JAVASCRIPT

Introducción.

JavaScript es un lenguaje multiplataforma orientado a objetos, liviano, que te permite realizar actividades complejas en una página web.

Es un lenguaje de programación interpretado, por lo que no es necesario hacer nada con estos programas, ni tan siquiera compilarlos. Los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

Hoy en día, las páginas hacen más que solo mostrar información estática o imágenes, también permiten actualización de partes de su contenido en el momento, posibilitan interactuar con mapas y gráficas de manera online y brindan al usuario una experiencia de navegación mucho más amena y dinámica.

Debido a su propósito y uso general, todos los navegadores web modernos interpretan correctamente JavaScript y su uso está por lo general relacionado con la interfaz de usuario que presenta un sitio web.

Permite crear en una página web, elementos como cuadros de diálogo, recolectores de información de parte del usuario y pequeños procesos de seguridad sobre los datos enviados al servidor.

Por ser un lenguaje abierto, existen variantes, como por ejemplo la versión de Microsoft conocida como JScript que utilizan tanto el navegador Internet Explorer como el sistema operativo Windows bajo el nombre de Windows Scripting Host.

Pese a ser utilizado del lado del cliente, también existen implementaciones del lado del servidor que son ampliamente utilizadas, como por ejemplo Node.js.

A pesar de todos los cambios y actualizaciones, JavaScript mantiene la compatibilidad hacia versiones anteriores.

Páginas web dinámicas.

Una página web dinámica, es una página que va actualizando su contenido a medida que el usuario va realizando peticiones, ya sea navegando por la propia página, o actualizando todo o parte de su contenido.

A diferencia de las páginas estáticas, que tienen su contenido predeterminado, las páginas dinámicas, la información tiene un ciclo de vida determinado por el usuario.

Esto es posible porque la página tiene asociada una aplicación web o una base de datos que le permite modificar su contenido a demanda.

Los principales lenguajes para el desarrollo de este tipo de páginas web son ASP, PHP, JSP y por supuesto JavaScript.

Programa, aplicación y sistema.

Mucho antes del uso masivo de internet del que gozamos hoy en día, era habitual adquirir programas para instalar en nuestras computadoras personales.

Estos programas iban desde lo más elemental, como un editor de texto a desarrollos verdaderamente complejos, como el AutoCAD o completos paquetes que integraban varios programas en un solo entorno para el usuario.

Un programa informático es un conjunto de instrucciones que le damos a la computadora para que las ejecute.

Con el tiempo, empezaron a hacerse populares los dispositivos móviles, como las tablets y los teléfonos inteligentes, y con ellos se popularizaron las tiendas de aplicaciones.

¿Cuál es la diferencia entre un programa y una aplicación?
Las aplicaciones son programas informáticos con un objetivo concreto, por ejemplo, en nuestros teléfonos tenemos una aplicación destinada a escanear documentos, otra a decirnos el clima en nuestra ciudad, etc. En general son de un tamaño mucho menor al de un programa y consumen menos recursos, lo que las hace ideales para este tipo de dispositivos.
Su principal función es la de ayudar al usuario en la realización de una determinada tarea, a diferencia de los programas, que en general requieren de un curso o capacitación para poder utilizarlos, las aplicaciones son por demás intuitivas y muy sencillas de utilizar.

Por último, tenemos a los sistemas. Es importante no confundir sistemas informáticos con sistemas operativos.
Un sistema operativo, es el conjunto de programas destinados a administrar los recursos de la computadora o más específicamente, los recursos de hardware del equipo, ya sea PC, Tablet, teléfono, etc.
Un sistema informático, incluye al sistema operativo y a las aplicaciones o programas y en general también suelen incluir los recursos humanos necesarios para su funcionamiento.
Básicamente es el conjunto de software (aplicaciones, programas, sistema operativo) y hardware.

Hoy en día es difícil que como desarrolladores nos pidan algún trabajo que no se relacione con otros sistemas, con bases de datos, redes sociales u otros desarrollos en la nube.
Este tipo particular de aplicaciones, son las que llamamos Aplicaciones Web, y suelen tener tres capas bien definidas:

1- El modelo de datos.
2- La interfaz de usuario.
3- La lógica que relaciona los puntos 1 y 2.

El lenguaje JavaScript.

JavaScript (que no debe ser confundido con Java) fue creado en mayo de 1995 por Brendan Eich, programador en Netscape.
El nombre original del lenguaje era Mocha, elegido por uno de los fundadores de Netscape.
En mayo de ese mismo año se cambió a LiveScript y finalmente, en el mes de diciembre, se lo bautizó JavaScript.
En 1996 el lenguaje fue llevado a ECMA, una organización internacional dedicada, entre otras cosas, a estandarizar el uso de las tecnologías de información y comunicación de dispositivos electrónicos, con la idea de hacer del lenguaje, un estándar que pudiera ser utilizado por otros proveedores de navegadores web.
Esto llevo a la liberación oficial de ECMAscript, que es el nombre oficial de la norma.
Define un lenguaje de tipos dinámicos, ligeramente inspirado en Java y en lenguaje C, soporta algunas características de la programación orientada a objetos, mediante el uso de prototipos y pseudoclases.
Cada navegador tiene su propia implementación del estándar ECMAscript, al igual que soporte para el acceso al Document Object Model (DOM) que facilita la manipulación de las páginas web.
Mientras todo esto sucedía y JavaScript se imponía como un estándar, las diferentes comunidades de desarrolladores comenzaron a trabajar para sacarle el máximo provecho al lenguaje, y así nació, en 2005, un conjunto de tecnologías, de las cuáles JavaScript era el núcleo principal, denominadas Ajax, cuyo mayor

uso se hacía en las aplicaciones web, permitiendo cargar los datos de la página en segundo plano, evitando la necesidad de cargar la página completa, dando como resultado, páginas mucho más dinámicas.
Actualmente, JavaScript continúa evolucionando y agregando nuevas tecnologías, como por ejemplo la plataforma Node.js que nos permite implementar este lenguaje del lado del servidor.

