos en un script, con la palabra *var*.

```
<SCRIPT>
```

```
    var variableGlobal
```

```
</SCRIPT>
```

Las 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, incluidos los manejadores de eventos, como el onclick, que ya vimos que se podía incluir dentro de determinadas etiquetas HTML.

Ámbito de las variables en Javascript

Variables locales

También podremos declarar variables en lugares más acotados, como por ejemplo una función. A estas variables les llamaremos locales. Cuando se declaren variables locales sólo podremos acceder a ellas dentro del lugar donde se ha declarado, es decir, si la habíamos declarado en una función solo podremos acceder a ella cuando estemos en esa función.

Las variables pueden ser locales a una función, pero también pueden ser locales a otros ámbitos, como por ejemplo un bucle. En general, son ámbitos locales cualquier lugar acotado por llaves.

```
<SCRIPT>
    function miFuncion (){
        var variableLocal

    }
</SCRIPT>
```

En el script anterior hemos declarado una variable dentro de una función, por lo que esa variable sólo tendrá validez dentro de la función. Se pueden ver cómo se utilizan las llaves para acotar el lugar donde está definida esa función o su ámbito.

Ámbito de las variables en Javascript

Diferencias entre declarar variables con var, o no declararlas

Como hemos dicho, en Javascript tenemos libertad para declarar o no las variables con la palabra var, pero los efectos que conseguiremos en cada caso serán distintos. En concreto, cuando utilizamos var estamos haciendo que la variable que estamos declarando sea local al ámbito donde se declara. Por otro lado, si no utilizamos la palabra var para declarar una variable, ésta será global a toda la página, sea cual sea el ámbito en el que haya sido declarada.

En el caso de una variable declarada en la página web, fuera de una función o cualquier otro ámbito más reducido, nos es indiferente si se declara o no con var, desde un punto de vista funcional. Esto es debido a que cualquier variable declarada fuera de un ámbito es global a toda la página. La diferencia se puede apreciar en una función por ejemplo, ya que si utilizamos var la variable será local a la función y si no lo utilizamos, la variable será global a la página. Esta diferencia es fundamental a la hora de controlar correctamente el uso de las variables en la página, ya que si no lo hacemos en una función podríamos sobrescribir el valor de una variable, perdiendo el dato que pudiera contener previamente.

Ámbito de las variables en Javascript



```
<SCRIPT>
  var numero = 2
  function miFuncion (){
    numero = 19
    document.write(numero) //imprime 19
  }
  document.write(numero) //imprime 2
  //llamamos a la función
  miFuncion()
  document.write(numero) //imprime 19
</SCRIPT>
```

En este ejemplo, tenemos una variable global a la página llamada `numero`, que contiene un 2. También tenemos una función que utiliza la variable `numero` sin haberla declarado con `var`, por lo que la variable `numero` de la función será la misma variable global `numero` declarada fuera de la función. En una situación como esta, al ejecutar la función se sobrescribirá la variable `numero` y el dato que había antes de ejecutar la función se perderá.



TIPOS DE DATOS

Tipos de datos en Javascript

Tipo de datos numérico

En este lenguaje sólo existe un tipo de datos numérico, al contrario que ocurre en la mayoría de los lenguajes más conocidos.

Todos los números son por tanto del tipo numérico, independientemente de la precisión que tengan o si son números reales o enteros. Los números enteros son números que no tienen coma, como 3 o 339. Los números reales son números fraccionarios, como 2.69 o 0.25, que también se pueden escribir en notación científica, por ejemplo 2.482e12.

Tipo booleano

El tipo booleano, boolean en inglés, sirve para guardar un si o un no o dicho de otro modo, un verdadero o un falso. Se utiliza para realizar operaciones lógicas, generalmente para realizar acciones si el contenido de una variable es verdadero o falso.

Si una variable es verdadero entonces Ejecuto unas instrucciones, Si no Ejecuto otras.

Los dos valores que pueden tener las variables booleanas son true o false.

Tipos de datos en Javascript

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.

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

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc.) Para resolver estos problemas, JavaScript define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto.

El mecanismo consiste en sustituir el carácter problemático por una combinación simple de caracteres. A continuación se muestra la tabla de conversión que se debe utilizar:

Tipos de datos en Javascript

Tabla con todos los caracteres de escape

- Salto de línea: `\n`
- Comilla simple: `\'`
- Comilla doble: `\"`
- Tabulador: `\t`
- Retorno de carro: `\r`
- Contrabarra: `\\`

Algunos de estos caracteres probablemente no los llegarás a utilizar nunca, pues su función es un poco rara y a veces poco clara.

Tipos de datos en Javascript

Tipo Array

En ocasiones, a los arrays se les llama vectores, matrices e incluso arreglos.

Un array es una colección de variables, que pueden ser todas del mismo tipo o cada una de un tipo diferente. Su utilidad se comprende mejor con un ejemplo sencillo: si una aplicación necesita manejar los días de la semana, se podrían crear siete variables de tipo texto.

En este tipo de casos, se pueden agrupar todas las variables relacionadas en una colección de variables o array.

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

Ahora, una única variable llamada dias almacena todos los valores relacionados entre sí, en este caso los días de la semana. Para definir un array, se utilizan los caracteres [y] para delimitar su comienzo y su final y se utiliza el carácter , (coma) para separar sus elementos.

Tipos de datos en Javascript

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. La única complicación, que es responsable de muchos errores cuando se empieza a programar, es que las posiciones de los elementos empiezan a contarse en el 0 y no en el 1:

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

En el ejemplo anterior, la primera instrucción quiere obtener el primer elemento del array. Para ello, se indica el nombre del array y entre corchetes la posición del elemento dentro del array. Como se ha comentado, las posiciones se empiezan a contar en el 0, por lo que el primer elemento ocupa la posición 0 y se accede a él mediante `dias[0]`.

