



UF1305: PROGRAMACIÓN CON LENGUAJES DE GUIÓN EN PÁGINAS WEB

Manual teórico.

INDICE

| | |
|---|----|
| 1. Metodología de la programación. | 4 |
| 1.1. Introducción. | 4 |
| 1.2. Lógica de programación | 6 |
| 1.3. Algoritmo y Pseudocódigo..... | 6 |
| 1.4. Representación gráfico-esquemática de programas. Diagramas de flujo..... | 12 |
| 2. Lenguaje JavaScript | 15 |
| 2.1. ¿Qué es JavaScript?. | 15 |
| 2.2. ¿Dónde y cómo insertar el código Javascript en las páginas web? | 16 |
| 2.3. Impedir que Javascript haga más lentas las páginas..... | 17 |
| 2.4. Funciones usadas para escribir con Javascript..... | 18 |
| 2.5. Como usar la función document.write, ejemplos prácticos..... | 18 |
| 3. Elementos básicos del Lenguaje JavaScript | 24 |
| 3.1. Variables en JavaScript | 24 |
| 3.1.1. Tipos de variables. | 24 |
| 3.1.2. Reglas para nombrar variables. | 24 |
| 3.2. Tipos de datos en JavaScript..... | 25 |
| 3.3. Declarando (creando) variables en JavaScript. | 27 |
| 3.4. Entrada de datos por teclado. | 32 |
| 3.5. Introducción a algunas funciones en JavaScript | 35 |
| 3.5.1. Función alert. | 35 |
| 3.5.2. Función write | 36 |
| 3.5.3. Función prompt | 37 |
| 3.5.4. Método print del objeto window..... | 38 |
| 3.5.5. Abrir ventana en JavaScript. Método open del objeto window..... | 38 |
| 3.6. Operadores en JavaScript | 41 |
| 3.6.1. Asignación. | 41 |
| 3.6.2. Incremento y decremento | 42 |
| 3.6.3. Lógicos..... | 44 |
| 3.6.4. Matemáticos..... | 47 |
| 3.6.5. Relacionales..... | 49 |
| 3.7. Tipos de estructuras de control en JavaScript | 51 |
| 3.7.1. Estructura de control javascript condicional (IF ELSE) | 52 |
| 3.7.2. IF ELSE IF ANIDADAS | 56 |

| | | |
|-----------|--|----|
| 3.7.3. | Estructura de control javascript condicional Switch. | 58 |
| 3.8. | Bucles | 67 |
| 3.8.1. | for | 68 |
| 3.8.2. | while | 68 |
| 3.8.3. | Control de bucle | 68 |
| 3.9. | Objetos | 68 |
| 3.9.1. | Jerarquía de Objetos..... | 69 |
| 3.9.2. | Objeto Window: | 69 |
| 3.9.3. | Objeto Document | 70 |
| 3.9.4. | Objeto String..... | 71 |
| 3.9.5. | Objeto Math | 72 |
| 3.9.6. | Objeto Date | 72 |
| 3.9.7. | Objeto Array | 73 |
| 3.9.8. | Objeto Form | 74 |
| 3.9.9. | Objeto History | 75 |
| 3.9.10. | Objeto Number..... | 75 |
| 3.9.11. | Objeto Boolean..... | 76 |
| 3.9.12. | Último nivel de Objetos | 76 |
| 3.9.12.1. | Text-Password | 76 |
| 3.9.12.2. | Text Area | 77 |
| 3.9.12.3. | Button | 77 |
| 3.9.12.4. | Check | 78 |
| 3.9.12.5. | Radio..... | 78 |
| 3.9.12.6. | Select | 78 |
| 3.9.12.7. | Reset..... | 79 |
| 3.9.12.8. | Submit | 79 |
| 3.10. | Algo más en JAVASCRIPT... .. | 80 |
| 3.10.1. | Eventos En JavaScript | 80 |
| 3.10.2. | Métodos Globales..... | 80 |
| 3.11. | Correspondencia entre propiedades CSS y propiedades DOM..... | 81 |
| 3.12. | Palabras reservadas | 83 |
| 3.13. | Manipulacion de objetos | 83 |
| 3.14. | Enlaces sobre JavaScript y JQuery. | 84 |

1. Metodología de la programación.

1.1. Introducción.

Los ordenadores no son más que un conjunto de componentes electrónicos o hardware, que están a la espera de recibir órdenes para realizar una determinada tarea. La forma de proporcionarle esas tareas son los programas o el software.

Un programa es un conjunto de instrucciones que le indican al ordenador como llevar a cabo una tarea.

El lenguaje que interpretan los ordenadores es el lenguaje binario, 1 y ceros, que determinan la existencia o no de impulsos o señales eléctricas y electromagnéticas. Por lo tanto los ordenadores interpretan y son capaces de ejecutar directamente ese código binario formado por unos y ceros.

¿La existencia del código binario querrá decir que para realizar un programa deberíamos escribir una sucesión interminable de dichos unos y ceros que le indiquen al ordenador que tarea realizar?.

No, con esa necesidad, la de no programar directamente en código binario, surgen los lenguajes de programación.

Un lenguaje de programación nos proporciona una serie de instrucciones y estructuras con una sintaxis muy específica que nos permiten detallar, al ordenador, las operaciones que debe realizar de forma sencilla.

En función del objetivo, de la forma de trabajar y de muchos otros factores, existen diferentes tipos de lenguajes de programación:

- Lenguajes de Alto-Bajo nivel
- Lenguajes Interpretados o Compilados
- Lenguajes clásicos, visuales y de Internet
- Por el objetivo

- **Lenguajes de Alto-Bajo nivel**

El nivel de un lenguaje hace referencia a su proximidad al lenguaje natural, considerándose de más nivel cuanto más cercanos están a este y de menos nivel cuando más cerca están del lenguaje máquina

- ✓ El lenguaje de más bajo nivel o lenguaje máquina es el que utiliza el ordenador, el que la máquina entiende, basado en un sistema de 0 y 1.

Son difíciles de aprender y manejar, ya que no resultan cercanos al ser humanos, pero son rápidos ya que evitan las traducciones intermedias. Fueron los primeros en aparecer.

- ✓ Los lenguajes de **alto nivel** son más fáciles de aprender y permiten despreocuparse de la arquitectura del ordenador. Ejemplos son: BASIC, PASCAL, FORTRAN, C (aunque este es intermedio)...

- **Lenguajes Interpretados o Compilados**

Los Lenguajes de Programación deben traducirse (excepto el código máquina) para que sean interpretables (o inteligibles) por el ordenador. Esta traducción puede hacerse mediante:

- ✓ Los **Lenguajes interpretados**, se encargan de realizar la traducción instrucción a instrucción a la vez que se ejecuta el programa. Son más lentos, pero mejores cuando el proceso de traducción/ejecución se realiza en repetidas ocasiones, por lo que son más adecuados para principiantes.
- ✓ Los **Lenguajes compilados** traducen el programa entero y luego lo montan generando un programa ejecutable por sí sólo. Una vez compilado el programa, el compilador no tiene por qué estar presente, pudiéndose transportar el programa ejecutable a cualquier ordenador, sin necesidad de manejar el compilador.

- **Lenguajes clásicos, visuales y de Internet**

- ✓ Los **Lenguajes clásicos** están basados en un lenguaje en el que se escribe el código necesario para realizar las operaciones que se requieren (posteriormente será traducido o compilado, generando un programa ejecutable). Los más conocidos son el BASIC, el PASCAL, el C, el COBOL y el CLIPPER
- ✓ Los **Lenguajes visuales** son más avanzados y están basados en objetos. Cada entidad del programa (eventos, acciones..) es un objeto sobre el que se definen operaciones. Estos permiten almacenar los objetos (con todo su código) en una serie de librerías. Son lenguajes muy intuitivos que sustituyen las líneas de código por directas representaciones gráficas. P.ej.: Visual Basic
- ✓ Los **Lenguajes de Internet** son lenguajes específicos diseñados para la creación de páginas Web y realizar su programación (motores de búsqueda, seguridad, establecimiento de comunicaciones...). Son la última generación de lenguajes. Existen distintos tipos dependiendo del grado de especialización. P ej.: JAVA, HTML

- **Por el objetivo**

Los programas pueden clasificarse por el objetivo para el que fueron creados:

- ✓ **BASIC, PASCAL:** aprendizaje de programación
- ✓ **C y C++:** programación de sistemas
- ✓ **COBOL, RPG, Natural:** gestión de empresas
- ✓ **FORTRAN:** cálculo numérico
- ✓ **CLIPPER, ACCESS, Dbase, Delphi, SQL:** bases de datos
- ✓ **Visual BASIC, Visual C:** programación en Windows
- ✓ **HTLM, JAVA, PERL:** Internet (páginas Web)
- ✓ **Lingo:** programas multimedia
- ✓ **Prolog, Lisp:** Inteligencia Artificial

1.2. Lógica de programación

En un programa, todas las instrucciones u operaciones hay que secuenciarlas en el orden correcto y organizarlas de forma que produzcan el resultado esperado. Esta serie de pasos ordenados con la fórmula para resolver el problema se denomina algoritmo, que luego será plasmado en un lenguaje de programación determinado.

1.3. Algoritmo y Pseudocódigo

El término **algoritmo** parece derivar del nombre de un matemático árabe llamado Mohamed ibn Musa al – Khuwarizmi, que vivió en Bagdad alrededor del año 830 de nuestra era, y que escribió un libro que contenía un sistema de numeración decimal y reglas de cálculo. Dicho libro sería utilizado con posterioridad impulsando la sustitución del uso del ábaco.

En el campo de la informática y la programación, se ha adoptado el término para describir cualquier serie de instrucciones precisas que dan lugar a un resultado. La relación con las matemáticas sigue siendo estrecha, aunque ha ganado peso la concepción como serie de instrucciones precisas, que no necesariamente implican cálculo. Así, podríamos hacernos una primera idea de qué es un algoritmo pensando en las instrucciones para montar un mueble desarmado, o en la preparación de un plato a partir de una receta de cocina. Veamos lo que sería un algoritmo para freír un huevo:

1. Inicio.
2. Poner a calentar aceite en una sartén.
3. Cuando el aceite humee ligeramente, romper el huevo y verterlo en el aceite.
4. Esperar que se solidifique el huevo.
5. Retirar el huevo del aceite, dejar que escurra y ponerlo en un plato. Apagar el fuego.
6. Fin.

Hemos construido el algoritmo con algunas características como:

- ✓ Se indica un inicio y un fin. No es estrictamente necesario, pero si tenemos muchos algoritmos, uno detrás de otro, nos servirá de ayuda para identificarlos.
- ✓ Se ha subdividido el proceso en pasos. De momento, qué abarca el paso es criterio del autor del algoritmo.
- ✓ Se numeran los pasos. Tampoco es estrictamente necesario, pero nos será útil al menos hasta que nos acostumbremos a escribir y leer algoritmos sin numeración.
- ✓ Se ha tratado de precisar todos y cada uno de los pasos, definiendo lo mejor posible cada uno de ellos.

EJEMPLO

Realizar un algoritmo para cruzar una calle.

SOLUCIÓN

1. Inicio.
2. Buscar un paso de peatones.
3. Mirar a la derecha y comprobar que no vienen coches.
4. Mirar a la izquierda y comprobar que no vienen coches.
5. Comprobar que no hay obstáculos ni huecos para llegar al extremo opuesto.
6. Si hay un semáforo y está en rojo volver al punto 3.
7. Cruzar.
8. Fin.

En relación con este caso destacaremos lo siguiente:

- ✓ Obtener el algoritmo ha sido un proceso creativo: dos personas pueden pensar en distintas formas de cruzar una calle, en distintas formas de organizar los pasos o en distintas posibilidades (por ejemplo, que haya un guardia regulando el tráfico o que no haya paso de peatones). Para programar nos apoyaremos en la creación previa de algoritmos. Por tanto, programar implica creatividad.

- ✓ Se busca la economía: que las instrucciones sean las mínimas posibles, que haya el menor número de repeticiones. “Lo sencillo es bello”.
- ✓ Se busca la eficiencia: obtener el resultado deseado empleando poco tiempo y pocos recursos y estando preparados para resolver situaciones imprevistas.
- ✓ Pueden realizarse “saltos” entre los distintos pasos, pero el orden es supremo y nunca se pierde. Ramificaciones y saltos nunca obedecen al azar: siguen un orden. Un camino siempre se puede repetir si las condiciones iniciales son las mismas.