JavaScript y HTML.

La integración entre JavaScript y HTML es muy simple y variada, ya que tenemos, al menos tres maneras de hacerlo.

1) Escribir JavaScript en el propio documento HTML.
   Podemos escribir nuestras sentencias entre las etiquetas <script> </script> y se pueden incluir en cualquier parte del documento, aunque lo usual es incluir todo el bloque de código dentro de la cabecera del documento, es decir, dentro de la etiqueta <head>.
   Para que la página web sea correctamente interpretada, deberá agregarse el atributo *type* a la etiqueta <script>, y los valores a los que se iguala este atributo están estandarizados, para el caso de JavaScript, el valor correcto es *text/javascript.*
   Este método se utiliza cuando solo necesitamos un pequeño bloque de código que sea específico para el documento en donde se lo incluye, por lo que no será reutilizable, y si llegáramos a incluir dicho código en más de un documento, se nos complicará el mantenerlo, ya que para cualquier tipo de modificación en el mismo, tendremos que modificar todos los documentos en los que lo hayamos utilizado.
   A continuación, un ejemplo de cómo utilizarlo:

```
<!DOCTYPE html>
<html>
    <head>

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

    </head>

    <body>
        <button onclick="mensaje()">Probar mensaje</button>
    </body>
</html>
```

Imagen 1 - 1

Así se ve por pantalla:

Imagen 1 - 2

2) Incluir JavaScript desde un archive externo.
Nuestras sentencias se pueden escribir y guardar en un archivo con la extensión .js que luego será incorporada al documento HTML mediante la etiqueta <script>.
Se pueden incluir en un documento HTML tantos archivos JavaScript como sean necesarios.
En este caso, además del atributo *type* será necesario también, asignarle un valor al atributo *src* que es el que contiene la URL correspondiente al archivo JavaScript que deseamos incorporar.
Cada etiqueta <script> puede definir la ruta a un único archivo, pero podemos incorporar tatas etiquetas como sean necesarias.
Un archivo con extensión .js es un archivo de texto normal que se puede crear o editar con cualquier editor de texto.
La mayor ventaja de este método, es la reutilización del código, lo que facilita, como vimos en el punto anterior, el mantenimiento de nuestras aplicaciones.
Por ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript" src="/js/codigo.js">
        </script>
    </head>

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

Imagen 1 - 3

3) Incluir código JavaScript dentro de las etiquetas HTML.
   Es la forma menos utilizada y consiste en crear bloques de código JavaScript dentro de las propias etiquetas HTML para afectarlas en su comportamiento.
   Sus principales desventajas son la no reutilización del código y la manera de ensuciar innecesariamente nuestro código HTML.

```
<!DOCTYPE html>
<html>
    <head>
    </head>

    <body>
        <p onclick="alert('Otro mensaje de prueba')">
            Un texto cualquiera.
        </p>
    </body>
</html>
```

Imagen 1 - 4

Y así se ve por pantalla si presionamos sobre el texto:

Imagen 1 - 5

Capítulo 2. Variables. Salida por pantalla. Tipos de datos.

Herramientas de trabajo.

Para escribir y probar nuestro código vamos a hacer uso de dos herramientas muy simples.

- Editor de texto (Notepad++, VS Code, SublimeText, etc).
- Navegador Web.