El valor `dias[5]` hace referencia al elemento que ocupa la sexta posición dentro del array `dias`. Como las posiciones empiezan a contarse en 0, la posición 5 hace referencia al sexto elemento, en este caso, el valor `Sábado`.

Tipos de variables

Numéricas

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

Cadenas de texto

- `var mensaje = "Bienvenido a nuestro sitio web";`
- `var nombreProducto = 'Producto ABC';`
- `var texto1 = "Una frase con 'comillas simples' dentro";`
- `var texto2 = 'Una frase con "comillas dobles" dentro';`
- `var texto1 = 'Una frase con \'comillas simples\' dentro';`
- `var texto2 = "Una frase con \"comillas dobles\" dentro";`

Tipos de variables

Arrays

- `var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];`
- `var diaSeleccionado = dias[0];` // diaSeleccionado = "Lunes"
- `var otroDia = dias[5];` // otroDia = "Sábado"

Booleanos

- `var clienteRegistrado = false;`
- `var ivaIncluido = true;`

EJERCICIO

Crear un array llamado meses y que almacene el nombre de los doce meses del año. Mostrar por pantalla los doce nombres utilizando la función alert().



Solución

JAVASCRIPT



OPERADORES

Operadores matemáticos



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

Operador	Descripción
=	Asignación
+	Suma
-	Resta
++	Incremento en uno
--	Decremento en uno
*	Multiplicación
/	División entera
%	División modular

Operadores de asignación

- = Asignación. Asigna la parte de la derecha a la variable de la izquierda. A la derecha se colocan los valores finales y a la izquierda generalmente se coloca una variable donde queremos guardar el dato.
- += Asignación con suma. Realiza la suma de la parte de la derecha con la de la izquierda y guarda el resultado en la parte de la izquierda.
- = Asignación con resta
- *= Asignación de la multiplicación
- /= Asignación de la división
- %= Se obtiene el resto y se asigna

```
var numero1 = 3;  
var numero2 = 0;  
numero1 = numero2; // numero1= 0
```



Ejemplo

Operadores Matemáticos



```
var numero1 = 10;
```

```
var numero2 = 5;
```

```
resultado = numero1 / numero2; // resultado = 2
```

```
resultado = 3 + numero1; // resultado = 13
```

```
resultado = numero2 - 4; // resultado = 1
```

```
resultado = numero1 * numero 2; // resultado = 50
```

```
resultado = numero1 % numero2; // resultado = 0
```

```
numero1 = 9;
```

```
numero2 = 5;
```

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

% → operador "*módulo*", calcula el resto de la división entera de dos números

Operadores Matemáticos

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

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

Incremento y decremento

- `var numero = 5;`
- `++numero; // numero= 6`
- `--numero; // numero= 4`
- `numero1++; // numero1= 1`
- `numero1--; // numero1= -1`



Ejemplo

Operadores de cadenas

Las cadenas de caracteres, o variables de texto, también tienen sus propios operadores para realizar acciones típicas sobre cadenas. Aunque javascript sólo tiene un operador para cadenas se pueden realizar otras acciones con una serie de funciones predefinidas en el lenguaje que veremos más adelante.

+ Concatena dos cadenas, pega la segunda cadena a continuación de la primera.

```
cadena1 = "hola"
```

```
cadena2 = "mundo"
```

```
cadenaConcatenada = cadena1 + cadena2 //cadena  
concatenada vale "holamundo"
```



Ejemplo

Operadores lógicos

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

Operador	Descripción
&&	And (disyunción – y)
	Or (conjunción – o)
!	Not (negación – no)

Operadores Lógicos



Estos operadores sirven para realizar operaciones lógicas, que son aquellas que dan como resultado un verdadero o un falso, y se utilizan para tomar decisiones en nuestros scripts. En vez de trabajar con números, para realizar este tipo de operaciones se utilizan datos booleanos, que son verdadero (true) y falso (false). Los operadores lógicos relacionan los datos booleanos para dar como resultado otro dato booleano, tal como podemos ver en el siguiente ejemplo.

Si tengo hambre y tengo comida entonces me pongo a comer

Nuestro programa Javascript utilizaría en este ejemplo un dato booleano para tomar una decisión. Primero mirará si tengo hambre, si es cierto (true) mirará si dispongo de comida. Si son los dos ciertos, se puede poner a comer. En caso de que no tenga comida o que no tenga hambre no comería, al igual que si no tengo hambre ni comida. El operando en cuestión es el operando **Y**, que valdrá verdadero (true) en caso de que los dos operandos valgan verdadero.

Operadores Lógicos



AND

La operación lógica AND obtiene su resultado combinando dos valores booleanos.

El operador se indica mediante el símbolo `&&` y su resultado solamente es true si los dos operandos son true:

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

`&` =ampersand

Operadores Lógicos



OR

La operación lógica OR también combina dos valores booleanos.

El operador se indica mediante el símbolo `||` y su resultado es `true` si alguno de los dos operandos es `true`:

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

| = pipe

Operadores Lógicos

Negación:

! Operador NO o negación. Si era true pasa a false y viceversa.

- `var visible = true;`
- `alert(!visible);` // Muestra "false" y no "true"
- `var cantidad = 0;`
- `vacio = !cantidad;` // vacio = true
- `cantidad = 2;`
- `vacio = !cantidad;` // vacio = false
- `var mensaje = "";`
- `mensajeVacio = !mensaje;` // mensajeVacio = true
- `mensaje = "Bienvenido";`
- `mensajeVacio = !mensaje;` // mensajeVacio = false



Ejemplo

Ejemplo

Operadores relacionales

Los operadores relacionales definidos por JavaScript son idénticos a los que definen las matemáticas. El resultado de todos estos operadores siempre es un valor booleano.

Operador	Descripción
<	Menor que...
<=	Menor igual que...
>	Mayor que...
>=	Mayor o igual que...
==	Igual que...
!=	Distinto de...