La idea de algoritmo está profundamente imbricada con la idea de programa informático. A veces diríamos que se confunden. Las normas aplicables a los algoritmos son normas aplicables a los programas.

**Aprender a desarrollar algoritmos eficientes es
aprender a programar**

Volvamos sobre los conceptos de economía y eficiencia a través de un ejemplo. Trataremos de desarrollar un algoritmo para poner platos y cubiertos en una mesa de tres comensales.

Algoritmo “Poner mesa”. Versión 1.

1. Inicio.
2. Colocar primer plato delante de una silla.
3. Colocar segundo plato delante de una silla.
4. Colocar tercer plato delante de una silla.
5. Colocar tenedor a la izquierda del primer plato.
6. Colocar tenedor a la izquierda del segundo plato.
7. Colocar tenedor a la izquierda del tercer plato.
8. Colocar cuchara a la derecha del primer plato.

9. Colocar cuchara a la derecha del segundo plato.
10. Colocar cuchara a la derecha del tercer plato.
11. Colocar cuchillo a la derecha de la primera cuchara.
12. Colocar cuchillo a la derecha de la segunda cuchara.
13. Colocar cuchillo a la derecha de la tercera cuchara.
14. Fin.

Algoritmo "Poner mesa". Versión 2.

1. Inicio.
2. Colocar tres platos, cada uno delante de una silla.
3. Colocar tres tenedores, cada uno a la izquierda de cada plato.
4. Colocar tres cucharas, cada una a la derecha de cada plato.
5. Colocar tres cuchillos, cada uno a la derecha de cada cuchara.
6. Fin.

1. Inicio.
2. Para cada silla.
 - a. Colocar plato.
 - b. Colocar tenedor a la izquierda del plato.
 - c. Colocar cuchara a la derecha del plato.
 - d. Colocar cuchillo a la derecha de la cuchara.
3. Siguiendo silla hasta completar la mesa.
4. Fin.

Algoritmo "Poner mesa". Versión 4.

1. Inicio.
2. Para cada silla.
 - a. Colocar plato con tenedor a la izquierda, cuchara a la derecha y cuchillo a la derecha de la cuchara.
3. Siguiendo silla hasta completar la mesa.
4. Fin.

Algoritmo "Poner mesa". Versión 5.

1. Inicio.
2. Para cada silla.
 - a. Colocar platos y cubiertos.
3. Siguiendo silla hasta completar la mesa.
4. Fin.

Algoritmo "Poner mesa ". Versión 6.

1. Inicio
2. Colocar platos y cubiertos en la mesa.
3. Fin.

Hemos utilizado este ejemplo para analizar cuestiones como economía del algoritmo, eficiencia del algoritmo y el lenguaje utilizado. Todavía no estamos utilizando ningún lenguaje de programación, sino un lenguaje más o menos libre al que denominamos **pseudocódigo**.

- o **Pseudocódigo**

Nuestra lengua apela a diversos elementos compositivos para formar palabras. Uno de los más habituales es **pseudo** o **seudo**, que permite referirse a que **algo no es original**, sino que es **falso** o una **imitación**.

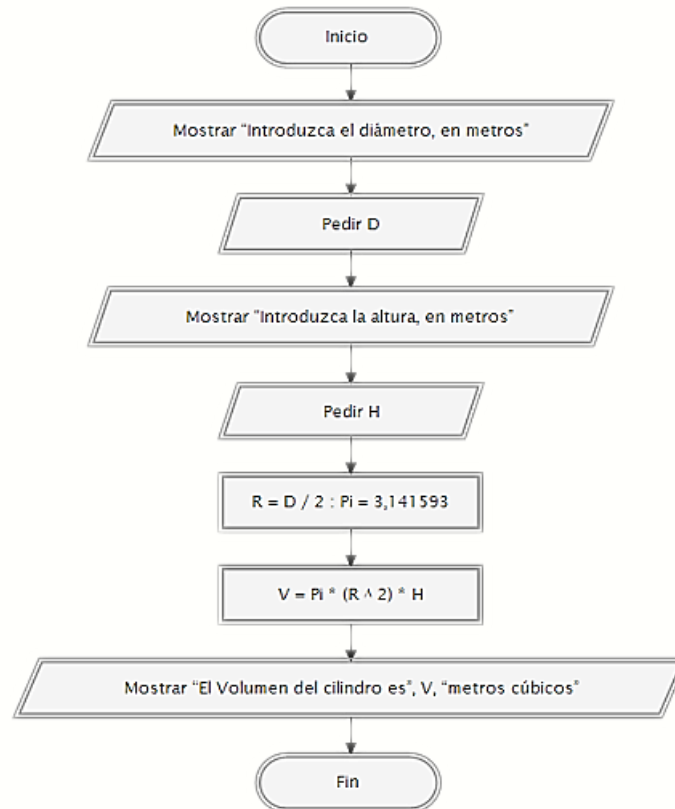
Se trata de un **falso lenguaje**, ya que apela a las normas de estructura de un lenguaje de programación aunque se encuentra desarrollado para que **pueda ser leído por un ser humano** y no interpretado por una máquina.

El pseudocódigo, en este sentido, está considerado como una **descripción** de un algoritmo que resulta independiente de otros **lenguajes de programación**.

1.4. Representación gráfico-esquemática de programas. Diagramas de flujo.

Un diagrama de flujo es una representación esquemática de los distintos pasos de un programa. Es otra forma distinta de representar algoritmos al pseudocódigo, pero que nos sirve de forma complementaria en el proceso de creación de la estructura del programa antes de ponernos delante del ordenador.

Ejemplo diagrama de flujo que proporciona el volumen de un cilindro dadas su altura y diámetro




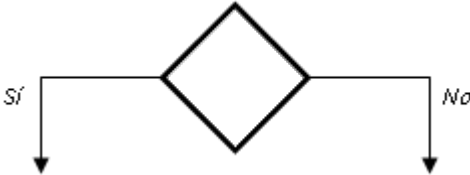

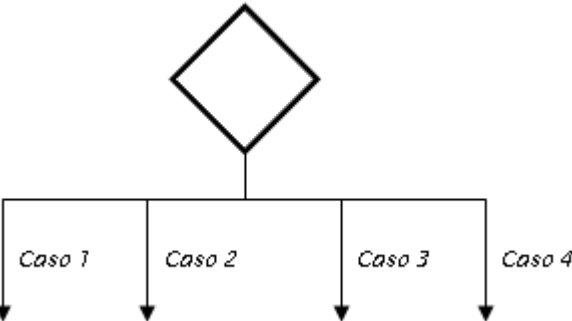





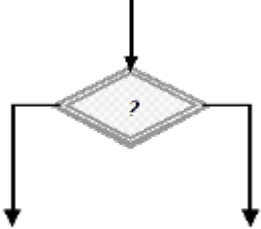
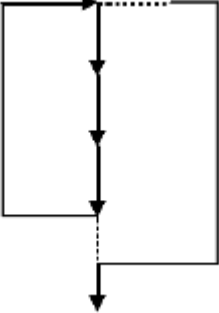
Ejemplo de pseudocódigo

El manejo u ordenación de un diagrama de flujo muy extenso se puede complicar. La solución a este problema la encontraremos, al menos parcialmente, utilizando el método del "Divide y vencerás". A través de lo que se denomina "programación modular".

En relación a los diagramas de flujo viene a ser equivalente a la organización de los planos de un proyecto: igual que tenemos planos de cimentación, planta 1, planta 2, secciones, instalaciones, etc. tendremos diagramas de flujo independientes para entrada de datos, proceso de cálculo número 1, proceso de cálculo número 2, salida de datos, etc.

Para la creación de diagramas de flujo utilizaremos unos símbolos y normas de construcción determinados. Podemos basarnos, por ejemplo, en un modelo simplificado de los estándares internacionales, con el objeto de poder interpretar y ser interpretados por otros programadores.

| | |
|---|---|
|  | Terminal. Indica comienzo o final de un programa, subprograma o módulo. |
|  | Captura y emisión de datos. Entrada o salida de información desde o hacia el ordenador. |
|  | Proceso. Cualquier proceso interno realizado por el ordenador como asignación de valor a variables, operaciones matemáticas, etc. |
|  | Decisión múltiple. El dato o condición planteada presenta distintas alternativas (casos), siguiendo el programa distinta vía en función del caso. |
|  | Línea de flujo. Sentido del flujo de procesos. Indica qué proceso viene a continuación del otro. |
|  | Decisión múltiple. El dato o condición planteada presenta distintas alternativas (casos), siguiendo el programa distinta vía en función del caso |
|  | Conector. Indica a través de una referencia (número, letra o texto) dónde debe continuar un diagrama de flujo que se interrumpe. |
|  | Módulo independiente. Recibe distintos nombres como subprograma, subrutina, proceso, procedimiento, etc. Al llegar a esta llamada el programa pasa a ejecutar todas las instrucciones contenidas en la subrutina para una vez terminadas continuar el flujo. |

| | |
|--|--|
|  | <p>Esquema o estructura secuencial</p> |
|  | <p>Esquema o estructura de decisión</p> |
|  | <p>Esquema o estructura de repetición o bucle.</p> |

2. Lenguaje JavaScript

2.1. ¿Qué es JavaScript?.

JavaScript, al igual que Flash, Visual Basic Script, es una de las múltiples maneras que han surgido para extender las capacidades del lenguaje HTML. Al ser la más sencilla, es por el momento la más extendida.

- JavaScript es un lenguaje script u orientado a documento, como pueden ser los lenguajes de macros que tienen muchos procesadores de texto y hojas de cálculo.
- No se puede desarrollar un programa con JavaScript que se ejecute fuera de un Navegador, aunque en este momento comienza a expandirse a otras áreas como la programación en el servidor con Node.js
- JavaScript es un lenguaje interpretado que se aloja en una página web HTML. Un lenguaje interpretado significa que a las instrucciones las analiza y procesa el navegador en el momento que deben ser ejecutadas.

Nuestro primer programa será el famoso "Hola Mundo", es decir un programa que muestre en el documento HTML el mensaje "Hola Mundo".

```
<html>

<head>

</head>

<body>

<script type="text/javascript">

    document.write('Hola Mundo');

</script>

</body>

</html>
```



Comprueba el código anterior desde el simulador online de código:

<http://elprofedemicurso.es/visorjavascript/llamada.html>

- El programa en **JavaScript** debe ir encerrado entre la marca **script** e inicializada la propiedad **type** con la cadena **text/javascript**:

```
<script type="text/javascript">
```

```
</script>
```

Para imprimir caracteres sobre la página debemos llamar al comando 'write' del objeto document. La información a imprimirse debe ir entre comillas y encerrada entre paréntesis. Todo lo que indicamos entre comillas aparecerá tal cual dentro de la página HTML.

Es decir, si pedimos al navegador que ejecute esta página mostrará el texto 'Hola Mundo'.

- Cada vez que escribimos una instrucción finalizamos con el carácter punto y coma.

ES IMPORTANTISIMO TENER EN CUENTA QUE JavaScript es SENSIBLE A MAYUSCULAS Y MINUSCULAS. NO ES LO MISMO ESCRIBIR:

document.write que DOCUMENT.WRITE (la primera forma es la correcta, la segunda forma provoca un error de sintaxis).

Nos acostumbraremos a prestar atención cada vez que escribamos en minúsculas o mayúsculas para no cometer errores sintácticos. Ya veremos que los nombres de funciones llevan letras en mayúsculas.

2.2. ¿Dónde y cómo insertar el código Javascript en las páginas web?

Hay varias formas de insertar los códigos Javascript en las páginas web.

- En el área del HEAD de la página o encabezado.
- En el área del BODY o cuerpo
- En un archivo externo. **(método que impedirá que las páginas cargen de forma más lenta).**



2.3. Impedir que Javascript haga más lentas las páginas

Para optimizar el rendimiento de una página, en el área del HEAD solo se debe poner el código Javascript que sea imprescindible para ejecutar funciones antes de cargar el cuerpo de la página, lo que se conoce como BODY.