Es recomendable crear una carpeta de trabajo donde iremos alojando los diferentes archivos que vayamos creando.
En general iremos probando nuestras primeras líneas de código dentro de un archivo HTML como el de la siguiente imagen:

```
<!DOCTYPE html>
<html>
    <head>
    
    </head>

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

Imagen 2- 1

El archivo lo veremos con cualquier navegador web que tengamos instalado.

Sintaxis.

La sintaxis de JavaScript es muy similar a la de otros lenguajes, veamos las principales características:

- No importan las nuevas líneas y los espacios en blanco al igual que sucede en HTML ya que el propio intérprete del lenguaje los ignora.

- Es case sensitive: distingue entre las mayúsculas y minúsculas.

- A diferencia de otros lenguajes de programación no se define el tipo de las variables (var). En JavaScript nunca sabemos el tipo de datos que va a contener una variable por lo que una misma variable puede almacenar diferentes tipos de datos.

- Cada sentencia en JavaScript acaba con el carácter; (punto y coma), aunque no en necesario ya que el intérprete lee cada sentencia, aunque no exista este carácter. Por convenio deberíamos incluirlo.

- Existe la opción de incluir comentarios para añadir información en el código fuente del programa. Estos comentarios suelen servir para dar información al propietario del código u otro desarrollador sobre el contenido del bloque de código en JavaScript. Los comentarios pueden ser de una sola línea o de varias líneas (en bloque).

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            // Comentario de una línea.
            /* Comentario de mas
               de una línea */
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 2

Variables.

Sin las variables sería imposible escribir y crear "programas genéricos", es decir, códigos que funcionan de la misma manera independientemente de los valores concretos usados. Las variables en JavaScript se utilizan mediante la palabra reservada VAR.

Dicha palabra se utiliza solamente para definir por primera vez una variable, lo que en programación se denomina “declarar una variable”.

A continuación de la palabra VAR debemos escribir el nombre que queremos darle a la variable y luego el valor que queremos asignarle, en caso de querer hacerlo.

Al nombre de la variable se lo conoce como IDENTIFICADOR y debe cumplir con dos reglas muy importantes:

1- El identificador únicamente puede estar formado por números, letras, y los símbolos ‘$’ y ‘_’ a lo sumo.
2- El primer carácter del identificador no debe ser un número.

Por ejemplo, queremos declarar dos variables llamadas valor1 y valor2 y asignarles un valor entero a cada una:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 5;
            var valor2 = 7;
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 3

Recordemos que no es necesario declarar un tipo de dato para las variables, y podemos reutilizar el nombre, incluso con diferentes tipos de datos, por ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 5;
            valor1 = 'Hola mundo';
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 4

Salida por pantalla.

Vamos a ver a continuación como podemos visualizar por pantalla, desde el explorador web, lo que necesitemos mostrar al usuario, por ejemplo, los mensajes de alerta.

Para eso tenemos la función ALERT mediante la cual no solamente podemos mostrar un mensaje, sino también el valor de una o más variables.
Hagamos el ejemplo dentro de un archivo HTML:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            alert('Mensaje al usuario');
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 5

Si guardamos el archivo y lo abrimos con el navegador web, por ejemplo Firefox, veremos el mensaje al usuario:

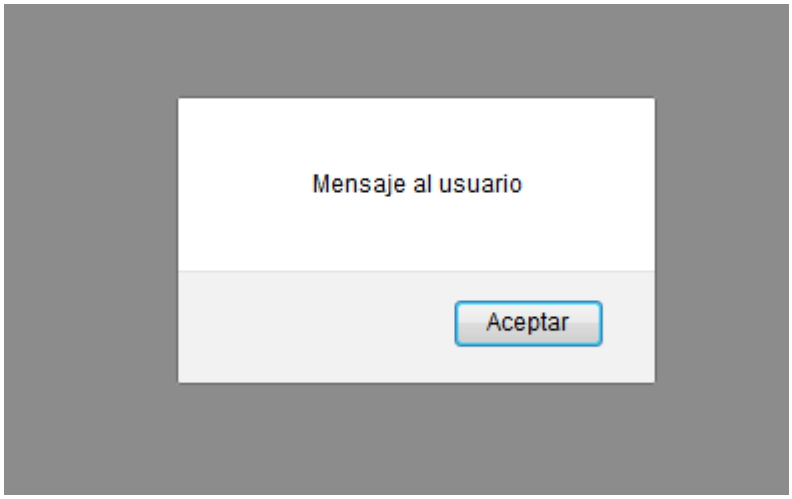


Imagen 2- 6

Veamos un ejemplo de cómo visualizar una variable mediante la función ALERT.
Modificamos nuestro código para declarar una variable, asignarle la frase HOLA MUNDO y mostrarla a l usuario.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje = 'Hola mundo';
            alert(mensaje);
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 7

Y así se ve el mensaje por pantalla:

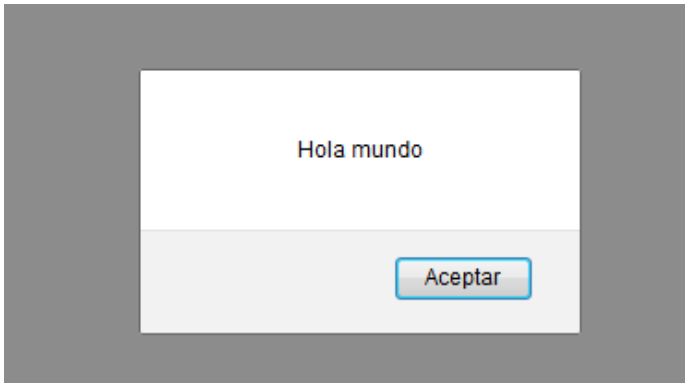


Imagen 2- 8

Veamos como mostrar más de una variable.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje = 'Bienvenido';
            var nombre = 'Pepe';
            alert(mensaje+'\n'+nombre);
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 9

El salto de línea se logra agregando \n entre las variables, así se ve por pantalla:

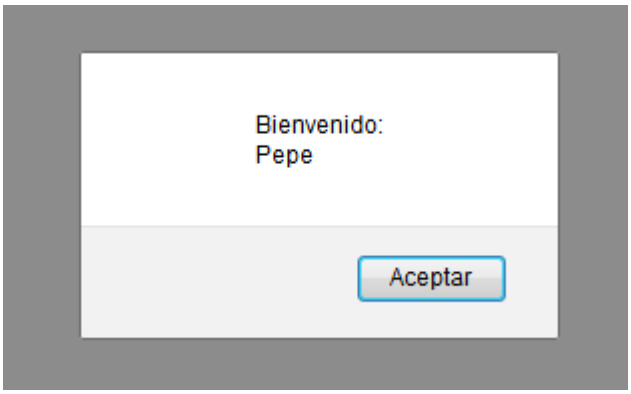


Imagen 2- 10

Otra forma de poder ver los valores que contienen las variables por pantalla, es mediante el uso de la instrucción DOCUMENT.WRITE(texto o variable).
Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje = 'Bienvenido';
            var nombre = 'Pepe';
            document.write(mensaje+'<br/>');
            document.write(nombre);
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2-11

Al agregar la etiqueta de HTML
 la utilizamos para agregar un salto de línea:

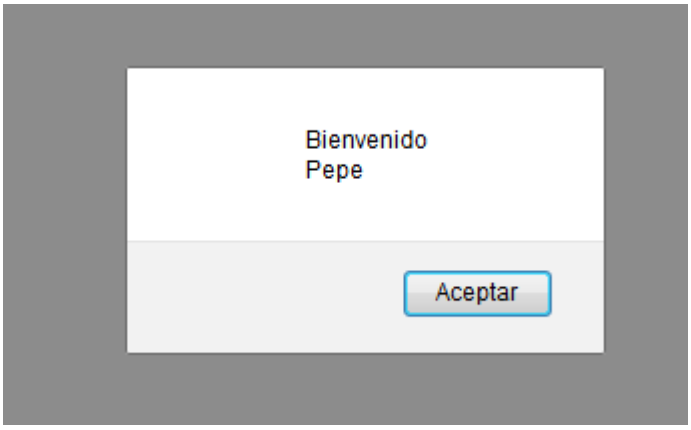


Imagen 2-12

Tipos de datos.

Ya vimos que todas las variables en JavaScript se crean a través de la palabra reservada VAR pero dependiendo de los valores que almacenen pueden ser de un tipo u otro.
Veamos los tipos de datos que podemos utilizar en JavaScript.

Numéricos.

Se usan para contener valores numéricos enteros (llamados integer) o decimales (llamados float) en el cual utilizamos el punto para separar la parte entera de la parte decimal:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var entero = 70;
            var decimal = 123.7;
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 11

Cadenas de texto.

Se usan para contener caracteres alfanuméricos, palabras o frases enteras, como por ejemplo el famoso HOLA MUNDO.

El texto que carguemos en una variable siempre va entre comillas, que pueden ser simples o dobles, como se ve en la imagen a continuación.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje = 'Hola';
            var texto = "Mundo";
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 12

Si quisiéramos que nuestra frase o texto incluyera comillas, lo que deberemos hacer es no utilizar el mismo tipo de comillas para mostrar por pantalla que para encerrar el texto.

Veamos un ejemplo.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            document.write('Muestro "las" comillas dobles'+'<br/>');