Operadores relacionales



Sirven para realizar expresiones condicionales todo lo complejas que deseemos. Estas expresiones se utilizan para tomar decisiones en función de la comparación de varios elementos, por ejemplo si un numero es mayor que otro o si son iguales.

Nota: Por supuesto, los operadores condicionales sirven también para realizar expresiones en las que se comparan otros tipos de datos. Nada impide comparar dos cadenas, para ver si son iguales o distintas, por ejemplo. Incluso podríamos comparar booleanos.

El operador `==` se utiliza para comparar el valor de dos variables, por lo que es muy diferente del operador `=`, que se utiliza para asignar un valor a una variable:

Operadores relacionales

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

Operadores relacionales

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

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

Precedencia de los operadores

La evaluación de una sentencia de las que hemos visto en los ejemplos anteriores es bastante sencilla y fácil de interpretar, pero cuando en una sentencia entran en juego multitud de operadores distintos puede haber una confusión a la hora de interpretarla y dilucidar qué operadores son los que se ejecutan antes que otros. Para marcar unas pautas en la evaluación de las sentencias y que estas se ejecuten siempre igual y con sentido común existe la precedencia de operadores, que no es más que el orden por el que se irán ejecutando las operaciones que ellos representan. En un principio todos los operadores se evalúan de izquierda a derecha, pero existen unas normas adicionales, por las que determinados operadores se evalúan antes que otros. Muchas de estas reglas de precedencia están sacadas de las matemáticas y son comunes a otros lenguajes, las podemos ver a continuación.

() [] . Paréntesis, corchetes y el operador punto que sirve para los objetos

! - ++ -- negación, negativo e incrementos

*** / % Multiplicación división y módulo**

+ - Suma y resta

< <= > >= Operadores relacionales

== != Operadores relacionales de igualdad y desigualdad

&& || Lógicos booleanos

= += -= *= /= %= <<= >>= >>>= &= ^= != Asignación

Precedencia de los operadores

En el siguiente ejemplo podemos ver cómo las expresiones podrían llegar a ser confusas, pero con la tabla de precedencia de operadores podremos entender sin errores cuál es el orden por el que se ejecutan.

$12 * 3 + 4 - 8 / 2 \% 3$

En este caso primero se ejecutan los operadores $*$ / y $\%$, de izquierda a derecha, con lo que se realizarían estas operaciones. Primero la multiplicación y luego la división por estar más a la izquierda del módulo.

$36 + 4 - 4 \% 3$

Ahora el módulo.

$36 + 4 - 1$

Por último las sumas y las restas de izquierda a derecha.

$40 - 1$

Lo que nos da como resultado el valor siguiente.

39

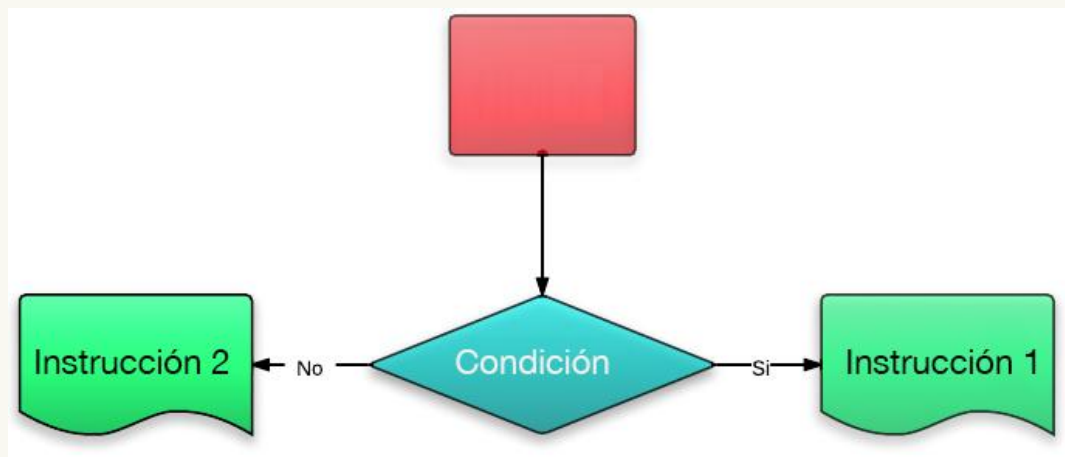
JAVASCRIPT



ESTRUCTURAS DE CONTROL DE FLUJO

Estructuras de control de flujo

Si tenemos alguna experiencia en la programación sabremos que en los programas generalmente se necesitará hacer cosas distintas dependiendo del estado de nuestras variables o realizar un mismo proceso muchas veces sin escribir las mismas líneas de código una y otra vez. Para realizar cosas más complejas en nuestros scripts se utilizan las estructuras de control. Con ellas podemos realizar tomas de decisiones.



Estructuras de control de flujo

Toma de decisiones

Nos sirven para realizar unas acciones u otras en función del estado de las variables. Es decir, tomar decisiones para ejecutar unas instrucciones u otras dependiendo de lo que esté ocurriendo en ese instante en nuestros programas.

Por ejemplo, dependiendo si el usuario que entra en nuestra página es mayor de edad o no lo es, podemos permitirle o no ver los contenidos de nuestra página.

Si edad es mayor que 18 entonces

Te dejo ver el contenido para adultos

Si no

Te mando fuera de la página

En Javascript podemos tomar decisiones utilizando dos enunciados distintos.

- IF
- SWITCH

Estructuras de control de flujo

Estructura if:

If es una estructura de control utilizada para tomar decisiones. Es un condicional que sirve para realizar unas u otras operaciones en función de una expresión. Funciona de la siguiente manera, primero se evalúa una expresión, si da resultado positivo se realizan las acciones relacionadas con el caso positivo.

La sintaxis de la estructura IF es la siguiente.

```
if (expresión a evaluar) {  
    //acciones a realizar en caso positivo  
    //...  
}
```

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



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

Las comparaciones de igualdad siempre se realizan con el operador ==, ya que el operador = solamente asigna valores:

Estructuras de control de flujo

Estructura if...else

El comando Else, es opcional y se utiliza para ejecutar las instrucciones que no cumplen la condición.