Muchas funciones contenidas en el área del HEAD retardan considerablemente el inicio, por lo que el código excesivo se debe de ponerlo en área del BODY, preferentemente en el final, aunque lo ideal si es extenso es usarlo en un archivo externo.

Este método también se utiliza cuando se comparte el mismo código entre varias páginas web, se logra escribiendo el código en un pequeño archivo de texto con la extensión .JS, que lo podemos crear con el Bloc de notas y nos aseguramos que la página web lo cargue, insertando en el área del head la siguiente línea:

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

2.4. Funciones usadas para escribir con Javascript

Existen varias funciones en **Javascript** que permiten escribir y representar texto u otro contenido en una página web, son: **document.write**, **document.writeln** y **innerHTML**.

Los dos primeros escriben texto u otros elementos al cargar la página, si son llamados posteriormente, será necesario volverla a cargar.

innerHTML permite agregar dinamismo a la página, es decir poder escribir en ella después de que este cargada y sin utilizar un lenguaje de servidor como PHP.

Mediante ella podemos agregar contenido de forma dinámica sin modificar el resto del contenido.

2.5. Como usar la función document.write, ejemplos prácticos

La función **document.write** de Javascript permite escribir en una página texto, el resultado de una función o ambos.

Se utiliza de cualquiera de las siguientes formas:

- **document.write('texto');**
- **document.write(+funcion);**
- **document.write(+ 1raFuncion+ 2daFuncion);**

El resultado aparecerá en el lugar exacto de la página donde se inserte el código.

Lógicamente, este código debe ir encerrado entre dos etiquetas `<script>` para que el navegador lo interprete como tal.

El texto encerrado entre comillas será escrito literalmente.

Si no se usan comillas se asumirá que es una variable, esta debe estar antecedida por el signo: +

Algunos ejemplos a continuación.

- **Escribir texto y variables con Javascript**

- El primer ejemplo de **document.write** escribe en la página el texto 'Hola', el segundo el resultado de la variable **screen.width** que devuelve el ancho de la pantalla en píxeles y el tercero agrupa texto y la variable.

```
<html>

<head>

<title> Ejemplos JavaScript </title>

</head>

<body>

<script type="text/javascript">

    document.write('Hola ')

</script>

</body>

</html>
```



```
<html>

<head>

<title> Ejemplos JavaScript </title>

</head>

<body>

<script type="text/javascript">

    document.write(+screen.width)

</script>

</body>

</html>

<html>

<head>

<title> Ejemplos JavaScript </title>
```



```
</head>

<body>

<script type="text/javascript">

    document.write('Ancho de la pantalla: '+screen.width+' pixeles')

</script>

</body>

</html>
```

Comprueba los códigos anteriores desde el simulador online de código:

<http://elprofedemicurso.es/visorjavascript/llamada.html>

Por supuesto que no se utiliza Javascript para escribir un simple texto, se usa cuando el texto está acompañado de una variable que cambiará su valor dependiendo lo que se exprese con ella.

- Por ejemplo, de la siguiente forma podemos escribir la fecha de la última modificación de la página, observemos que la variable no está encerrada entre comillas y posee un signo de más.

```
<html>
```

```
<head>
```

```
<title> Ejemplos JavaScript </title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
    document.write('Ultima modificacion: '+window.document.lastModified)
```

```
</script>
```

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

- O podemos escribir el nombre del navegador con que se cargue la página:

```
<html>
```

```
<head>
```

```
<title> Ejemplos JavaScript </title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
    document.write('Usas el navegador: '+navigator.appName)
```

```
</script>
```

```
</body>
```

```
</html>
```



- Si quieres continuar escribiendo después de la variable, necesitas otro signo de más después de ella, por ejemplo:

```
<html>
```

```
<head>
```

```
<title> Ejemplos JavaScript </title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
    document.write('El título de esta página es: '+window.document.title+'
```

```
y la direccion URL es: '+window.document.URL)
```

```
</script>
```

```
</body>
```

```
</html>
```



○ Escribir vínculos con Javascript

La función **document.write** es muy utilizada para escribir vínculos e inclusive otros scripts.

En el siguiente ejemplo se utiliza **document.write** para escribir un link en la página, pero solo si el navegador usado es Firefox o Google Chrome, si se emplea Internet Explorer no se verá.

```
<html>
```

```
<head>
```

```
<title> Ejemplos JavaScript </title>
```

```
</head>
```

```
<body>
```

```
<script type="text/javascript">
```

```
//</pre></div><div data-bbox="113 836 504 853" data-label="Text"><pre>if(navigator.appName.indexOf("Netscape") != -1){</pre></div><div data-bbox="113 854 770 890" data-label="Text"><pre>    document.write('&lt;a href="http://elprofedemicurso.es"&gt;Solo para Firefox y Google<br/>    Chrome&lt;/a&gt;');} //]]&gt;</pre></div><div data-bbox="113 891 190 908" data-label="Text"><pre>&lt;/script&gt;</pre><hr/></div><div data-bbox="635 672 772 801" data-label="Image"><img alt="Script Editor icon" data-bbox="635 672 772 801"/>A purple rounded square icon with a white document and a black pen, labeled 'Script Editor' at the bottom.</div><div data-bbox="850 920 886 939" data-label="Page-Footer">22</div>
```

`</body></html>`

- **Escribir el mismo contenido en múltiples páginas**

Insertar texto repetitivo en múltiples páginas, usando javascript en un único archivo externo que pueda ser editado con facilidad. Al usar **document.write** en un archivo js externo vinculado a cada página del sitio, es posible escribir

Por ejemplo el siguiente código irá en un archivo de texto de nombre: "creditos.js", escribirá en nuestra página nuestra información de Copyright.

```
<html>

<head>

<title> Ejemplos JavaScript </title>

</head>

<body>

<script type="text/javascript">

//

document.write("&lt;div style='color:blue;font-size:14px;'&gt;");

document.write("© Copyright MiNombre 2015");

document.write("&lt;/div&gt;");

//]]&gt;

&lt;/script&gt;&lt;/body&gt;

&lt;/html&gt;</pre><hr/></div><div data-bbox="707 472 850 602" data-label="Image"><img alt="Icon of a script editor showing a document with a pen and the text 'Script Editor'." data-bbox="707 472 850 602"/></div><div data-bbox="113 693 790 730" data-label="Text"><p>En cada página solo necesitamos insertar en el final o en el lugar donde deseamos que se muestre:</p></div><div data-bbox="113 743 666 760" data-label="Text"><pre>&lt;script type="text/javascript" src="creditos.js"&gt; &lt;/script&gt;</pre></div><div data-bbox="113 772 501 790" data-label="Text"><p>Se mostrará en cada página lo siguiente:</p></div><div data-bbox="113 802 389 820" data-label="Text"><p>© Copyright MiNombre 2015</p></div><div data-bbox="113 832 866 869" data-label="Text"><p>Al modificar o editar el contenido del archivo "credito.js", cambiará el contenido que aparezca en cada página.</p></div><div data-bbox="850 920 886 938" data-label="Page-Footer">23</div>
```

3. Elementos básicos del Lenguaje JavaScript

3.1. Variables en JavaScript

Las Variables en JavaScript son “contenedores” que almacenan información.

Ejemplos:

```
var x=5;
```

```
var y=6;
```

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

Como el álgebra usamos letras (como la X) para almacenar valores (como 5).

De la expresión $z=x+y$ del ejemplo anterior, podemos calcular el valor de z, que es 11.

En JavaScript dichas letras son llamadas **variables**.

Las **variables** pueden almacenar valores ($x=5$) o expresiones ($z=x+y$).

Una variable es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo (numérico, cadena de caracteres, etc.)

3.1.1. Tipos de variables.

Una variable puede almacenar:

- Valores Enteros (100, 260, etc.)
- Valores Reales (1.24, 2.90, 5.01, etc.)
- Cadenas de caracteres ('Juan', 'Compras', 'Listado', etc.)
- Valores lógicos (true,false)

3.1.2. Reglas para nombrar variables.

- Las variables pueden tener nombres cortos (como x o y) o más descriptivos como edad, suma, volumentotal.
- Los nombres de las variables deben iniciar con una letra.
- Los nombres de variables también pueden iniciar con \$ o _.

- Los nombres de variable son sensibles a las mayúsculas, y e Y son variables diferentes.

3.2. Tipos de datos en JavaScript.

Las variables pueden almacenar otro tipo de datos, como texto (nombre="Antono López").

En JavaScript un texto como "Antonio López" es llamado cadena o string.

Hay muchos tipos de variables en JavaScript pero por ahora solo pensaremos en números o cadenas.

- Cuando se asigna un texto a una variable, debe estar encerrado en comillas dobles o simples.
- Cuando se asigna un número a una variable, no debe ponerse entre comillas, si se pone entre comillas, será tratado como texto.

En el siguiente ejemplo definimos una serie de variables y las mostramos en la página:

```
<html>

<head>

<title> trabajando con variables de JavaScript </title>

</head>

<body>

<script type="text/javascript">

  var nombre='Juan';

  var edad=10;

  var altura=1.92;

  var casado=false;

  document.write(nombre);

  document.write('<br>');

  document.write(edad);

  document.write('<br>');
```



```
document.write(altura);
```

```
document.write('<br>');
```

```
document.write(casado);
```

```
</script>
```

```
</body>
```

```
</html>
```

Ejemplos:

```
var pi=3.14;
```

```
var name="Pedro Páramo";
```

```
var answer='Si, yo soy';
```

3.3. Declarando (creando) variables en JavaScript.

Crear una variable en JavaScript se conoce más como “declarar” una variable.

Las variables en JavaScript se declaran con la palabra clave var.

```
var nombreAuto;
```

Después de declarada la variable está vacía (no tiene valor).

Para asignar un valor a la variable se utiliza el signo igual.

```
marcaCoche="Peugeot";
```

Sin embargo se puede asignar el valor a la variable desde el momento de ser creada.

```
var marcaCoche="Peugeot";
```

En el siguiente ejemplo, creamos una variable llamada marcaCoche con el valor “Peugeot” y colocamos dicho valor en párrafo HTML con el id=“demo”.

```
<html>
```

```
<head>
```

```
<title> trabajando con variables de JavaScript </title>
```

```
</head>
```

```
<body>
```

```
<p>Presiona el botón para crear la variable y mostrar el resultado.</p>
```

```
<button onclick="myFuncion()">Inténtalo</button>
```

```
<p id="demo"></p>
```

```
<script>
```

```
function myFuncion()
```

```
{
```



```
var marcaCoche="Peugeot";

document.getElementById("demo").innerHTML=marcaCoche;

}

</script>

</body>

</html>
```

Comprueba los códigos anteriores desde el simulador online de código:
<http://elprofedemicurso.es/visorjavascript/llamada.html>

- **Una declaración, muchas variables.**

Se pueden declarar varias variables en un solo paso. Solo se inicia con la palabra var y se separan las variables con comas.

```
var nombre="José", edad=35, trabajo="Carpintero";
```

La declaración también puede dividirse en varias líneas

```
var nombre="José",
edad=35,
trabajo="Carpintero";
```

- **Ámbito de las variables**

El ámbito de una variable (llamado "scope" en inglés) es la zona del programa en la que se define la variable. JavaScript define dos ámbitos para las variables: global y local.

El siguiente ejemplo ilustra el comportamiento de los ámbitos:

```
function creaMensaje() {

    var mensaje = "Mensaje de prueba";

}
```

```
creaMensaje();
```

```
alert(mensaje);
```

El ejemplo anterior define en primer lugar una función llamada `creaMensaje` que crea una variable llamada `mensaje`. A continuación, se ejecuta la función mediante la llamada `creaMensaje()`; y seguidamente, se muestra mediante la función `alert()` el valor de una variable llamada `mensaje`.

Sin embargo, al ejecutar el código anterior no se muestra ningún mensaje por pantalla. La razón es que la variable `mensaje` se ha definido dentro de la función `creaMensaje()` y por tanto, es una variable local que solamente está definida dentro de la función.