            document.write("Muestro 'las' comillas simples");
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2 - 13

En el primer caso, para encerrar todo el texto a mostrar, utilizo las comillas simples, por lo que uso las comillas dobles para mostrarlas por pantalla.

En el segundo caso hago lo contrario. Esto es lo que se ve por pantalla:

Muestro "las" comillas dobles
Muestro 'las' comillas simples

Arrays.

Un array es una colección de variables, sin importar los tipos de los que sean cada una. Sirven para guardar colecciones de valores, de manera que serviría para agrupar diferentes variables. Por ejemplo, tenemos esta sucesión de variables con los días de la semana:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var dia1 = 'Lunes';
            var dia2 = 'Martes';
            var dia3 = 'Miércoles';
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 14

Si quisiéramos manipularlas todas juntas no podríamos hacerlo, tendríamos que buscar la forma de relacionarlas o crear alguna estructura de código compleja que nos permita poder trabajar con todas las variables en un mismo paso.

Para eso existe el ARRAY también conocidos como arreglos o vectores.
Para declararlos procedemos de la misma manera que lo hacíamos con las variables, utilizando la palabra VAR y sin necesidad de especificar ningún tipo de dato.
Esta es la sintaxis básica para declarar un array:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var semana = ['Lunes','Martes','Miercoles'];
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 15

Como siempre, cuando trabajamos con cadenas de texto, debemos usar comillas, ya sean simples o dobles.
Cada uno de los componentes del array se denomina elemento, y recordemos que no es necesario que los elementos de un array sean todos del mismo tipo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var semana = ['Lunes',56,123.7];
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 16

Cada elemento del array ocupa una posición, y estas posiciones se empiezan a contar desde el número 0 (cero).
En nuestro ejemplo de la imagen 2-16, el elemento 'lunes' ocupa la posición cero, el elemento 56 ocupa la posición uno y finalmente, el elemento 123.7 ocupa la posición dos.
A esta posición de cada elemento se la denomina índice, y es precisamente mediante dicho índice que podremos acceder al valor guardado en el array, por ejemplo, para mostrarlo por pantalla:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var semana = ['Lunes',56,123.7];

            document.write(semana[1])
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 2- 17

En nuestro ejemplo de la imagen 2-17 tenemos un array llamado 'semana' con tres elementos y mostramos por pantalla al elemento cuyo índice es el número uno.
Para indicar el índice de un elemento, utilizamos los caracteres [ ].

Booleanos.

Las variables de tipo booleano también son llamadas o denominadas con el nombre de variables de tipo lógico. Estas variables suelen servir para condiciones o para la programación lógica.

Las variables de este tipo almacenan un tipo particular de valor que solo puede tomar dos valores, TRUE o FALSE (verdadero o falso).
No podemos utilizar este tipo de variables para almacenar otro tipo de dato, como por ejemplo, una cadena de texto.

Para ver la utilidad de este tipo de variables es necesario avanzar más en la programación en JavaScript, por lo que más adelante veremos casos prácticos de aplicación.

Capítulo 3. Operadores.

Operadores.

Ya aprendimos a declarar variables y a guardar valores dentro de ellas, pero si solo pudiésemos hacer esto, nos serían de poca utilidad.
Para escribir código de cierta complejidad que brinde una verdadera utilidad al usuario, es necesario recurrir a otro tipo de herramientas, como son los operadores.

Los operadores nos permiten manipular los valores contenidos en las variables y relacionarlas de tal manera que podamos obtener nuevos valores en base a los valores originales o hacernos de información específica que permita a nuestro código tomar una decisión al momento de realizar una u otra acción.

Veamos los principales operadores con los que podemos trabajar en JavaScript.

Asignación.

El operador de asignación sirve para un valor a una variable, es el que estuvimos utilizando cuando vimos variables.
El símbolo utilizado es el =.
A la izquierda del operador de asignación va el nombre de la variable y a su derecha el valor que queremos asignarle:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var nombre = 'Pepe';
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 1

Como ya vimos, podemos asignar a una misma variable, dos tipos diferentes de datos, es decir, podemos reutilizar el nombre, siempre y cuando sepamos que una variable solo va a tener un valor a la vez.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var nombre = 'Pepe';
            document.write('Variable nombre = '+nombre+'<br/>');
            nombre = 'Ricardo';
            document.write('Variable nombre = '+nombre);
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 2

Si guardamos el archivo y lo vemos usando el navegador, veremos por pantalla que la misma variable, tenía asignada una cadena de texto y luego le asignamos un valor numérico.

Variable nombre = Pepe
Variable nombre = Ricardo

Además de poder asignarle a una variable un determinado valor, también podemos asignarle el contenido de otra variable.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 14;
            var valor2 = 'Hola mundo';

            valor1 = valor2;

            document.write(valor1);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 3

Analicemos el código.
Declaramos dos variables llamadas valor1 y valor2.
A valor1 le asignamos un valor numérico, más específicamente, el número 14.
A valor2 le asignamos una cadena de texto, 'hola mundo.'.
Luego, le asignamos a valor1 (que hasta este momento contiene un valor numérico) el contenido de la variable valor2.
El resultado obtenido es que cargamos la cadena de texto contenida en la variable valor2, dentro de la variable valor1.
Por último, mostramos la variable valor1 por pantalla y esto es lo que vemos:

hola mundo.

Incremento y decremento.

Estos dos operadores se utilizan solo con variables del tipo numéricas, y sirven para aumentar o disminuir el valor de la variable en una unidad.
Para aumentar se utiliza el operador ++ y para disminuir, el operador --.
Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 14;
            ++valor1;
            document.write(valor1);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 4