Para hacer estructuras condicionales más complejas podemos anidar sentencias IF, es decir, colocar estructuras IF dentro de otras estructuras IF. Con un solo IF podemos evaluar y realizar una acción u otra según dos posibilidades, pero si tenemos más posibilidades que evaluar debemos anidar IFs para crear el flujo de código necesario para decidir correctamente.

```
if(condicion) {...}
else { ... }
```

```
if(edad < 12)
{ alert("Todavía eres muy pequeño"); }
else if(edad < 19) { alert("Eres un adolescente"); }
else if(edad < 35) { alert("Aun sigues siendo joven"); }
else { alert("Estás viejo 😊"); }
```

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



Ejemplo

Ελεωβοιο

Estructuras de control de flujo

EJERCICIO

Escribe un programa con 2 variables, las cuales deben introducir el usuario su valor. Una vez introducidos los valores, pueden ocurrir tres cosas:

- Que el número 1 sea mayor que el número 2.
- Que el número 2 sea mayor que el número 1.
- Que los números sean iguales.

El programa debe mostrar por pantalla cual de las 3 opciones se ha cumplido.



Solución

Solución

Estructuras de control de flujo

Estructura switch

Las estructuras de control son la manera con la que se puede dominar el flujo de los programas, para hacer cosas distintas en función de los estados de las variables. SWITCH es una estructura un poco más compleja que permite hacer múltiples operaciones dependiendo del estado de una variable.

```
switch (expresión) {  
    case valor1:  
        Sentencias a ejecutar si la expresión tiene como valor a valor1  
        break  
    case valor2:  
        Sentencias a ejecutar si la expresión tiene como valor a valor2  
        break  
    case valor3:  
        Sentencias a ejecutar si la expresión tiene como valor a valor3  
        break  
    default:  
        Sentencias a ejecutar si el valor no es ninguno de los anteriores  
}
```

Estructuras de control de flujo

Estructura switch

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

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

También es opcional la opción default u opción por defecto.



Ejemplo

Ejemplo

Estructuras de control de flujo



EJERCICIO

Escribe un programa que tenga 2 funciones. Una función "Suma", que sume dos números pasados por parámetro, y otra igual pero para "Restar".

Se mostrará por pantalla a opción de:

1. Suma
2. Resta

Depende qué número elija el usuario (1 o 2), se ejecutará la función correspondiente y mostrará el resultado (obligatorio utilizar SWITCH).



Solución

Solución

Estructuras de control de flujo

Estructura for:

El bucle FOR se utiliza para repetir una o más instrucciones un determinado número de veces. De entre todos los bucles, el FOR se suele utilizar cuando sabemos seguro el número de veces que queremos que se ejecute. La sintaxis del bucle for se muestra a continuación.

```
for(inicializacion; condicion; actualizacion) {  
    //sentencias a ejecutar en cada iteración  
}
```



Ejemplo

Ejemplo

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.

Después del for se colocan las sentencias que queremos que se ejecuten en cada iteración, acotadas entre llaves.

Estructuras de control de flujo

```
var mensaje = "Hola, estoy dentro de un bucle";  
for(var i = 0; i < 5; i++)  
{ alert(mensaje); }
```

Por ejemplo si queremos escribir los números pares del 1 al 100, se escribirá el siguiente bucle.

```
for (i=1;i<=100;i+=2)  
    {document.write(i);}
```

En este caso se inicializa la variable *i* a 0. Como condición para realizar una iteración, se tiene que cumplir que la variable *i* sea menor que 5. Como actualización se incrementará en 1 la variable *i*.

El siguiente ejemplo ilustra como mostrar los valores contenidos en un array:

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado",  
            "Domingo"];  
for(var i=0; i<7; i++)  
{ alert(dias[i]); }
```

Estructuras de control de flujo



EJERCICIO 1

Crear un programa que mediante un bucle FOR, muestre por pantalla los números del 1 al 10 en líneas separadas.



Solución

Solución

EJERCICIO 2

Hacer un bucle que escriba en una página web los encabezamientos desde <H1> hasta <H6> con un texto que ponga "Encabezado de nivel x".



Solución

Solución

EJERCICIO 3

Hacer un programa que nos solicite un número y muestre por pantalla su tabla de multiplicar.



Solución

Solución

Estructuras de control de flujo



Estructura for...in

Sirve para recorrer todos los elementos que forman un array.

```
for(indice in array)
{ ... }
```

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves",
            "Viernes", "Sábado", "Domingo"];
for(i in dias)
{ alert(dias[i]); }
```

Estructuras de control de flujo

Estructura While:

Estos bucles se utilizan cuando queremos repetir la ejecución de unas sentencias un número indefinido de veces, siempre que se cumpla una condición. Sólo se indica, como veremos a continuación, la condición que se tiene que cumplir para que se realice una iteración.

```
While(condicion) {  
    //sentencias a ejecutar en cada iteración  
}
```

```
var color = ""  
while (color != "rojo"){  
    color = prompt("dame un color (escribe rojo para salir)","")  
}
```

Lo que hace es pedir que el usuario introduzca un color y lo hace repetidas veces, mientras que el color introducido no sea rojo. Para ejecutar un bucle como éste, primero tenemos que inicializar la variable que vamos utilizar en la condición de iteración del bucle.

Estructuras de control de flujo

Estructura Do ... While:

Es una variación del bucle while visto anteriormente. Se utiliza generalmente cuando no sabemos cuantas veces se habrá de ejecutar el bucle, igual que el bucle WHILE, con la diferencia de que sabemos seguro que el bucle por lo menos se ejecutará una vez.

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

```
var color  
do {  
    color = prompt("dame un color (escribe rojo para salir)","")  
} while (color != "rojo")
```

Este ejemplo funciona exactamente igual que el anterior, excepto que no tuvimos que inicializar la variable color antes de introducirnos en el bucle. Pide un color mientras que el color introducido es distinto que "rojo".