Cualquier instrucción que se encuentre dentro de la función puede hacer uso de esa variable, pero todas las instrucciones que se encuentren en otras funciones o fuera de cualquier función no tendrán definida la variable `mensaje`. De esta forma, para mostrar el mensaje en el código anterior, la función `alert()` debe llamarse desde dentro de la función `creaMensaje()`:

```
function creaMensaje() {  
  
    var mensaje = "Mensaje de prueba";  
  
    alert(mensaje);  
  
}
```

```
creaMensaje();
```

Además de **variables locales**, también existe el concepto de variable global, que está definida en cualquier punto del programa (incluso dentro de cualquier función).

```
var mensaje = "Mensaje de prueba";  
  
function muestraMensaje() {  
  
    alert(mensaje);  
  
}
```

El código anterior es el ejemplo inverso al mostrado anteriormente. Dentro de la función `muestraMensaje()` se quiere hacer uso de una variable llamada `mensaje` y que no ha sido definida dentro de la propia función. Sin embargo, si se ejecuta el código anterior, sí que se muestra el mensaje definido por la variable `mensaje`.

El motivo es que en el código JavaScript anterior, la variable `mensaje` se ha definido fuera de cualquier función. Este tipo de variables automáticamente se

transforman en variables globales y están disponibles en cualquier punto del programa (incluso dentro de cualquier función).

De esta forma, aunque en el interior de la función no se ha definido ninguna variable llamada mensaje, la variable global creada anteriormente permite que la instrucción **alert()** dentro de la función muestre el mensaje correctamente.

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada var o no. Sin embargo, las variables definidas dentro de una función pueden ser globales o locales.

Si en el interior de una función, las variables se declaran mediante var se consideran locales y las variables que no se han declarado mediante var, se transforman automáticamente en variables globales.

Por lo tanto, se puede rehacer el código del anterior ejemplo para que muestre el mensaje correctamente. Para ello, simplemente se debe definir la variable dentro de la función sin la palabra reservada var, para que se transforme en una variable global:

```
function creaMensaje() {  
  
    mensaje = "Mensaje de prueba";  
  
}
```

```
creaMensaje();
```

```
alert(mensaje);
```

- **¿Qué sucede si una función define una variable local con el mismo nombre que una variable global que ya existe?**

En este caso, las variables locales prevalecen sobre las globales, pero sólo dentro de la función:

```
var mensaje = "gana la de fuera";
```

```
function muestraMensaje() {  
  
    var mensaje = "gana la de dentro";  
  
    alert(mensaje);  
  
}
```

```
alert(mensaje);  
  
muestraMensaje();  
  
alert(mensaje);
```

El código anterior muestra por pantalla los siguientes mensajes:

gana la de fuera
gana la de dentro
gana la de fuera

Dentro de la función, la variable local llamada mensaje tiene más prioridad que la variable global del mismo nombre, pero solamente dentro de la función.

- **¿Qué sucede si dentro de una función se define una variable global con el mismo nombre que otra variable global que ya existe?**

En este otro caso, la variable global definida dentro de la función simplemente modifica el valor de la variable global definida anteriormente:

```
var mensaje = "gana la de fuera";  
  
function muestraMensaje() {  
  
    mensaje = "gana la de dentro";  
  
    alert(mensaje);  
  
}  
  
alert(mensaje);
```

```
muestraMensaje();
```

```
alert(mensaje);
```

En este caso, los mensajes mostrados son:

gana la de fuera

gana la de dentro

gana la de dentro

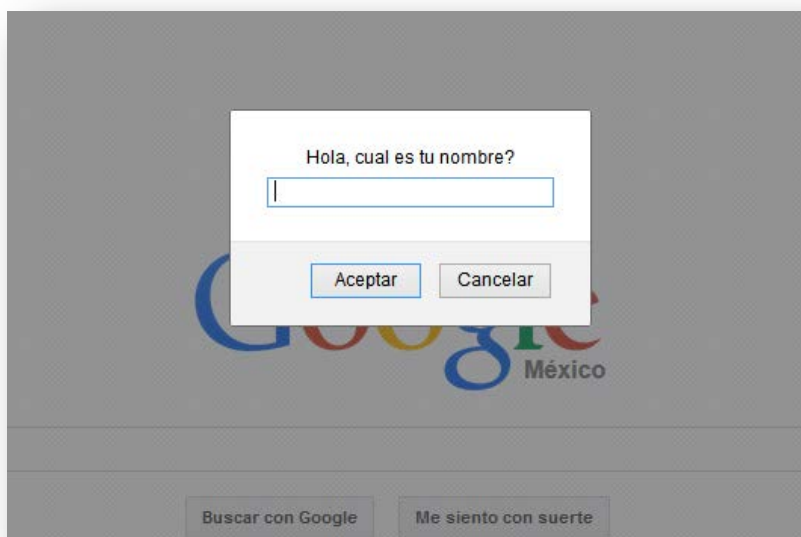
La recomendación general es definir como variables locales todas las variables que sean de uso exclusivo para realizar las tareas encargadas a cada función. Las variables globales se utilizan para compartir variables entre funciones de forma sencilla.

Comprueba los códigos anteriores desde el simulador online de código:

<http://writecodeonline.com/javascript/>

3.4. Entrada de datos por teclado.

Para la entrada de datos por teclado tenemos la función `prompt`. Cada vez que necesitamos ingresar un dato con esta función, aparece una ventana donde cargamos el valor.



Hay otras formas más sofisticadas para la entrada de datos en una página HTML, pero para el aprendizaje de los conceptos básicos de JavaScript nos resultará más práctica esta función.

La sintaxis de la función prompt es:

La función prompt tiene dos parámetros: uno es el mensaje y el otro el valor inicial a mostrar.

```
<variable que recibe el dato>=prompt(<mensaje a mostrar en la ventana>,<valor  
inicial a mostrar en la ventana>);
```

Para ver su funcionamiento analicemos este ejemplo:

```
<html>

<head>

<title> ejemplo recogida de datos </title>

</head>

<body>

<script type="text/javascript">

    var nombre;

    var edad;

    nombre=prompt('Ingrese su nombre:','');

    edad=prompt('Ingrese su edad:','');

    document.write('Hola ');

    document.write(nombre);

    document.write(', así que tienes ');

    document.write(edad);

    document.write(' años');

</script>

</body>

</html>
```



Cabe mencionar que el dato ingresado por el método prompt siempre se va a almacenar como con un tipo string, entonces si pedimos un dato numérico este se almacenara como tipo string, ejemplo.

```
var mensaje = "Hola, Cual es tu edad?";

var edad = prompt(mensaje); // edad = "18";
```

Si ingresamos la edad de 18 este dato sera almacenado como tipo string y no podremos realizar operaciones matemáticas con él, para esto podremos convertir el tipo string a tipo numérico con ayuda de la función parseInt(), ejemplo.-

```
var mensaje = "Hola, Cual es tu edad?";

var edad = parseInt(prompt(mensaje));
```

Y de este modo ya podríamos utilizar la variable edad para operaciones matemáticas.

- **Parámetros de prompt**

Hasta el momento utilizamos prompt con un solo parámetro donde pasamos el mensaje a mostrar, pero también podemos utilizar un segundo parámetro en el que vamos a colocar un valor por defecto en caso de que el usuario no ingrese nada en la caja de dialogo prompt, ejemplo.-

```
var mensaje = "Hola, Cual es tu nombre?";
```

```
var nombre = prompt(mensaje, "Luisa");
```

Conclusiones:

Prompt es un método que nos será de gran ayuda en caso de querer obtener un dato por parte del usuario, al menos al principio del curso ya que más adelante veremos otros métodos para obtener información.

Comprueba los códigos anteriores desde el simulador online de código:

<http://elprofedemicurso.es/visorjavascript/llamada.html>

3.5. Introducción a algunas funciones en JavaScript

Pequeña introducción para el uso de algunas funciones estándar.

3.5.1. Función alert.

Esta función es un método del objeto Window y es una de las más utilizadas al momento de iniciarnos en JavaScript, pues 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. La función alert recibe como parámetro el mensaje que se debe mostrar en la ventana.

Sintaxis:

```
alert("mensaje a mostrar")
```

A continuación se muestra un ejemplo donde aparece en la pantalla una pequeña ventana con un mensaje de saludo:

```
<html>

<head>

<title> Ejemplo alert </title>

<script language="javascript" type="text/javascript">

alert ( "Hola, como estás" )

</script>

</head>

<body>

</body>

</html>
```

3.5.2. 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.

Sintaxis:

```
document.write("mensaje")
```

Ejemplo:

```
<html>

<head>

<title>Ejemplo write </title>

<script language="javascript" type="text/javascript">

document.write( "Hola, nos alegra que estés por aquí... ")

</script>

</head>

<body>

</body>

</html>
```

3.5.3. Función prompt

Al igual que la función **alert**, la función **prompt** es también un método del objeto **Window**. Esta función se utiliza cuando el usuario ingresa datos por medio del teclado. Con esta función aparece una ventana en la pantalla, con un espacio para el valor que se debe ingresar 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.

Sintaxis:

```
variable = prompt("mensaje", "valor inicial")
```

Ejemplo:

```
<html>

<head>

<title>Ejemplo prompt </title>

<script language="javascript" type="text/javascript">

var nombre = prompt ("Ingrese su nombre", "")

document.write( "Hola "+ nombre)

</script>

</head>

<body>

</body>

</html>
```

3.5.4. Método print del objeto window.

```
window.print()
```

Si lo deseas vincular a la pulsación de un botón quedaría así:

```
<input type="button" name="imprimir" value="Imprimir" onclick="window.print();">
```

3.5.5. Abrir ventana en JavaScript. Método open del objeto window.

Para poder abrir una ventana con JavaScript utilizaremos **el método .open del objeto Window**.

Este método tiene la siguiente sintaxis:

```
window.open (url:string,nombreVentana:string,caracteristicas :string)
```

Dónde sus parámetros pueden ser:

- **url: string** - Indicamos cual es la URL de la ventana que se va a abrir.
- **nombreVentana: string** - Nombre que se le va a asignar a la ventana. Este nombre es especialmente interesante cuando usamos esta ventana como frame.
- **caracteristicas: string** - Proporcionamos una cadena con las características que debe de tener la ventana. Dichas características son:
 - **directories** - en el caso de que esté activada nos mostrara la barra de vínculos.
 - **height** - indicaremos la altura que debe de tener la página.
 - **location** - nos servirá para desactivar la barra de navegación.
 - **menubar** - representa a la barra de menús superior.
 - **scrollbars** - sirve para indicar si aparecerán o no las barras de scroll.
 - **status** - representa a la barra de estado.
 - **titlebar** - representa a la barra del título.

A la mayoría de las características se les asignaran un "yes" o un "no" dependiendo de si queremos que aparezcan o no. Por defecto las características están a "yes".

```
function abrirVentana(url) {
```

```
    window.open(url, "nuevo", "directories=no, location=no, menubar=no, scrollbars=yes,  
statusbar=no, tittlebar=no, width=400, height=400");
```

Veamos un ejemplo de sentencia Javascript completa para abrir una ventana secundaria:

```
window.open("http://www.elprofedemicurso.es", "ventana1",  
"width=120,height=300,scrollbars=NO", statusbar=no, tittlebar=no,)
```

Ejemplo con enlace que abre una ventana secundaria

```
<script language=javascript>
function ventanaSecundaria (){
    window.open("URL","ventana1","width=120,height=300,scrollbars=NO")
}
</script>
```

Ahora Veamos cómo quedaría todo ese enlace en la página.

```
<a href="javascript:ventanaSecundaria()"> Pincha en este enlace para abrir la ventana
secundaria</a>
```

Código final a probar:

```
<html>

<head><title>ventana javascript popup</title>

<script language=javascript>

function ventanaSecundaria (){

window.open("http://www.elprofedemicurso.es","ventana1","width=120,height=300,
scrollbas=NO")

}

</script>

</head>

<body>

<a href="javascript:ventanaSecundaria()"> Pincha en este enlace para abrir la ventana
secundaria</a>

<!--un botón llama a la función ventana secundaria //-->

<input type="button" value="Abrir ventana" onclick="ventanaSecundaria()" />

</body>

</html>
```

3.6. Operadores en JavaScript

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

Los operadores se dividen en:

1. Operadores de asignación.
2. Operadores de incremento y decremento.
3. Operadores lógicos.
4. Operadores matemáticos
5. operadores relacionales.