En este caso (imagen 3-4) estamos asignando a la variable llamada valor1, el valor 14.
Luego, mediante el operador de incremento, le estamos aumentando en una unidad el valor asignado, por lo que, al mostrar la variable por pantalla, veremos el número 15.

De igual manera podemos utilizar el operador --.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 14;
            --valor1;
            document.write(valor1);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 5

En este caso, el resultado visualizado por pantalla, será el número 13.

En ambos casos hemos colocado los operadores a la izquierda de la variable, es decir como prefijos.
Si probamos ponerlos a la derecha dela variable, veremos que el resultado es el mismo, pero en realidad el comportamiento es diferente.

- Si el operador está colocado a la izquierda de la variable, el valor de dicha variable aumentara o disminuirá ANTES de cualquier otra operación.
- Si el operador está colocado a la derecha, el valor de la variable aumentará o disminuirá DESPUÉS de que la variable sea utilizada.

Para clarificar lo dicho con un ejemplo, supongamos que tenemos el siguiente código:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 3;
            var valor2 = 7;
            var suma = ++valor1 + valor2;

            document.write(suma);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 6

Tenemos dos variables numéricas y una tercera variable llamada suma donde guardamos la suma de las otras dos.

Al momento de sumar ambas variables, vemos que la variable valor1 tiene un operador de incremento a su izquierda, por lo que ANTES de participar en la suma, la variable aumentara su valor en una unidad.

El resultado de la suma debería ser entonces el número 11.

Si probamos este código desde el navegador web veremos que es así.

Hagamos ahora una pequeña modificación, colocando el operador a la derecha de la variable valor1, de esta manera:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 3;
            var valor2 = 7;
            var suma = valor1++ + valor2;

            document.write(suma);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 7

Al guardar las modificaciones y actualizar el navegador, vemos ahora que el resultado ha cambiado, ahora, la variable valor1 aumenta su valor en una unidad DESPUES de haber participado en la suma, por lo tanto, el resultado que vemos por pantalla es el número 10.

Operadores lógicos.

El uso de operadores lógicos está relacionado con los desarrollos complejos, ya que se utilizan para tomas decisiones sobre las instrucciones que debería ejecutar el programa en función de determinadas condiciones.

El resultado de cualquier operación que utilice operadores lógicos, es siempre un valor lógico o booleano.

Negación.

El operador lógico NEGACION se utiliza para obtener el valor contrario al que tiene una variable. La negación se hace colocando como prefijo al identificador de una variable el símbolo ! , veamos qué resultados obtenemos de la negación de las variables booleanas:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var booleano_falso = false;
            var booleano_verdadero = true;

            document.write('Negar booleano falso ='+!booleano_falso+'<br/>');
            document.write('Negar booleano verdadero = '+!booleano_verdadero);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 8

Las variables que no son booleanas, también pueden negarse pero teniendo en cuenta ciertas definiciones.
Las variables numéricas se consideran TRUE cuando su valor es distinto de cero.
Las variables con cadenas de texto se consideran TRUE cuando su valor no es una cadena vacía.

Podemos verificar que esto es cierto negándolas y viendo que valores nos devuelve el intérprete:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var numerico_verdadero = 5;
            var texto_verdadero = 'Hola';

            document.write('Negar numerico verdadero ='+!numerico_verdadero+'<br/>');
            document.write('Negar texto verdadero = '+!texto_verdadero);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 9

Por lo visto anteriormente, ambas variables son consideradas TRUE, por lo tanto, al negarlas, veremos por pantalla el valor FALSE para ambas.

And.

La operación lógica AND obtiene su resultado de la combinación de dos valores booleanos, y dicho resultado será TRUE únicamente cuando los dos valores booleanos combinados, también sean TRUE.

El operador de representa así: &&.

Cualquier otra combinación de valores que no sea TRUE && TRUE dará como resultado FALSE.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 'true';
            var valor2 = 'false';

            document.write('Booleanos combinados ='+valor1 && valor2);
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 10

Según lo visto hasta ahora, al no ser ambos valores TRUE, el resultado que veremos por pantalla será FALSE.
Se recomienda probar diferentes combinaciones y ver qué resultados arroja el operador AND.

Or.

La operación lógica OR también combina dos valores booleanos.
El operador se indica mediante el símbolo || y su resultado será TRUE únicamente si alguno de los dos valores booleanos es TRUE (o los dos).

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 'false';
            var valor2 = 'false';

            document.write('Booleanos combinados ='+valor1 || valor2);
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 11

En el ejemplo de la imagen 3-11 veremos por pantalla el valor FALSE porque ninguno de los valores booleanos combinados era TRUE.

Se recomienda utilizar este código y probar las diferentes combinaciones y el resultado de la operación OR en cada una de ellas.

Operadores matemáticos.

En JavaScript tenemos definidos los operadores matemáticos que nos permitirán manipular u operar sobre las variables del tipo numérico.

Estas operaciones son:

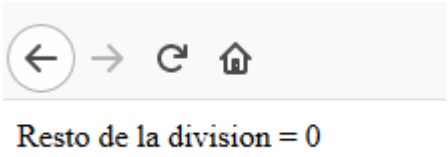
- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Módulo (%)

Las cuatro primeras son las más conocidas y no requieren explicación alguna, pero la última sí.

La operación módulo nos da el resto de una división, por ejemplo, si dividimos 9 sobre 3, es una división exacta y no tiene resto, por lo que si usamos el operador % obtendremos el valor cero.
Veamos esto en código:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 9;
            var valor2 = 3;

            document.write('Resto de la division = '+valor1%valor2);
        </script>
    </head>

    <body>

    </body>
</html>
```