Estructuras de control de flujo

Ejemplo

Vamos a ver a continuación un ejemplo más práctico sobre cómo trabajar con un bucle WHILE. Como resulta muy difícil hacer ejemplos prácticos con lo que sabemos de Javascript, vamos a adelantar una instrucción que aun no conocemos.

En este ejemplo vamos a declarar una variable e inicializarla a 0. Luego iremos sumando a esa variable un número aleatorio del 1 al 100 hasta que sumemos 1.000 o más, imprimiendo el valor de la variable suma después de cada operación. Será necesario utilizar el bucle WHILE porque no sabemos exactamente el número de iteraciones que tendremos que realizar (dependerá de los valores aleatorios que se vayan obteniendo).

```
var suma = 0
while (suma < 1000){
    suma += parseInt(Math.random() * 100)
    document.write (suma + "<br>")
}
```



JAVASCRIPT



FUNCIONES

Funciones



¿Qué es una función?

A la hora de hacer un programa ligeramente grande existen determinados procesos que se pueden concebir de forma independiente, y que son más sencillos de resolver que el problema entero. Además, estos suelen ser realizados repetidas veces a lo largo de la ejecución del programa. Estos procesos se pueden agrupar en una función, definida para que no tengamos que repetir una y otra vez ese código en nuestros scripts, sino que simplemente llamamos a la función y ella se encarga de hacer todo lo que debe.

Así que podemos ver una función como una serie de instrucciones que englobamos dentro de un mismo proceso. Este proceso se podrá luego ejecutar desde cualquier otro sitio con solo llamarlo. Por ejemplo, en una página web puede haber una función para cambiar el color del fondo y desde cualquier punto de la página podríamos llamarla para que nos cambie el color cuando lo deseemos.

Qué es una función?



Las funciones se utilizan constantemente, no sólo las que escribes tú, sino también las que ya están definidas en el sistema, pues todos los lenguajes de programación suelen tener un montón de funciones para realizar procesos habituales, como por ejemplo obtener la hora, imprimir un mensaje en la pantalla o convertir variables de un tipo a otro. Ya hemos visto alguna función en nuestros sencillos ejemplos anteriores. Por ejemplo, cuando hacíamos un `document.write()` en realidad estábamos llamando a la función `write()` asociada al documento de la página, que escribe un texto en la página.

Cómo se escribe una función?

Una función se debe definir con una sintaxis especial que vamos a conocer a continuación.

```
function nombrefuncion (){  
    instrucciones de la función  
    ...  
}
```

Primero se escribe la palabra **function**, reservada para este uso. **Seguidamente se escribe el nombre de la función**, que como los nombres de variables puede tener números, letras y algún carácter adicional como en guión bajo. A continuación se colocan entre llaves las distintas instrucciones de la función. Las llaves en el caso de las funciones no son opcionales, además es útil colocarlas siempre como se ve en el ejemplo, para que se reconozca fácilmente la estructura de instrucciones que engloba la función.

Cómo llamar a una función?

Para ejecutar una función la tenemos que invocar en cualquier parte de la página. Con eso conseguiremos que se ejecuten todas las instrucciones que tiene la función entre las dos llaves.

Para ejecutar la función utilizamos su nombre seguido de los paréntesis. Por ejemplo, así llamaríamos a la función `escribirBienvenida()` que acabamos de crear.

En principio, podemos colocar las funciones en cualquier parte de la página, siempre entre etiquetas `<SCRIPT>`, claro está. No obstante existe una limitación a la hora de colocarla con relación a los lugares desde donde se la llame. Te adelantamos que lo más fácil es describir la función antes de cualquier llamada a la misma y así seguro que nunca nos equivocaremos.

¿Dónde colocamos las funciones?

Existen dos opciones posibles para colocar el código de una función:

a) Colocar la función en el mismo bloque de script: En concreto, la función se puede definir en el bloque **<SCRIPT>** donde esté la llamada a la función, aunque es indiferente si la llamada se encuentra antes o después del código de la función, dentro del mismo bloque **<SCRIPT>**.

```
<SCRIPT>
    miFuncion()
    function miFuncion(){
        //hago algo...
        document.write("Esto va bien")
    }
</SCRIPT>
```

Este ejemplo funciona correctamente porque la función está declarada en el mismo bloque que su llamada.

¿Dónde colocamos las funciones?



b) Colocar la función en otro bloque de script: También es válido que la función se encuentre en un bloque `<SCRIPT>` anterior al bloque donde está la llamada.

```
<HTML>
<HEAD>
<TITLE>MI PÁGINA</TITLE>
<SCRIPT>
    function miFuncion(){
        //hago algo...
        document.write("Hola")
    }
</SCRIPT>
</HEAD>
<BODY>
    <SCRIPT>
        miFuncion()
    </SCRIPT>
</BODY>
</HTML>
```

Vemos un código completo sobre cómo podría ser una página web donde tenemos funciones Javascript. Como se puede comprobar, las funciones están en la cabecera de la página (dentro del HEAD). Éste es un lugar excelente donde colocarlas, porque se supone que en la cabecera no se van a utilizar todavía y siempre podremos disfrutar de ellas en el cuerpo porque sabemos seguro que ya han sido declaradas.

Parámetros en las funciones

Los parámetros se usan para mandar valores a las funciones. Una función trabajará con los parámetros para realizar las acciones. Por decirlo de otra manera, los parámetros son los valores de entrada que recibe una función.

```
function escribirBienvenida(nombre){  
    document.write("<H1>Hola " + nombre + "</H1>")  
}
```

Como podemos ver en el ejemplo, para definir en la función un parámetro tenemos que poner el nombre de la variable que va a almacenar el dato que le pasemos. Esa variable, que en este caso se llama nombre, tendrá como valor el dato que le pasemos a la función cuando la llamemos. Además, la variable donde recibimos el parámetro tendrá vida durante la ejecución de la función y dejará de existir cuando la función termine su ejecución.