3.6.1. Asignación.

El operador de asignación es el más utilizado y el más sencillo. Este operador se utiliza para guardar un valor específico en una variable. El símbolo utilizado es `=` (no confundir con el operador `==` que se verá más adelante):

```
var numero1 = 3;
```

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

```
var numero1 = 3;
```

```
var numero2 = 4;
```

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

```
5 = numero1;
```

```
// Ahora, la variable numero1 vale 5
```

```
numero1 = 5;
```

```
// Ahora, la variable numero1 vale 4
```

```
numero1 = numero2;
```

3.6.2. Incremento y decremento

Estos dos operadores solamente son válidos para las variables numéricas y se utilizan para incrementar o decrementar en una unidad el valor de una variable.

Ejemplo:

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

El operador de incremento se indica mediante el prefijo ++ en el nombre de la variable. El resultado es que el valor de esa variable se incrementa en una unidad. Por tanto, el anterior ejemplo es equivalente a:

```
var numero = 5;  
  
numero = numero + 1;  
  
// numero = 6  
  
alert(numero);
```

De forma equivalente, el operador decremento (indicado como un prefijo -- en el nombre de la variable) se utiliza para decrementar el valor de la variable:

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

El anterior ejemplo es equivalente a:

```
var numero = 5;  
  
numero = numero - 1;  
  
// numero = 4  
  
alert(numero);
```

Los operadores de incremento y decremento no solamente se pueden indicar como prefijo del nombre de la variable, sino que también es posible utilizarlos

como sufijo. En este caso, su comportamiento es similar pero muy diferente. En el siguiente ejemplo:

```
var numero = 5;

numero++;

// numero = 6

alert(numero);
```

El resultado de ejecutar el script anterior es el mismo que cuando se utiliza el operador `++numero`, por lo que puede parecer que es equivalente indicar el operador `++` delante o detrás del identificador de la variable. Sin embargo, el siguiente ejemplo muestra sus diferencias:

```
var numero1 = 5;

var numero2 = 2;

// numero3 = 7, numero1 = 6

numero3 = numero1++ + numero2;

var numero1 = 5;

var numero2 = 2;

// numero3 = 8, numero1 = 6

numero3 = ++numero1 + numero2;
```

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

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

3.6.3. 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*.

- o **Negación**

Uno de los operadores lógicos más utilizados es el de la negación. Se utiliza para obtener el valor contrario al valor de la variable:

```
var visible = true;
```

```
// Muestra "false" y no "true"
```

```
alert(!visible);
```

La negación lógica se obtiene prefijando el símbolo ! al identificador de la variable. El funcionamiento de este operador se resume en la siguiente tabla:

| variable | !variable |
|----------|-----------|
| true | false |
| false | true |

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

- o Si la variable contiene un número, se transforma en false si vale 0 y en true para cualquier otro número (positivo o negativo, decimal o entero).

- Si la variable contiene una cadena de texto, se transforma en false si la cadena es vacía (""), y en true en cualquier otro caso.

```
var cantidad = 0;

// vacio = true

vacio = !cantidad;

cantidad = 2;

//vacio = false

vacio = !cantidad;

var mensaje = "";

// mensajeVacio = true

mensajeVacio = !mensaje;

mensaje = "Bienvenido";

// mensajeVacio = false

mensajeVacio = !mensaje;
```

- **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 |

| variable1 | variable2 | variable1 && variable2 |
|-----------|-----------|------------------------|
| false | false | false |

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

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

o **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 |

| variable1 | variable2 | variable1 variable2 |
|-----------|-----------|------------------------|
| false | false | false |

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

```
valor1 = false;
valor2 = false;
// resultado = false
resultado = valor1 || valor2;
```

3.6.4. Matemáticos

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

```
var numero1 = 10;
var numero2 = 5;
// resultado = 2
resultado = numero1 / numero2;
// resultado = 13
resultado = 3 + numero1;
// resultado = 1
resultado = numero2 - 4;
```

```
// resultado = 50
```

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

Además de los cuatro operadores básicos, JavaScript define otro operador matemático que no es sencillo de entender cuando se estudia por primera vez, pero que es muy útil en algunas ocasiones.

Se trata del operador "*módulo*", que calcula el resto de la división entera de dos números. Si se divide por ejemplo 10 y 5, la división es exacta y da un resultado de 2. El resto de esa división es 0, por lo que módulo de 10 y 5 es igual a 0.

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

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

```
var numero1 = 10;
```

```
var numero2 = 5;
```

```
// resultado = 0
```

```
resultado = numero1 % numero2;
```

```
numero1 = 9;
```

```
numero2 = 5;
```

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

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

```
var numero1 = 5;
```

```
// numero1 = numero1 + 3 = 8
```

```
numero1 += 3;
```

```
// numero1 = numero1 - 1 = 4
```

```
numero1 -= 1;
```

```
// numero1 = numero1 * 2 = 10
```

```
numero1 *= 2;
```

```
// numero1 = numero1 / 5 = 1
```

```
numero1 /= 5;
```



```
// numero1 = numero1 % 4 = 1
```

```
numero1 %= 4;
```

3.6.5. Relacionales

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

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

```
var numero1 = 3;
```

```
var numero2 = 5;
```

```
// resultado = false
```

```
resultado = numero1 > numero2;
```

```
// resultado = true
```

```
resultado = numero1 < numero2;
```

```
numero1 = 5;
```

```
numero2 = 5;
```

```
// resultado = true
```

```
resultado = numero1 >= numero2;
```

```
// resultado = true
```

```
resultado = numero1 <= numero2;
```

```
// resultado = true
```

```
resultado = numero1 == numero2;
```

```
// resultado = false
```

```
resultado = numero1 != numero2;
```

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

// El operador "=" asigna valores

```
var numero1 = 5;
```

// numero1 = 3 y resultado = 3

```
resultado = numero1 = 3;
```

// El operador "==" compara variables

```
var numero1 = 5;
```

// numero1 = 5 y resultado = false

```
resultado = numero1 == 3;
```

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

```
var texto1 = "hola";
```

```
var texto2 = "hola";
```

```
var texto3 = "adios";
```

// resultado = false

```
resultado = texto1 == texto3;
```

//resultado = false

```
resultado = texto1 != texto2;
```

// resultado = false

```
resultado = texto3 >= texto2;
```

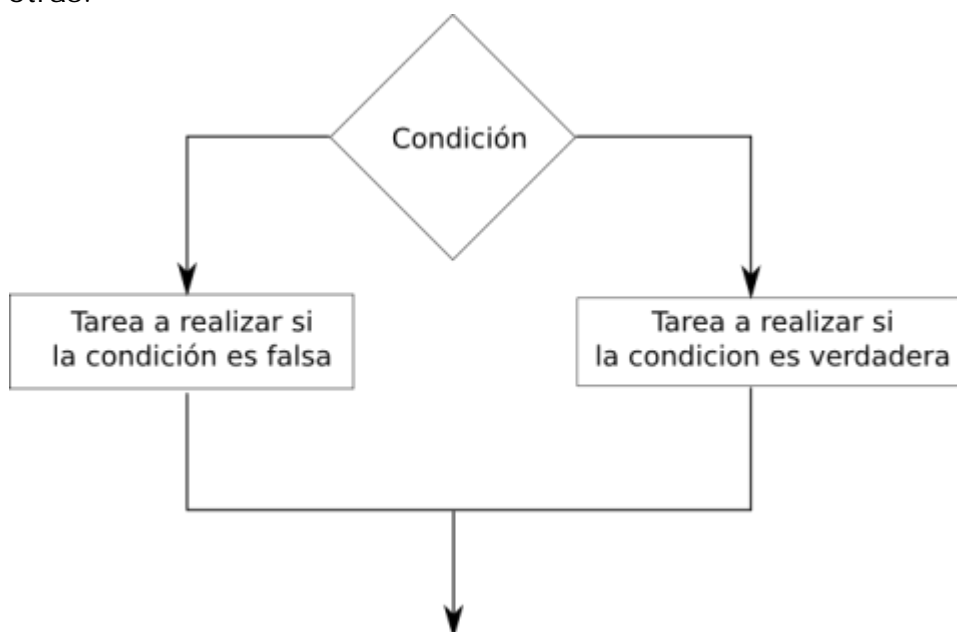
Cuando se utilizan cadenas de texto, los operadores "mayor que" (>) y "menor que" (<) siguen un razonamiento no intuitivo: se compara letra a letra comenzando desde la izquierda hasta que se encuentre una diferencia entre las dos cadenas de texto. Para determinar si una letra es mayor o menor que otra,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

3.7. Tipos de estructuras de control en JavaScript

Las estructuras de control en JavaScript y en la mayoría de los lenguajes de programación se utilizan en los para definir el flujo de instrucciones que se van ejecutando. Si no fuera por las estructuras de control lo único que podríamos hacer es ejecutar una instrucción tras otra y no tendríamos forma de aplicar unas funciones u otras en según las condiciones que nosotros queramos establecer.

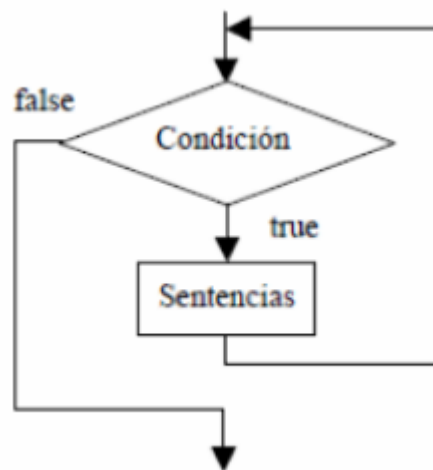
En JavaScript disponemos de 2 tipos de estructuras de control:

- **Estructuras condicionales.** Este tipo de estructura de control tiene como objetivo realizar una bifurcación del flujo de instrucciones. Cuando el programa llega a un punto, nosotros establecemos una condición en función de la misma el programa seguirá ejecutando unas instrucciones u otras.



- **Estructuras de repetición.** Este tipo de estructuras de control también conocidas como **bucles** se utilizan para realizar de forma repetida varias acciones. Con un bucle podemos por ejemplo mostrar en pantalla todos los números del 1 al 100 sin tener que escribir 100 veces las instrucciones **alert** o **document.write**.

Otro ejemplo muy fácil de comprender relacionado con la seguridad informática se da cuando en formularios html de control de acceso se piden una y otra vez los credenciales de acceso hasta que el usuario proporciona una pareja usuario/clave válida y accede al sistema. Tu como programador no sabes cuantas veces vas a tener que pedir la clave al usuario, puede que la introduzca bien a la primera, a la segunda, a la tercer o quizá nunca. Para estos casos los bucles son la solución. Con un bucle puedes pedir de forma reiterada los credenciales de acceso hasta que o bien proporciona unos válidos y accede o bien llega a un número máximo de intentos y se le deniega incluso el derecho a seguir probando contraseñas por un tiempo.



- **Estructuras de control de errores.** Estas estructuras son clave en el mundo de la seguridad informática. Son aquellas que **permiten controlar los errores que el usuario final comete de forma fortuita o intencionada** y poder seguir trabando de forma normal. Los errores más típicos a tener en cuenta se producen cuando pedimos al usuario que introduzca un número pero el usuario nos introduce una letra. Esto producirá un error en nuestro programa y hará que se comporte de forma inadecuada si no hemos controlado mediante una estructura de control de errores este caso.

3.7.1. Estructura de control javascript condicional (IF ELSE)

Las estructuras condicionales en javascript nos sirven para **tomar decisiones en función de una condición** que nosotros establecemos. Su sintaxis es así:

```
if (condicion)
{
instrucciones que se ejecutarán si se cumple la condición
}
else
{
instrucciones que se ejecutarán si NO se cumple la condición
}
```

"If" es una sentencia que significa "si condicional". La idea es que si sucede tal cosa, (si la condición es verdadera) se debe ejecutar la sentencia que le sigue, es decir la misma sólo se ejecutaría en caso de que la expresión de tipo Boolean sea verdadera. Si es falsa, el intérprete pasará a la parte **"else"** que significa "sino"; y ejecutará las instrucciones que existen dicha parte. La parte **"else"** no es obligatoria, pero hay muchos casos en los que necesitamos ejecutar unas instrucciones en caso de que se cumpla la condición u otras en caso contrario.