Si lo ejecutamos desde nuestro navegador, veremos el siguiente mensaje:



Resto de la division = 0

Imagen 3 - 12

Probemos ahora con una división que tenga resto, por ejemplo, dividir 12 sobre 5:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 12;
            var valor2 = 5;

            document.write('Resto de la division = '+valor1%valor2);
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 13

El resultado que veremos por pantalla será:

Resto de la division = 2

Una utilidad muy conocida de este operador es la de averiguar si un número es múltiplo de otro (de ser así, el resto de la división entre ambos será cero), por ejemplo si un número es múltiplo de 2, entonces podemos decir que es un número par.

El resto de los operadores matemáticos son muy sencillos de usar.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor1 = 8;
            var valor2 = 3;
            var suma = valor1 + valor2;
            var resta = valor1 - valor2;
            var multiplicacion = valor1 * valor2;
            var division = valor1 / valor2;

            document.write('Suma = '+suma+'<br/>');
            document.write('Resta = '+resta+'<br/>');
            document.write('Multiplicacion = '+multiplicacion+'<br/>');
            document.write('Division = '+division+'<br/>');
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 14

Estos resultados serán los que se vean por pantalla:

Suma = 11
Resta = 5
Producto = 24
Division = 2.6666666666666665

LOS OPERADORES MATEMATICOS SE PUEDEN COMBINAR CON EL OPERADOR DE ASIGNACIÓN PARA OBTENER CÓDIGO ABREVIADO.

Veamos algunos ejemplos:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var numero = 7;
            numero +=2; // Suma un 2 a la variable numero.
            numero -=4; // Resta un 4 a la variable numero.
            numero *=3; // Multiplica por 3 a la variable numero.
            numero /=7; // Divide a la variable numero sobre 7.
        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 15

Escribir numero +=2; es lo mismo que escribir numero = numero + 2;
Escribir numero -=4; es lo mismo que escribir numero = numero – 4;
Escribir numero *=3; es lo mismo que escribir numero = numero * 3;
Escribir numero /=7; es lo mismo que escribir numero = numero / 7;

Operadores relacionales.

Los operadores relacionales son los que establecen una relación entre las variables sobre las que operan y siempre dan como resultado un valor booleano.
Los operadores relacionales con los que podemos trabajar en JavaScript son:

- Mayor que: >
- Menor que: <
- Mayor o igual: >=
- Menor o igual: <=
- Igual que: ==
- Distinto de: !=

Veamos cómo utilizar la comparación de dos variables:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var numero1 = 15;
            var numero2 = 12;
            resultado = numero1 > numero2;

            document.write('La relacion es: '+resultado);

        </script>
    </head>

    <body>

    </body>
</html>
```

Imagen 3 - 16

En este caso, tenemos dos variables numéricas y lo que hacemos es verificar si la variable numero1 es mayor que la variable numro2, el resultado es un booleano (en este caso TRUE) que lo almacenamos en la variable resultado y lo mostramos por pantalla.

Se sugiere hacer la misma operación utilizando el resto de los operadores relacionales.

Estos operadores también se pueden utilizar con cadenas de texto, y para el caso particular que necesitemos saber si una cadena es mayor o menor a otra, lo que se hará es comparar letra a letra hasta encontrar una diferencia.

Para determinar si una letra es mayor o menor que otra, se sigue el siguiente razonamiento de considerar a una letra mayúscula menor que una minúscula y en ambos casos, las primeras letras del alfabeto serán menores que las últimas.

```
<script type="text/javascript">
    var texto1 = 'hola';
    var texto2 = 'Hola';
    resultado = texto1 == texto2;

    document.write('La relacion es: '+resultado);

</script>
```

Imagen 3 - 17

En la imagen 3-17 comparamos dos cadenas de texto cuya única diferencia es la última letra (a mayúscula y a minúscula).

Cómo son dos cadenas diferentes, la comparación por el igual (==) dará como resultado el valor FALSE.

Capítulo 4. Estructuras de control (decisión y repetición).

Que son las estructuras de control de flujo.

Los códigos que se pueden escribir usando solo variables y operadores, son una sucesión de instrucciones básicas, que cumplen una instrucción a continuación de otra sin nada de inteligencia o siquiera una toma básica de decisión

Hay programas un poco más complejos como por ejemplo recorrer un array o establecer una condición ante determinado evento, que no pueden ser realizadas simplemente con una sucesión de instrucciones básicas, es por ello que necesitamos instrucciones de control de flujo que nos permite elegir líneas para ejecutar dentro de nuestro código o repetir una serie de líneas un número de veces según una condición.

Son instrucciones del tipo:

"si se cumple esta condición, haz esto; si no se cumple, haz esto otro"

"repite esto mientras se cumpla esta condición".

Utilizar este tipo de estructuras de control de flujo, convierte a los programas en “inteligentes” permitiendo tomar decisiones en función del valor de las variables utilizadas, de los eventos del programa o de las decisiones que tome un usuario.

Una estructura, en programación, es un bloque de código trabajando en conjunto para un único objetivo común.

Estructura if.

Es quizá la estructura de código más utilizada, no solo en JavaScript, sino también en la mayoría de los lenguajes de programación.
Se utiliza para tomar decisiones en base a una condición, es decir, si la condición se cumple, ejecutamos determinadas acciones.
Esta es la sintaxis:

```
If (condición) {
```

```
        Código a ejecutar
        En caso de ser verdadera
        La condición.
}
```

Para que el código que pongamos entre los corchetes se ejecute, la condición debe ser VERDADERA.

Por ejemplo, tenemos una variable numérica y queremos compararla con el número 15.
Si la variable es mayor o igual a 15, mostraremos por pantalla el mensaje 'La variable es mayor', caso contrario, no pasará nada, es decir, no se ejecutará el código que muestra el mensaje.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor = 17;
            if(valor >= 15){
                document.write('La variable es mayor.');
            }
        </script>
            </head>

    <body>

    </body>
</html>
```

Imagen 4 - 1

En el ejemplo de la imagen 4-1 nuestra variable es mayor a 15, por lo que veremos por pantalla, el mensaje 'La variable es mayor'.