Para llamar a una función que tiene parámetros se coloca entre paréntesis el valor del parámetro. Para llamar a la función del ejemplo habría que escribir:

```
escribirBienvenida("Alberto García")
```

Múltiples parámetros



Una función puede recibir tantos parámetros como queramos y para expresarlo se colocan los nombres de los parámetros separados por comas, dentro de los paréntesis. Veamos rápidamente la sintaxis para que la función de antes, pero hecha para que reciba dos parámetros, el primero el nombre al que saludar y el segundo el color del texto.

```
function escribirBienvenida(nombre,colorTexto){  
    document.write("<FONT color=" + colorTexto + ">")  
    document.write("<H1>Hola " + nombre + "</H1>")  
    document.write("</FONT>")  
}
```

Llamaríamos a la función con esta sintaxis. Entre los paréntesis colocaremos los valores de los parámetros.

```
escribirBienvenida("Pepe", "red")
```



Los parámetros se pasan por valor

Esto quiere decir que estamos pasando valores y no variables. En la práctica, aunque modifiquemos un parámetro en una función, la variable original que habíamos pasado no cambiará su valor. Se puede ver fácilmente con un ejemplo.

```
function pasoPorValor(miParametro){  
  miParametro = 32  
  document.write("he cambiado el valor a 32")  
}
```

```
var miVariable = 5  
pasoPorValor(miVariable)  
document.write ("el valor de la variable es: " + miVariable)
```

Devolución de valores en las funciones

Las funciones en Javascript también pueden retornar valores. De hecho, ésta es una de las utilidades más esenciales de las funciones, que debemos conocer, no sólo en Javascript sino en general en cualquier lenguaje de programación. De modo que, al invocar una función, se podrá realizar acciones y ofrecer un valor como salida.

Veamos un ejemplo de función que calcula la media de dos números. La función recibirá los dos números y retornará el valor de la media.

```
function calculaMedia(valor1,valor2){  
    var resultado  
    resultado = (valor1 + valor2) / 2  
    return resultado  
}
```

Para especificar el valor que retornará la función se utiliza la palabra **return** seguida de el valor que se desea devolver. En este caso se devuelve el contenido de la variable resultado, que contiene la media calculada de los dos números.

Devolución de valores en las funciones

Quizás nos preguntemos ahora cómo recibir un dato que devuelve una función. Realmente en el código fuente de nuestros programas podemos invocar a las funciones en el lugar que deseemos. Cuando una función devuelve un valor simplemente se sustituye la llamada a la función por ese valor que devuelve. Así pues, para almacenar un valor de devolución de una función, tenemos que asignar la llamada a esa función como contenido en una variable, y eso lo haríamos con el operador de asignación =.

Para ilustrar esto se puede ver este ejemplo, que llamará a la función `media()` y guardará el resultado de la media en una variable para luego imprimirla en la página.

```
var miMedia  
miMedia = calculaMedia(12,8)  
document.write (miMedia)
```

Múltiples return



En realidad en Javascript las funciones sólo pueden devolver un valor, por lo que en principio no podemos hacer funciones que devuelvan dos datos distintos.

Ahora bien, aunque sólo podamos devolver un dato, en una misma función podemos colocar más de un return. Como decimos, sólo vamos a poder retornar una cosa, pero dependiendo de lo que haya sucedido en la función podrá ser de un tipo u otro, con unos datos u otros.

En esta función podemos ver un ejemplo de utilización de múltiples return. Se trata de una función que nos indica si el número es par o impar.

```
function multipleReturn(numero){  
    var resto = numero % 2;  
    if (resto == 0) {  
        valor='par';  
        return valor;  
    } else {  
        valor='impar'  
        return valor;  
    }  
}
```



Ejemplo 1

ΕΙΣΑΓΩΓΗ 1



Ejemplo 2

ΕΙΣΑΓΩΓΗ 2

Para averiguar si un número es par hallamos el resto de la división al dividirlo entre 2. Si el resto es cero es que era par y devolvemos un 0, en caso contrario, el número es impar.

Ámbito de las variables en funciones

Dentro de las funciones podemos declarar variables. Sobre este asunto debemos de saber que todas las variables declaradas en una función son locales a esa función, es decir, sólo tendrán validez durante la ejecución de la función.

Podría darse el caso de que podemos declarar variables en funciones que tengan el mismo nombre que una variable global a la página. Entonces, dentro de la función, la variable que tendrá validez es la variable local y fuera de la función tendrá validez la variable global a la página.

En cambio, si no declaramos las variables en las funciones se entenderá por javascript que estamos haciendo referencia a una variable global a la página, de modo que si no está creada la variable la crea, pero siempre global a la página en lugar de local a la función.

Ámbito de las variables en funciones

Veamos el siguiente código:

```
function variables_globales_y_locales(){  
    var variableLocal = 23  
    variableGlobal = "qwerty"  
}
```

En este caso `variableLocal` es una variable que se ha declarado en la función, por lo que será local a la función y sólo tendrá validez durante su ejecución. Por otra parte `variableGlobal` no se ha llegado a declarar (porque antes de usarla no se ha utilizado la palabra `var` para declararla). En este caso la variable `variableGlobal` es global a toda la página y seguirá existiendo aunque la función finalice su ejecución. Además, si antes de llamar a la función existiese la variable `variableGlobal`, como resultado de la ejecución de esta función, se machacaría un hipotético valor de esa variable y se sustituiría por "qwerty".

Funciones y propiedades básicas para cadenas de texto

length

Devuelve el número de caracteres de la cadena

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.length; // resultado= 10
```

charAt(indice)

Devuelve el carácter que hay en la posición indicada como índice. Las posiciones de un string empiezan en 0.

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.charAt(5); // resultado= M
```



Ejemplo
Example

Funciones y propiedades básicas para cadenas de texto

indexOf(carácter, desde)

Devuelve la posición de la primera vez que aparece el carácter indicado por parámetro en un string. Si no encuentra el carácter en el string devuelve -1. El segundo parámetro es opcional y sirve para indicar a partir de que posición se desea que empiece la búsqueda.