A continuación un ejemplo práctico en el que solicitamos la edad al visitante y le dejamos entrar o no en base a su mayoría de edad:

```
<script>
var edad = prompt("Dime tu edad");
if (num >=18) {
alert('Eres mayor de edad, puedes acceder');
}else {
alert('Eres menor de edad; NO puedes acceder');
}
</script>
```

A continuación un ejemplo donde comparamos la igualdad de la contraseña 2 veces, para continuar con un registro a una aplicación web o móvil.

```
<!DOCTYPE html>

<html>
<head>
<meta charset="utf-8">
<title>JavaScript – Comprobar igualdad de contraseñas en un registro de usuario nuevo</title>

<script>

//declaramos las variables

var contr1;
var contr2;

function capturar(){

    // Obtenemos el valor de la contraseña por su id

    contr1=document.getElementById("contr1").value;
    contr2= document.getElementById("contr2").value;
    // comparamos el valor de las contraseñas introducidas con un condicional if-else

    if (contr1==contr2) {
        alert('Todo correcto');

        /*Mediante la propiedad .innerHTML accedemos al contenido de texto de un elemento y
podemos rellenar una capa que espera. En el ejemplo, rellenamos la capa con id resultado.*/

        document.getElementById("resultado").innerHTML="las dos contraseñas coinciden";

    }else {

        alert("las contraseñas no coinciden, inténtalo de nuevo");

        /*Mediante la propiedad .innerHTML accedemos al contenido de texto de un
elemento y podemos rellenar una capa que espera. En el ejemplo, rellenamos la capa con id
resultado.*/

        document.getElementById("resultado").innerHTML="las dos contraseñas
no coinciden, rellena el formulario de nuevo";
```

```
    }  
  }  
  
</script>  
  
</head>  
<body>  
<h1>Comparar contraseñas</h1>  
  <form id="formulario_registro">  
    Nombre:  
    <br>  
    <input type="text" name="nombre" value="Nombre" id="nombre">  
    <br>  
    Contraseña:  
    <br>  
    <input name="contraseña1" type="text" id="contr1" value="Inserte la contraseña"  
size="40">  
    <br>  
    Repite la Contraseña:  
    <br>  
    <input name="contraseña2" type="text" id="contr2" value="Inserte la contraseña de nuevo"  
size="40">  
  
    <p><input type="checkbox" name="acepto" id="acepto"> Acepto el contrato</p>  
  
    <input type="button" value="aceptar registro" onclick="capturar()">  
  
</form>  
  
<br>  
  <div id="resultado"></div>  
</body>  
</html>
```



Comprueba el código anterior desde el simulador online de código:

<http://elprofedemicurso.es/visorjavascript/llamada.html>

3.7.2. IF ELSE IF ANIDADAS

Decimos que una estructura condicional es anidada cuando por la rama del verdadero o el falso de una estructura condicional hay otra estructura condicional.

Ejemplo: Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:

Si el promedio es ≥ 7 mostrar "Bien".

Si el promedio es ≥ 4 y < 7 mostrar "Mejorable".

Si el promedio es < 4 mostrar "Deberías estudiar más".

Solución:

```
<html>
<head>
<meta charset="utf-8">
  <title>Promedio de notas</title>
</head>
<body>
<script type="text/javascript">
var nota1,nota2,nota3;
//El usuario introduce las 3 notas y convertimos los 3 string en enteros

nota1=parseInt(prompt('Ingrese 1ra. nota:',''));

nota2= parseInt(prompt('Ingrese 2da. nota:',''));

nota3= parseInt(prompt('Ingrese 3ra. nota:',''));

//Definir y calcular promedio

var pro;

pro=(nota1+nota2+nota3)/3;

if (pro>=7)

{
  document.write('Bien');
```



```

}
else
{
if (pro>=4)

{
document.write('mejorable');
}

Else
document.write('Deberías estudiar más');

</script>

</body>
</html>

```



```

{
}

```

Comprueba el código anterior desde el simulador online de código:

<http://elprofedemicurso.es/visorjavascript/llamada.html>

Ejercicio propuesto:

Construir un programa que calcule el índice de masa corporal de una persona

($IMC = \text{peso [kg]} / \text{altura}^2 \text{ [m]}$) e indique el estado en el que se encuentra esa persona en función del valor de IMC:

| Valor de IMC | Diagnóstico |
|--------------|---|
| < 16 | Criterio de ingreso en hospital |
| de 16 a 17 | infrapeso |
| de 17 a 18 | bajo peso |
| de 18 a 25 | peso normal (saludable) |
| de 25 a 30 | sobrepeso (obesidad de grado I) |
| de 30 a 35 | sobrepeso crónico (obesidad de grado II) |
| de 35 a 40 | obesidad premórbida (obesidad de grado III) |
| >40 | obesidad mórbida (obesidad de grado IV) |

Nota 1: se recomienda el empleo de sentencias if–else anidadas.

Nota 2: Los operandos (peso y altura) deben ser introducidos por teclado por el usuario.

Nota 3: utiliza **parseInt()** para números enteros o **parseFloat()** para números con decimales.

3.7.3. Estructura de control javascript condicional Switch.

La instrucción **switch** es una alternativa para remplazar los **if/else if**.

No en todos los casos nos es suficiente una estructura de control que nos permita realizar una acción si se cumple una condición u otra acción si no se cumple. A veces nos ocurrirá que debemos hacer unas acciones si una variable tiene un valor, otras si tiene un valor distinto y otras si tiene otro valor distinto al anterior. Es el caso típico de los menús de elección de opciones. En función de la opción elegida por el usuario nosotros debemos hacer lo que nos pide. No podemos preguntar por mayor o menor. Para estos casos entre otros muchos se crearon las estructuras de control SWITCH; cuya sintaxis es así:

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

Con un ejemplo sencillo veremos cuál es su sintaxis. Confeccionar un programa que solicite que ingrese un valor entre 1 y 5. Luego mostrar en castellano el valor ingresado. Mostrar un mensaje de error en caso de haber ingresado un valor que no se encuentre en dicho rango.

```
<!DOCTYPE html>

<html>
<head>
<meta charset="utf-8">
<title>escoge entre 1 y 5</title>

<body>

<script type="text/javascript">

    var valor;

    valor=parseInt(prompt('Ingresa un valor comprendido entre 1 y 5:',''));

    switch (valor) {

        case 1: document.write('uno');

            break;

        case 2: document.write('dos');

            break;

        case 3: document.write('tres');

            break;

        case 4: document.write('cuatro');

            break;

        case 5: document.write('cinco');

            break;

        default:document.write('debe ingresar un valor comprendido entre 1 y 5.');
```

Debemos tener en cuenta que la variable que analizamos debe ir después de la instrucción `switch` entre paréntesis. Cada valor que se analiza debe ir luego de la palabra clave `'case'` y seguido a los dos puntos, las instrucciones a ejecutar, en caso de verificar dicho valor la variable que analiza el `switch`.

Es importante disponer la palabra clave `'break'` al finalizar cada caso. Las instrucciones que hay después de la palabra clave `'default'` se ejecutan en caso que la variable no se verifique en algún `case`. De todos modos el `default` es opcional en esta instrucción.

Ejemplo switch. En función del día de la semana, escribe un mensaje u otro:

```
<!DOCTYPE html>

<html>
<head>
<meta charset="utf-8">
<title>Mensaje según dia</title>

</head>

<body>

<script type="text/javascript">

var a = new Date();

Dia=a.getDay();

switch (Dia)

{

case 1:

    document.write("<b>Lunes. A clase de Javascript</b>");

    break;

case 2:

    document.write("<b>Martes. A clase de Javascript</b>");

    break;

case 3:

    document.write("<b>Miercoles. A clase de Javascript</b>");

    break;

case 4:

    document.write("<b>Jueves. A clase de Javascript</b>");

    break;

case 5:

    document.write("<b>Viernes social</b>");
```

```
break;
```

```
case 6:
```

```
    document.write("<b>Sábado sexual !!!</b>");
```

```
    break;
```

```
case 0:
```

```
    document.write("<b>Domingo familiar</b>");
```

```
    break;
```

```
default:
```

```
    document.write("<b>no estamos aún en fin de semana!</b>");
```

```
}
```

```
</script>
```

```
<p>En este script el Domingo=0, Lunes=1, Martes=2, etc.</p>
```

```
</body>
```

```
</html>
```

Ejemplo switch. Ingresar por teclado el nombre de un color (rojo, verde o azul), luego pintar el fondo de la ventana con dicho color:

```
<!DOCTYPE html>

<html>
<head>
<meta charset="utf-8">
<title>Escoge color por teclado </title>

</head>

<body>

<script type="text/javascript">

    var col;

    col=prompt('Ingresa el color con que se quiere pintar el fondo de la ventana (rojo, verde, azul)'
,'recuerda: rojo, verde o azul');

    switch (col) {

        case 'rojo': document.bgColor='#ff0000';

            break;

        case 'verde': document.bgColor='#00ff00';

            break;

        case 'azul': document.bgColor='#0000ff';

            break;

    }

</script>

</body>

</html>
```

Cuando verificamos cadenas debemos encerrarlas entre comillas el valor a analizar:

```
case 'rojo': document.bgColor='#ff0000';  
  
    break;
```

Para cambiar el color de fondo de la ventana debemos asignarle a la propiedad `bgColor` del objeto `document` el color a asignar (el color está formado por tres valores hexadecimales que representan la cantidad de rojo, verde y azul), en este caso al valor de rojo le asignamos `ff` (255 en decimal) es decir el valor máximo posible, luego `00` para verde y azul (podemos utilizar algún software de graficación para que nos genere los tres valores).

Veamos una página de ejemplo en la que se elige entre varios colores para cambiar el color del texto. su código será el siguiente:

```
<!DOCTYPE html>

<html>
<head>
<meta charset="utf-8">
<title>JavaScript – Comprobar igualdad de contraseñas en un registro de usuario nuevo</title>

<script type="text/javascript">

window.onload = function() {

document.fgColor = "blue"

}

function cambia() {

    opciones = document.formulario.color; //acceso a los botones radio

    for (i=0;i<opciones.length;i++) { //busca valor del botón pulsado

        if (opciones[i].checked == true) {

            elegido = opciones[i].value

        }

    }

    switch (elegido) { //elegir color según boton pulsado

        case "rojo":

            document.fgColor = "red";

            break;

        case "verde":

            document.fgColor = "green";

            break;

        case "purpura":

            document.fgColor = "purple";

            break;
```

```
case "oliva":

    document.fgColor = "olive"

    break;

case "marron":

    document.fgColor = "maroon"

    break;

case "azul_oscuro":

    document.fgColor = "navy"

    break;

case "azul_marino":

    document.fgColor = "teal"

    break;

default:

    document.fgColor = "blue"

    break;

}

}

</script>

</head>

<body>

<h1>Cambiar de color el texto de esta página</h1>

    <p>Usaremos la estructura switch de javascript para cambiar de color el
texto de esta página. Elige un color y pulsa luego el botón cambiar.</p>

<form action="#" name="formulario" onreset="document.fgColor='blue'">

    <input type="radio" name="color" value="rojo" /> Rojo<br/>

    <input type="radio" name="color" value="verde" /> Verde<br/>

    <input type="radio" name="color" value="purpura" /> Purpura<br/>
```

```
<input type="radio" name="color" value="oliva" /> Oliva<br/>
<input type="radio" name="color" value="marron" /> Marron<br/>
<input type="radio" name="color" value="azul_oscuro" /> Azul oscuro<br/>
<input type="radio" name="color" value="azul_marino" /> Azul Marino<br/><br/>
<input type="button" onclick="cambia()" value="Cambia" /><br/><br/>
<input type="reset" value="reiniciar formulario"/> (color azul)

</body>
</html>
```



Comprueba el código anterior desde el simulador online de código:

<http://elprofedemicurso.es/visorjavascript/llamada.html>

3.8. Bucles

Un bucle es un conjunto de comandos que se ejecutan repetitivamente un cierto número de veces.

3.8.1. for

Permite un bucle repetitivo sabiendo de antemano el número de ejecuciones que será necesario.