¿Qué pasaría si cambiamos el valor de la variable por uno menor a 15?

Al guardar el código y actualizar el navegador, no veríamos mensaje alguno porque no le hemos aclarado a nuestra estructura que hacer en dicho caso.

Un error muy común al iniciarnos con el uso de operadores, es confundir el operador de asignación = con el operador de comparación ==.

La condición que coloque en el IF puede ser tan compleja como lo necesitemos, y para ello podemos combinar los diferentes operadores vistos hasta ahora.

Veamos el caso en que queremos saber si el valor de una variable está dentro de determinado rango, por ejemplo, si la variable es mayor a 5 y además menor o igual a 10. Para ello, utilizamos una estructura IF con dos condiciones combinadas:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor = 6;

            if(valor >5 && valor <=10){
                document.write('La variable esta dentro del rango.');
            }
        </script>

            </head>

    <body>

    </body>
</html>
```

Imagen 4 - 2

El valor de nuestra variable llamada valor, está dentro del rango que establecimos en la condición, ya que mayor a 5 y menor o igual a 10, por lo que veremos por pantalla el mensaje programado entre corchetes.

Si probamos poner el valor 23 a nuestra variable, y refrescamos la página, no veremos ningún mensaje ya que el valor no está dentro del rango y al igual que en el ejemplo anterior, no le hemos aclarado a nuestra estructura, que hacer en ese caso.

Estructura if else.

En nuestros ejemplos anteriores vimos cómo utilizar la estructura IF para ejecutar determinadas acciones o líneas de código, si se cumplía una determinada condición, pero no le dijimos que hacer si dicha condición no se cumplía.

En determinadas ocasiones, necesitamos ejecutar un bloque de código si la condición es verdadera, pero en caso de ser falsa, necesitaremos ejecutar otro bloque de código.

Para ello contamos con la instrucción else que agregamos a nuestra estructura IF.

Veamos el caso de la variable que analizamos para saber si estaba dentro de un determinado rango, en el código de la imagen 4-2. En dicho ejemplo, si la condición era falsa, no hacíamos nada, pero ahora queremos avisarle al usuario tanto si la condición es verdadera como si es falsa.

Este es el código:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var valor = 23;

            if(valor >5 && valor <=10){
                document.write('La variable esta dentro del rango.');
            }
            else {document.write('La variable esta fuera del rango.');}
        </script>

            </head>

    <body>

    </body>
</html>
```

Imagen 4 - 3

En el ejemplo 4-3, nuestra variable está fuera del rango que declaramos en la condición del IF, y ahora si estamos ejecutando una acción para el caso en que dicha condición resulte falsa.

Resumiendo, con IF ejecutamos determinadas acciones si la condición es VERDADERA, y con el ELSE ejecutamos otras acciones si dicha condición es FALSA.

Este tipo de estructuras se utilizan con todos los tipos de variables y operadores, por ejemplo, veamos una estructura que analiza una variable en la que el usuario debería haber cargado su nombre.

Con una estructura IF ELSE podemos analizarla y decidir qué debe hacer el programa en base al resultado.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var nombre = 'pepe';

            if(nombre != ''){
                document.write('Nombre ingresado, el programa continua.');
            }
            else {document.write('Debe ingresar un nombre.');}
        </script>

            </head>

    <body>

    </body>
</html>
```

Imagen 4 - 4

En el ejemplo de la imagen 4-4 la variable tiene un valor cargado, por lo que la condición del IF resulta verdadera, pero si no le hubiésemos asignado un valor, el resultado de dicha condición sería falso.

Estructura for.

Una estructura for es una estructura de repetición, es una forma eficiente de repetir las mismas acciones durante un número determinado de veces o mientras se cumpla determinada condición.
Este tipo de acciones no se pueden ejecutar con una estructura IF ya que dichas estructuras comprueban la veracidad o falsedad de la condición solo una vez.

La sintaxis de la estructura FOR es la siguiente:

for (inicialización; condición; actualización) {

… código a ejecutar

}

Estas estructuras llevan lo que denominamos 'variables de control', veamos que hace cada elemento de la estructura.

- Inicialización: es el punto de partida dado por la variable de control.
- Condición: es el criterio que decide si se sigue o no con la repetición.
- Actualización: es el nuevo valor que recibe la variable de control.

Vamos a ver un ejemplo sencillo en el que mostraremos cinco mensajes por pantalla:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            for(inicio = 1; inicio <= 5; inicio++){
                document.write('Mensaje numero : '+inicio+'<br/>');
            }

        </script>

            </head>

    <body>

    </body>
</html>
```

Imagen 4 - 5

Analicemos el código.
Dentro del FOR declaramos una variable llamada inicio y le asignamos el valor 1.

A continuación, ponemos la condición de que el bucle se siga repitiendo siempre y cuando el valor de la variable inicio sea menor o igual a 5.
Luego asignamos el nuevo valor a la variable inicio, dicha asignación, y la verificación de la condición, se realizarán luego de la primer ejecución del código que encerramos entre corchetes.
A partir de la primer ejecución, nuestra variable inicio se irá incrementando en una unidad por cada ejecución y cuando su valor sea mayor a 5, la condición dará por terminada la ejecución del bucle.
Si lo probamos, esto es lo que veremos por pantalla:

Mensaje numero : 1
Mensaje numero : 2
Mensaje numero : 3
Mensaje numero : 4
Mensaje numero : 5

El bucle se seguirá ejecutando siempre que se siga comprobando la condición, es muy importante prestar atención a los posibles valores que pueda llegar a tomar la variable de control, por ejemplo, ¿Qué sucedería si ejecutáramos el siguiente código?