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.indexOf('Mundo',0); // resultado= 5
```

lastIndexOf(carácter, desde)

Busca la posición de un carácter exactamente igual a como lo hace la función indexOf pero desde el final en lugar del principio.

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.lastIndexOf('Mundo'); // resultado= 5
```

Funciones y propiedades básicas para cadenas de texto

split(separador)

Sirve para crear un array a partir de un String en el que cada elemento es la parte del String que está separada por el separador indicado por parámetro.

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.split(' '); // resultado= ['Hola','Mundo']  
var resultado= mensaje.split(""); // resultado= ['H','o','l','a',' ','M','u','n','d','o']
```

substring(inicio,fin)

Devuelve el substring que empieza en el carácter de inicio y termina en el carácter de fin. Si intercambiamos los parámetros de inicio y fin también funciona. Simplemente nos da el substring que hay entre el carácter menor y el mayor.

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.substring(5,8); // resultado= Mun
```

Funciones y propiedades básicas para cadenas de texto

toLowerCase()

Pone todas los caracteres de un string en minúsculas.

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.toLowerCase(); // resultado= hola mundo
```

toUpperCase()

Pone todas los caracteres de un string en mayúsculas.

```
var mensaje = "Hola Mundo";  
var resultado= mensaje.toUpperCase(); // resultado= HOLA MUNDO
```

Funciones y propiedades básicas para cadenas de texto

EJERCICIO

Crear una función que nos permita separar en caracteres una frase que introduzcamos por el símbolo que le indiquemos.



Solución

Solución

H-o-l-a- -M-u-n-d-o

EJERCICIO

Hacer un script que rompa un string en dos mitades y las imprima por pantalla en dos líneas diferentes.



Solución

Solución

Hola
Mundo

Funciones y métodos básicos para números

parseInt(cadena,base)

Recibe una cadena y una base. Devuelve un valor numérico entero resultante de convertir la cadena en un número en la base indicada.

```
document.write (parseInt("34")); // resultado= 34  
document.write (parseInt("3.34")); // resultado= 3  
document.write (parseInt("Hola")); // resultado= NaN (Not a Number)
```

isNaN(número)

Devuelve un booleano dependiendo de lo que recibe por parámetro. Si no es un número devuelve un true, si es un número devuelve false.

```
numero= parseInt("3.6");  
isNaN(numero); // resultado= false
```

Funciones y métodos básicos para números

parseFloat(cadena)

Recibe una cadena y la convierte a un número con decimales.

```
miFloat = parseFloat("4.7");  
isNaN(miFloat); // devuelve false, porque miFloat es un número = 4.7
```

eval(cadena)

Esta función recibe una cadena de caracteres y la ejecuta como si fuera una sentencia de Javascript.

```
var miTexto = "3 + 5";  
eval("document.write(" + miTexto + ")"); // resultado = 8
```



Ejemplo

ΕΙεωbjo

Funciones y métodos básicos para números

toFixed(número)

Devuelve una representación de cadena de un número en notación de punto fijo, que contiene los decimales indicados en el número.

```
numero= 13.8567;  
numero.toFixed(3); // resultado = 13.856
```



Ejemplo

Otras funciones Matemáticas: Math

Math es un objeto incorporado por JavaScript que posee propiedades y métodos creados por constantes y funciones matemáticas.



ENLACE

Funciones y métodos básicos para números y cadenas de texto

EJERCICIO

Realiza un programa con 2 funciones. El programa principal hará una llamada a cada una de las funciones, para comprobar que funcionan correctamente.

Las funciones son las siguientes:

1- Una función llamada "raiz" que reciba un número por parámetro. Esta función realizará la raíz cuadrada de ese número, y devolverá el valor.

2- Una función llamada "letra", que reciba 2 parámetros, una cadena de caracteres, y un nº (índice). La función devolverá la letra cuyo índice ocupe en la cadena, pero en mayúsculas. Por ejemplo, si tengo la cadena "Hola", y llamo a la función, le paso la cadena "Hola" y de índice el número 3, el programa devolverá una "A"... O sea, la letra que ocupa la posición del índice en la cadena.



Solución

solucion

Funciones y métodos básicos para Fechas



Date ()

Si no proporciona argumentos, el constructor crea un objeto Date con la hora y fecha de hoy según la hora local.

Si proporciona algunos argumentos, debe proporcionar al menos 2 argumentos.

Los argumentos vacíos se establecen a 0 (ó 1 si falta el día).

new Date()

new Date(milisegundos)

new Date(cadenaFecha)

new Date(año_num, mes_num, dia_num [, hor_num, min_num, seg_num, mils_num])

Continúa...

Funciones y métodos básicos para Fechas

Date ()

Parámetros

Milisegundos: Valor entero que representa el número de milisegundos desde las 00:00:00 UTC del 1 de enero de 1970.

cadenaFecha: Valor de tipo cadena que representa una fecha. La cadena debería estar en un formato reconocido por el método Date.parse().

año num, mes num, día num: Valores enteros con las representaciones de las partes de una fecha. Como valor entero, el mes se representa de 0 a 11, con 0=enero y 11=diciembre.

hor num, min num, seg num, mils num: Valores enteros que representan las partes de una hora completa.