Sintaxis:

```
for ([inicial;][final;][incremento]) {instrucciones }
```

```
Ejemplo: for (i=0; i<4; i++){alert("Ahora van "+i)}
```

3.8.2. while

Permite un bucle repetitivo cuyo número de repeticiones dependerá de una condición. Aquí normalmente no sabemos de antemano el número de repeticiones.

Sintaxis:

```
while (condicion) { instrucciones }
```

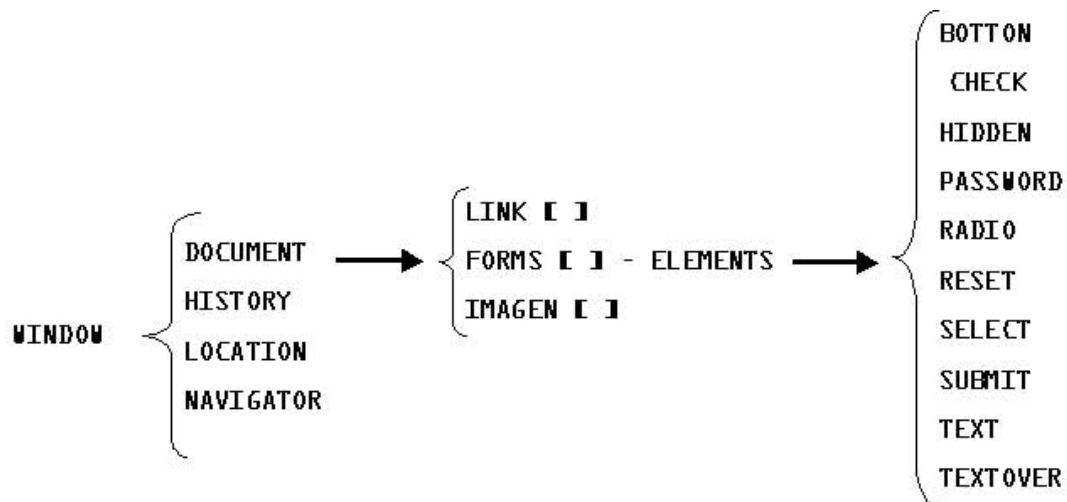
3.8.3. Control de bucle

Tenemos dos comandos para el control de bucles: **break** que termina el bucle y transfiere el control del programa a la siguiente instrucción a continuación del bucle. **continue** interrumpe la ejecución de comandos y regresa el control al inicio del bucle.

3.9. Objetos

Existen objetos predefinidos, cada uno con distintas propiedades, métodos y manejadores de eventos. A diferencia de JAVA aquí no se puede derivar clases, por lo que se habla de una jerarquía de instancia.

3.9.1. Jerarquía de Objetos



3.9.2. Objeto Window:

| window | Es el objeto de nivel superior para cada grupo de objetos document, location e history. | |
|-------------|---|--|
| Propiedades | | |
| | DefaultStatus | Mensaje por omision en la barra de estado de la ventana |
| | document | como document |
| | Frame | como Frame |
| | frames | Es un arreglo que contiene todos los recuadros de la ventana |
| | length | Indica el numero de recuadros de la ventana madre |
| | location | como location |
| | name | refleja el argumento nombre con que se creo la ventana |
| | parent | Es un sinonimo del argunmento nombre y se refiere a una ventana con recuadros |
| | self | Es un sinonimo del argumento nombre y se refiere a la ventana en uso |
| | status | Especifica un mensaje a presentar en la barra de estado de la ventana. <> |
| | top | Es un sinonimo del argumento nombre y se refiere a una ventana de nivel superior del navegador. |
| | window | Es un sinonimo del argumento nombre y se refiere a la ventana en uso |
| Metodos | | |
| | open | Abre una nueva ventana en tu navegador, con una pagina en blanco o la URL que tu indiques. <> |
| | close | Cierra la ventana activa o indicada. <> |

| | | |
|---------|--------------|--|
| | alert | Muestra una caja de alerta con un mensaje y el botón de aceptar. < > |
| | confirm | Muestra una caja de información con un mensaje y los botones aceptar y cancelar. < > |
| | prompt | Muestra una caja de dialogo con un mensaje y un campo para entrada de datos y los botones aceptar y cancelar. < > |
| | setTimeout | Evalúa una expresión después de transcurrido un tiempo en milisegundos |
| | clearTimeout | |
| Eventos | | |
| | onLoad | Se ejecuta cuando el navegador termina de cargar una ventana o todos los recuadros indicador en <FRAMESET> |
| | onUnload | Se ejecuta cuando el navegador descarga la página. |

3.9.3. Objeto Document

| | |
|----------------|---|
| METODOS | <p style="text-align: center;">W R I T E</p> <p>Escribe una o más expresiones de HTML en el documento contenido en la ventana indicada.</p> <p style="text-align: center;">Document.write(exp1[,exp2]...[,expN])</p> |
| | <p style="text-align: center;">W R I T E L N</p> <p>Escribe una o más expresiones de HTML en el documento contenido en la ventana indicada, seguidas de un carácter de nueva línea.</p> <p style="text-align: center;">Document.writeln(exp1[,exp2]...[,expN])</p> |

| | |
|-------------|---|
| PROPIEDADES | B G C O L O R Corresponde al parámetro BGCOLOR del comando "BODY". |
| | F G C O L O R Corresponde al parámetro TEXT del comando "BODY" |
| | L I N K C O L O R Corresponde al parámetro LINK del comando "BODY" |
| | V L I N K C O L O R Corresponde al parámetro VLINK del comando "BODY" |
| | L O C A T I O N URL completo del documento |

3.9.4. Objeto String

| | | |
|---------------|--|--|
| string | Es una cadena de caracteres encerrados entre comillas simples o dobles. Es un objeto básico en javascript. | |
| Propiedades | length | Indica la longitud de la cadena. Sintaxis: string.length(nombre) |
| Métodos | | |
| | anchor | Crea un marcador HTML |
| | big | Hace que una cadena se muestre con tipos grandes, como si estuviera entre comandos <BIG> |
| | blink | Hace que una cadena parpadee, como si estuviese entre comandos <BLINK> |
| | bold | Hace que una cadena este en negritas, como si estuviese entre comandos |
| | charAt | Devuelve el carácter en la posición indicada (0 a length-1) de la cadena |
| | fixed | Hace que la cadena se muestre como si estuviese |

| | | |
|---------|-------------|---|
| | | entre comandos <TT> |
| | fontcolor | Hace que la cadena se muestre en el color indicado, como si estuviese entre comandos |
| | fontsize | Hace que la cadena se muestre del tamaño indicado, como si estuviese entre comandos |
| | indexOf | Devuelve la posición dentro de la cadena donde se encuentra el texto de búsqueda. De izquierda a derecha. |
| | italics | Hace que la cadena se muestre en cursiva, como si estuviese entre comandos <I> |
| | lastIndexOf | Devuelve la última posición dentro de la cadena en que se encuentra el texto de búsqueda. De derecha a izquierda. |
| | link | Crea un enlace de hipertexto que apunta a otro URL |
| | small | Hace que una cadena se muestre con tipos pequeños, como si estuviera entre etiquetas <SMALL> |
| | strike | Hace que una cadena se muestre atravesada, como si estuviese entre etiquetas <STRIKE> |
| | sub | Hace que una cadena se muestre en subíndice, como si estuviese entre etiquetas <SUB> |
| | substring | Devuelve una subcadena de la cadena indicada, entre las posiciones indicadas. |
| | toLowerCase | Devuelve la cadena convertida en minúsculas |
| | toUpperCase | Devuelve la cadena convertida en mayúsculas |
| Eventos | ninguno | |

3.9.5. Objeto Math

| | | |
|-------------|--|--|
| math | Es un objeto predefinido que contiene propiedades y métodos para constantes y funciones matemáticas. | |
| Propiedades | | |
| | E | La constante de EULER. Aproximadamente 2.71818 |
| | LN2 | El logaritmo natural de 2. Aproximadamente 0.693 |
| | LN10 | El logaritmo natural de 10. Aproximadamente 2.302 |
| | LOG2E | El logaritmo en base 2 de e. Aproximadamente 1.442 |
| | LOG10E | El logaritmo vulgar de e. Aproximadamente 0.434 |
| | PI | La constante PI. Aproximadamente 3.14159 |
| | SQRT1_2 | La raíz cuadrada de 1/2. Aproximadamente 0.707 |
| | SQRT2 | La raíz cuadrada de 2. Aproximadamente 1.414 |
| Metodos | ninguno | |
| Eventos | ninguno | |

3.9.6. Objeto Date

| | |
|-------------|---|
| Date | Permite trabajar con fechas y horas. Para crear nuevos objetos se puede usar: |
|-------------|---|

| | | |
|-------------|---|--|
| | <ul style="list-style-type: none"> • variable=new Date() • variable=new Date("mes día año horas:minutos:segundos") • variable=new Date(año,mes,día) • variable=new Date(año,mes,día,horas,minutos,segundos) | |
| Propiedades | ninguna | |
| Métodos | getDate | el día del mes (1 a 31) |
| | getDay | el día de la semana (0=dom, 1=lun...6=sábado) |
| | getMonth | el mes en números (0=ene, 1=feb...11=dic) |
| | getFullYear | el año con dos dígitos |
| | getHours | la hora (0 a 24) |
| | getMinutes | los minutos (0 a 59) |
| | getSeconds | los segundos (0 a 59) |
| | getTime | los milisegundos transcurridos desde el 01-01-1970 |
| | getTimezoneoffset | diferencia horaria con GMT en minutos |
| | Sintaxis: variable.metodo() | |
| | setDate | Establece el día del mes (1 a 31) |
| | setMonth | Establece el mes del año (1 a 11) |
| | setYear | Establece el año a partir de 1900 |
| | setHours | Establece la hora del día (0 a 23) |
| | setMinutes | Establece los minutos (0 a 59) |
| | setSeconds | Establece los segundos (0 a 59) |
| | setTime | Establece el valor del objeto Date, en milisegundos a partir de las 0:00:00 del 1º de enero de 1970 |
| | Sintaxis: variable.metodo(valor) | |
| | toGMTString | Convierte un objeto Date a una cadena usando un formato GMT para Internet |
| | toLocaleString | Convierte un objeto Date a una cadena usando el formato local |
| | parse | Convierte una cadena representando una fecha al tiempo en milisegundos a partir de 0:00:00 del 1º de enero de 1970 |
| | UTC | Convierte una fecha al tiempo en milisegundos a partir de las 0:00:00 del 1º de enero de 1970 |
| Eventos | ninguno | |

3.9.7. Objeto Array

| | |
|-----------------------------|--|
| Array | <p>Permite trabajar con arreglos.</p> <p>Ejemplo:</p> <p>Vector=new array().</p> <p>Vector =["rojo", "verde","azul"]</p> |
| array.length | Nos da el número de elementos que tiene el arreglo. |
| array.push("ultimo") | Es como una pila, pone el elemento al final del arreglo. |
| array.pop() | Quita un elemento del arreglo. |
| array.shift() | Quita el primer elemento del arreglo. |
| array.unshift() | Quita un elemento en la primera posición del arreglo. |
| array.join(":") | Separador de Campo. |
| array.sort() | Ordena ascendentemente los elementos. |
| array.reverse() | Invierte las posiciones que tenemos. |

3.9.8. Objeto Form

Este objeto permite manipular todos los formularios y sus elementos.

```
window.document.forms[]
```

```
window.document.forms[0].elements[0]
```

Propiedades

| | | |
|------------|---|---|
| elements[] | Esta propiedad tiene todos los elementos de un formulario | window.document.forms[0].elements[0].value |
| action | especifico en el programa que quiero que se ejecute | <form action: 'ingreso.php' method="post" name: "formula1"> |

| | | |
|--------|---|---|
| length | Nos da el número de elementos del formulario. | window.document.forms[0].elements[0].length |
| name | nos da el nombre del elemento deseado | window.document.forms[0].elements[0].name |

Métodos

| |
|----------|
| reset() |
| submit() |

3.9.9. Objeto History

| | | |
|----------------|---|--|
| history | Contiene la información de los URLs que el cliente ha visitado desde una ventana. | |
| Propiedades | Indica la cantidad de entradas en el objeto history. sintaxis: history.length | |
| Metodos | | |
| | back | retrocede al URL anterior. <> |
| | forward | avanza al URL siguiente (después de haber retrocedido). |
| | go | desplaza al URL indicado en relación a la actual posición, hacia adelante (+) o hacia aras (-). Sintaxis: history.go(desplazamiento) |
| Eventos | ninguno | |