```
hoy = new Date();  
cumpleanos = new Date("December 17, 1995 03:24:00");  
cumpleanos = new Date(1995,11,17);  
cumpleanos = new Date(1995,11,17,3,24,0);
```

Funciones y métodos básicos para Fechas

Date.UTC()

UTC toma los parámetros de la fecha delimitados por comas y devuelve el número de milisegundos entre las 00:00:00 del 1 de enero de 1970 (hora universal) y la hora que especifique.

Date.UTC(año, mes[, día[, hora[, minutos[, segundos[, milisegundos]]]])

Parámetros

Año: Un año mayor de 1900.

Mes: Un entero entre 0 y 11 que representa al mes.

Día: Un entero entre 1 y 31 que representa al día del mes.

Hora: Un entero entre 0 y 23 que representa la hora.

Minutos: Un entero entre 0 y 59 que representa los minutos.

Segundos: Un entero entre 0 y 59 que representa los segundos.

Milisegundos: Un entero entre 0 y 999 que representa los milisegundos.

```
fechaGmt = new Date(Date.UTC(2017, 11, 1, 0, 0, 0));
```

Funciones y métodos básicos para fechas

getDate()

El método getDate() devuelve el día del mes para la fecha especificada.

```
var Xmas95 = new Date('December 25, 1995 23:15:30');  
var day = Xmas95.getDate(); // 25
```

getDay()

El valor devuelto por getDay() es un entero correspondiente al día de la semana; siendo 0 (Domingo) el primer día, 1 (Lunes) el segundo, etcétera.

```
var day = Xmas95.getDay(); // 1
```

getFullYear()

El método getFullYear() devuelve el año del objeto de fecha especificado.

```
var today = new Date();  
var year = today.getFullYear(); // año actual
```

Funciones y métodos básicos para Fechas

getMonth()

Devuelve un entero entre 0 y 11, donde 0 corresponde a Enero, 1 a Febrero y así sucesivamente.

```
var Xmas95 = new Date('December 25, 1995 23:15:30');  
var day = Xmas95.getMonth(); // 11
```



Ejemplo

ΕΙ6ωb1o

Otras funciones de Fecha: Date

Date es un objeto incorporado por JavaScript que permite trabajar con fechas y horas. Para más información consultar el enlace...



ENLACE

ΕΙΓΛCE

Funciones y métodos básicos para fechas



EJERCICIO

Realiza un programa que calcule el día de la semana de una fecha.



Solución

Solución

EJERCICIO

Realiza un programa que nos pida dos fechas y calcule el número de días transcurridos entre ellas.



Solución

Solución

Funciones y métodos básicos para arrays

new Array()

El primer paso para utilizar un array es crearlo. Esta es la sentencia para crear un objeto array:

```
var miArray = new Array(10); // el array tendrá 10 posiciones
```

Los arrays en Javascript empiezan siempre en la posición 0, así que un array que tenga por ejemplo 10 posiciones, tendrá casillas de la 0 a la 9. Para recoger datos de un array lo hacemos igual: poniendo entre corchetes el índice de la posición a la que queremos acceder. Veamos cómo se imprimiría en la pantalla el contenido de un array.



Ejemplo

El ejemplo

Funciones y métodos básicos para arrays

length

Todos los arrays en javascript, aparte de almacenar el valor de cada una de sus casillas, también almacenan el número de posiciones que tienen. Para ello utilizan una propiedad del objeto array, la propiedad length. Almacena un número igual al número de casillas que tiene el array.

```
var vocales = ["a", "e", "i", "o", "u"];  
var numeroVocales = vocales.length; // numeroVocales = 5
```

join(separador)



Ejemplo

El método join() une todos los elementos de un array en una cadena. El separador es una cadena usada para separar cada uno de los elementos del array. Si se omite, los elementos son separados por una coma. Si el separador es una cadena vacía todos los elementos quedan pegados.

```
var a = ['Viento', 'Lluvia', 'Fuego'];  
var miVar1 = a.join(); // asigna 'Viento,Lluvia,Fuego' a miVar1
```

Funciones y métodos básicos para arrays

pop()

El método pop elimina el último elemento de un array y devuelve su valor al método que lo llamó.

```
var myFish = ['angel', 'clown', 'mandarin', 'sturgeon'];  
console.log(myFish); // ['angel', 'clown', 'mandarin', 'sturgeon']  
var popped = myFish.pop();  
console.log(myFish); // ['angel', 'clown', 'mandarin']  
console.log(popped); // 'sturgeon'
```

push()

El método push() agrega uno o más elementos al final de un array y devuelve la nueva longitud del array.

```
var deportes = ["soccer", "baseball"];  
deportes.push("football", "swimming");
```

Funciones y métodos básicos para arrays

shift()

El método shift() elimina el primer elemento del array y devuelve dicho elemento. Este método modifica la longitud del array.

```
var array = [1, 2, 3];  
var primero = array.shift(); // ahora array = [2, 3], primero = 1
```

unshift()

El método unshift() agrega uno o más elementos al inicio del array, y devuelve la nueva longitud del array.

```
var array = [1, 2, 3];  
array.unshift(0); // ahora array = [0, 1, 2, 3]
```

reverse()

El método reverse() coloca al revés (inversamente) un array. El primer elemento pasa a ser el último y el último pasa a ser el primero.

```
var array = [1, 2, 3];  
array.reverse(); // ahora array = [3, 2, 1]
```

Funciones y métodos básicos para arrays

Existen **arrays multidimensionales**, que son un estructuras de datos que almacenan los valores en más de una dimensión. Los arrays que hemos visto hasta ahora almacenan valores en una dimensión, por eso para acceder a las posiciones utilizamos tan solo un índice. Los arrays de 2 dimensiones guardan sus valores, por decirlo de alguna manera, en filas y columnas y por ello necesitaremos dos índices para acceder a cada una de sus posiciones. Los elementos del array son a su vez otros arrays.

```
var arrayMuchasDimensiones = [1, ["hola", "que", "tal"], ["si", "no"], 2, 5];  
alert (arrayMuchasDimensiones[1][3]);
```

EJERCICIO

Realizar un programa que nos cree una tabla que contenga los siguientes datos en un array multidimensional y nos la muestre por pantalla:

Ciudad 0	12	10	11
Ciudad 1	5	0	2
Ciudad 2	10	8	10



Solución

Solución