3.9.10. Objeto Number

| | |
|---------------|---|
| Number | <p>Este objeto representa el tipo de dato número con el que JavaScript trabaja. Podemos asignar a una variable un número, o podemos darle valor, mediante el constructor Number, de esta forma:</p> <p>a = new Number(valor); ,</p> <p>por ejemplo,</p> <p>a = new Number(3.2); da a a el valor 3.2.</p> <p>Si no pasamos algún valor al constructor, la variable se inicializará con el valor 0.</p> |
|---------------|---|

| | |
|--|---|
| | <p>Propiedades</p> <ul style="list-style-type: none"> • MAX_VALUE. Valor máximo que se puede manejar con un tipo numérico • MIN_VALUE. Valor mínimo que se puede manejar con un tipo numérico • NaN. Representación de un dato que no es un número • NEGATIVE_INFINITY. Representación del valor a partir del cual hay desbordamiento negativo (underflow) • POSITIVE_INFINITY. Representación del valor a partir del cual hay desbordamiento positivo (overflow) <p>Para consultar estos valores, no podemos hacer:</p> <pre>a = new Number(); alert(a.MAX_VALUE);</pre> <p>porque JavaScript nos dirá undefined, tenemos que hacerlo directamente sobre Number, es decir, tendremos que consultar los valores que hay en Number.MAX_VALUE, Number.MIN_VALUE, etc.</p> |
|--|---|

3.9.11. Objeto Boolean

| | |
|----------------|--|
| Boolean | <p>Este objeto nos permite crear booleanos, esto es, un tipo de dato que es cierto o falso, tomando los valores true o false. Podemos crear objetos de este tipo mediante su constructor. Veamos varios ejemplos:</p> <pre>a = new Boolean(); asigna a 'a' el valor 'false' a = new Boolean(0); asigna a 'a' el valor 'false' a = new Boolean(""); asigna a 'a' el valor 'false' a = new Boolean(false); asigna a 'a' el valor 'false' a = new Boolean(numero_distinto_de_0); asigna a 'a' el valor 'true' a = new Boolean(true); asigna a 'a' el valor 'true'</pre> |
|----------------|--|

3.9.12. Último nivel de Objetos

3.9.12.1. Text-Password

| | |
|--------------------|---|
| Propiedades | <p>Disable</p> <p>Maxlength</p> <p>Name</p> |
|--------------------|---|

| | | |
|----------------|----------|--|
| | Readonly | |
| | Size | |
| | Type | |
| | Value | |
| Métodos | Focus() | |
| | Blur() | |
| | | |

3.9.12.2. Text Area

| | | |
|--------------------|----------|--|
| Propiedades | Cols | |
| | Disbled | |
| | Name | |
| | Readonly | |
| | Rows | |
| | Type | |
| | Value | |
| Métodos | Focus() | |
| | Blur() | |
| | | |

3.9.12.3. Button

| | | |
|--------------------|---------|--|
| Propiedades | Disbled | |
| | Name | |
| | Type | |
| | Value | |
| Métodos | Focus() | |
| | Blur() | |
| | Click() | |
| | | |

3.9.12.4. Check

| | | |
|--------------------|---------|--|
| Propiedades | Disbled | |
| | Name | |
| Métodos | Checked | |
| | Type | |
| | Value | |
| | Focus() | |
| | Blur() | |
| | Click() | |
| | | |

3.9.12.5. Radio

| | | |
|--------------------|---------|--|
| Propiedades | Disbled | |
| | Name | |
| Métodos | Checked | |
| | Type | |
| | Value | |
| | Focus() | |
| | Blur() | |
| | Click() | |
| | | |

3.9.12.6. Select

| | | |
|--------------------|---------|--|
| Propiedades | Disbled | |
| | Length | |

| | | |
|----------------|---------|--|
| | Size | |
| | Name | |
| | Type | |
| | Value | |
| Métodos | Focus() | |
| | Blur() | |
| | | |
| | | |

3.9.12.7. Reset

| | | |
|--------------------|---------|--|
| Propiedades | Disbled | |
| | Name | |
| | Type | |
| | Value | |
| Métodos | Focus() | |
| | Blur() | |
| | Focus() | |
| | | |

3.9.12.8. Submit

| | | |
|--------------------|---------|--|
| Propiedades | Disbled | |
| | Name | |
| | Type | |
| | Value | |
| Métodos | Focus() | |
| | Blur() | |
| | Focus() | |

3.10. Algo más en JAVASCRIPT...

| 3.10.1. Eventos En JavaScript | |
|-------------------------------|---|
| | OnBlur: Acción al abandonar el Foco |
| | OnClick: Acción al hacer un click sobre el botón. |
| | OnChange: Acción al cambiar Texto. |
| | OnFocus: Acción al llegar a la casilla. |
| | OnReset: Realiza la Acción de resetear lo que se ha hecho. |
| | Submit: Se acciona con un botón submit. |
| | Unload: Acción al abandonar el documento. |
| | OnDbClick: Acción al hacer dos click sobre el botón |
| | OnMouseOver: Acción al pasar el ratón por encima de algo. |
| | OnLoad: Acción al cargar el documento. |

3.10.2. Métodos Globales.

JavaScript incluye las siguientes funciones, que no son métodos de ningún objeto sino propias del lenguaje:

- **eval**

Trata de evaluar una cadena y devolver un valor numérico, si el argumento es una expresión, la expresión se evalúa, si el argumento consiste en uno o más comandos, se ejecutan.
Sintaxis: eval(cadena)

- **parseFloat**

Convierte una cadena a un número en punto flotante. Si se encuentra un carácter que no es número, signo (+ o -), punto decimal o exponente, la función ignora la cadena a partir de esa posición y la evalúa hasta el carácter anterior. Si el primer carácter no se puede convertir, la función devuelve uno de estos valores: o en las plataformas Windows y "NaN" (Not a Number) para otras plataformas.

Sintaxis: parseFloat(cadena)

- **parseInt**

Convierte una cadena a un entero en la base especificada. Si no se especifica la base o se especifica como 0, se opta por lo siguiente: Si la cadena comienza con "0x", la base es 16 (hexadecimal), si la cadena empieza con 0, la base es 8 (octal), si la cadena comienza con otro valor, la base es 10 (decimal). Si se encuentra un carácter que no es numérico, la función ignora la cadena a partir de esa posición y la evalúa hasta la anterior. Si el primer carácter no se puede convertir, la función devuelve uno de estos valores: 0 para plataformas Windows y "NaN" (Not a Number) para otras plataformas.

Sintaxis: `parseInt(cadena [,base])`

- **isNaN**

Evalúa un argumento para determinar si es "NaN", en plataformas UNIX, devolviendo un valor Booleano true o false.

Sintaxis: `isNaN(valor prueba)`.

- **getElementById y setTimeout**

- **getElementById()**

| | |
|-----------------------|---|
| getElementById | Permite referirnos a los elementos por su identificador ID y modificarlos. |
| Estructura: | <p><code>document.getElementById(id).style.propiedad</code></p> <p>Ejemplo de Sintaxis:</p> <ul style="list-style-type: none"> <code>document.getElementById('imagen').style.backgroundColor</code> |

- **(setTimeout)**

| | |
|-------------------|---|
| setTimeout | <p>2. Evalúa una expresión después de transcurrido un tiempo en milisegundos.</p> <p>3. Este es un método del Objeto Window</p> |
|-------------------|---|

3.11. Correspondencia entre propiedades CSS y propiedades DOM

| Propiedad CSS | Propiedad DOM en Javascript |
|---------------|-----------------------------|
| background | background |

| | |
|-----------------------|----------------------|
| background-attachment | backgroundAttachment |
| background-color | backgroundColor |
| background-image | backgroundImage |
| background-position | backgroundPosition |
| background-repeat | backgroundRepeat |
| border | border |
| border-color | borderColor |
| border-style | borderStyle |
| border-top | borderTop |
| border-right | borderRight |
| border-left | borderLeft |
| border-bottom | borderBottom |
| border-top-color | borderTopColor |
| border-right-color | borderRightColor |
| border-bottom-color | borderBottomColor |
| border-left-color | borderLeftColor |
| border-top-style | borderTopStyle |
| border-right-style | borderRightStyle |
| border-bottom-style | borderBottomStyle |
| border-left-style | borderLeftStyle |
| border-top-width | borderTopWidth |
| border-right-width | borderRightWidth |
| border-bottom-width | borderBottomWidth |
| border-left-width | borderLeftWidth |
| border-width | borderWidth |
| clear | clear |
| clip | clip |
| color | color |
| display | display |
| float | cssFloat |
| font | font |
| font-family | fontFamily |
| font-size | fontSize |
| font-style | fontStyle |
| font-variant | fontVariant |
| font-wight | fontWight |
| height | height |
| left | left |
| letter-spacing | letterSpacing |
| line-height | lineHeight |
| list-style | listStyle |
| list-style-image | listStyleImage |
| list-style-position | listStylePosition |
| list-style-type | listStyleType |
| margin | margin |
| margin-top | marginTop |
| margin-right | marginRight |
| margin-bottom | marginBottom |
| margin-left | marginLeft |
| overflow | overflow |

| | |
|-----------------|----------------|
| padding | padding |
| padding-top | paddingTop |
| padding-right | paddingRight |
| padding-bottom | paddingBottom |
| padding-left | paddingLeft |
| position | position |
| text-align | textAlign |
| text-decoration | textDecoration |
| text-indent | textIndent |
| text-transform | textTransform |
| top | top |
| vertical-align | verticalAlign |
| visibility | visibility |
| white-space | whiteSpace |
| width | width |
| word-spacing | wordSpacing |
| z-index | zIndex |

3.12. Palabras reservadas

Dentro de JavaScript nos encontramos con las siguientes palabras reservadas (las cuales no podremos usar como nombre de variables):

| | | | |
|----------|--------------|------------|------------|
| abstract | boolean | break | byte |
| case | catch | char | class |
| const | continue | default | do |
| double | else | extends | false |
| final | finally | float | for |
| function | goto | int | implements |
| input | in | instanceof | interface |
| long | native | new | null |
| package | private | protected | public |
| return | short | static | super |
| switch | synchronized | this | throw |
| throws | transient | true | try |
| var | val | while | with |

3.13. Manipulación de objetos

Hay varias instrucciones para manejar objetos

- **new**

Permite crear un nuevo objeto de un tipo definido por el usuario.

Sintaxis:

```
variable=new tipo (parametri1 [,parametro2]...[,parametro n])
```

- **this**

Se usa para definir al objeto en uso, por lo general el que efectúa una llamada, al definir un método.

Sintaxis:

```
this[.propiedad]
```

- **for..in**

Itera una variable a lo largo de todas las propiedades de un objeto, para cada propiedad. JavaScript ejecuta las instrucciones especificadas.

Sintaxis:

```
for (variable in objeto) { instrucciones }
```

- **with**

Establece el objeto por omisión de una serie de instrucciones; si no especifica en las propiedades el objeto, se asume el indicado con with.

Sintaxis:

```
with (objeto) { instrucciones }
```

- **Comentarios**

Los comentarios son líneas que coloca el autor para propósitos explicativos dentro de un programa. El intérprete ignora los comentarios.

- **Comentarios de una línea**

Sintaxis:

```
// comentario
```

- **Comentarios de varias líneas**

Sintaxis:

```
/* comentario */
```

3.14. Enlaces sobre JavaScript y JQuery.

| Recurso |
|---|
| Sobre Javascript |
| http://www.aulafacil.com/cursos/l7296/informatica/programacion/javascript/introduccion |
| http://librosweb.es/libro/javascript/ |
| http://www.desarrolloweb.com/javascript/ |
| http://www.javascriptya.com.ar/ |
| Sobre JQuery y jquery mobile |
| http://www.jtech.ua.es/dadm/2011-2012/restringido/web/sesion03-apuntes.html |
| http://www.desarrolloweb.com/manuales/manual-jquery-mobile.html |