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            for(inicio = 1; inicio  >0; inicio++){
                document.write('Mensaje numero : '+inicio+'<br/>');
            }

        </script>

            </head>

    <body>

    </body>
</html>
```

Imagen 4 - 6

En este caso la condición va a ser siempre cierta y nuestra aplicación va a entrar en un bucle eterno que nos obligará a cerrar el navegador y reiniciarla.

Podemos también usar una estructura de repetición FOR para mostrar los valores guardados en un array (recordemos que el primer elemento de un array en el cero):

```
<script type="text/javascript">
    var alumnos = ['Pedro','Ricardo','Laura'];

    for(inicio = 0; inicio <=2; inicio++){
        document.write('Alumno numero : '+inicio+' '+alumnos[inicio]+'<br/>');
    }

</script>
```

Imagen 4 - 7

Estructura While.

Es también una estructura de repetición, de construcción más simple, que ejecutará el código que definamos entre corchetes siempre y cuando la condición sea verdadera.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var i=1;
            while (i<=3){
                document.write('Ejecucion '+i+'<br/>');
                i++;
            }
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 4 - 8

Esta estructura primero evalúa la condición y luego ejecuta el código.
Esto es lo que vemos por pantalla si lo ejecutamos:

Ejecucion 1
Ejecucion 2
Ejecucion 3

¿Qué pasaría si la condición fuese falsa desde el inicio?
Modificamos el código para que se de esta situación y probamos:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var i=6;
            while (i<=3){
                document.write('Ejecucion '+i+'<br/>');
                i++;
            }
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 4 - 9

La estructura verifica que la condición es falsa y no ejecuta el código ni una sola vez.

Estructura Do While.

La estructura Do While es similar a la estructura While, con la diferencia que antes de evaluar la condición, siempre ejecuta el código.

Es decir, si necesitamos que nuestro código se ejecute al menos una vez independientemente del valor de la condición, esta es la estructura que necesitamos.

Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var i=6;

            do {
                document.write('Valor de i :'+i+' '+'<br/>');
                i++;
            }
            while(i<=3)
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 4 - 10

Si ejecutamos el código de la imagen 4-10, veremos que la estructura primero ejecuta el código entre los corchetes y recién después, evalúa la condición para decidir si sigue ejecutando el código o sale del bucle.

Esto se ve por pantalla:

Valor de i :6

Capítulo 5. Funciones de JavaScript.

Como casi todos los lenguajes de programación, JavaScript incorpora una serie de herramientas para el manejo de variables, llamadas funciones.

De esta forma, muchas de los complejos códigos que necesitamos programar para obtener determinados resultados, ya vienen resueltos en JavaScript, son las funciones que incorpora el lenguaje.

Funciones para el manejo de texto.

Veamos algunas de las principales funciones que incorpora JavaScript para el manejo de texto.

FUNCIÓN LENGTH: calcula la longitud de una cadena de texto, es decir, cuenta la cantidad de caracteres que la conforman.
Para utilizarla, debemos colocarla a continuación de la variable de texto a la cual le queremos calcular la longitud, separada por un punto.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var texto = 'Hola mundo';
            document.write('Longitud del texto = '+texto.length);
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 1

Este es el resultado que se muestra por pantalla:

Longitud del texto = 10

Como vemos, la función LENGTH cuenta todos los caracteres, incluidos los espacios.
Podemos utilizarla dentro de una estructura IF ELSE por ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var texto = 'Hola mundo';
            if(texto.length <= 8){
            document.write('La longitud del texto es aceptable')}
            else {document.write('La longitud del texto es excesiva')}
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 2

En la condición del IF estamos especificando que si la longitud del texto es menor o igual a 8 caracteres, muestre el mensaje de que dicha longitud es aceptable.

Caso contrario, es decir si la cantidad de caracteres es mayor a 8, el programa la considera excesiva y ejecuta las acciones correspondientes, en este caso simple, muestra un mensaje por pantalla.

Así se ve:

La longitud del texto es excesiva

OPERADOR +: además de utilizarlo como operador matemático, también lo podemos utilizar para concatenar variables de texto.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var variable1 = 'Bienvenido';
            var variable2 = 'usuario.';

            document.write(variable1+' '+variable2);
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 3

Esto es lo que vemos por pantalla:

Bienvenido usuario.

FUNCION CONCAT ( ): se utiliza, al igual que el operador +, para concatenar variables de texto.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var variable1 = 'Bienvenido';
            var variable2 = ' usuario.';
            var mensaje = variable1.concat(variable2);

            document.write(mensaje);
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 4

Esto veríamos por pantalla:

Bienvenido usuario.

Podemos concatenar la cantidad de variables que sean necesarias, por ejemplo, agreguemos otra cadena de texto al código anterior:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var variable1 = 'Bienvenido';
            var variable2 = ' usuario.';
            var variable3 = ' Disfrute de la web.';
            var mensaje = variable1.concat(variable2).concat(variable3);

            document.write(mensaje);
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 5

Este es el mensaje que se muestra en pantalla:

Bienvenido usuario. Disfrute de la web.

Las cadenas de texto también pueden unirse a variables numéricas, veamos un ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var documento_tipo = 'DNI ';
            var documento_numero = 12345678;
            var documento = documento_tipo.concat(documento_numero);
            document.write('Documento '+documento);
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 6

Esto se ve por pantalla:

Documento DNI 12345678

Es común olvidarse de colocar un espacio en las variables de texto, son muy útiles para que el mensaje final sea entendible. Dicho espacio puede dejarse al comienzo o al final, como en el ejemplo de la imagen 5-6 donde dejamos un espacio en la variable documento_tipo a continuación de la palabra DNI.

En caso de no poder incorporar espacios dentro de las variables, también se pueden colocar al momento de concatenarlas, como venimos haciendo desde ejemplos anteriores.

FUNCION toUpperCase( ): convierte todos los caracteres de la cadena de texto en mayúsculas.

Se utiliza de la misma manera que las funciones anteriores: variable_de_texto.funcion

Ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje = 'hola mundo';
            document.write(mensaje.toUpperCase());
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 7

El texto contenido en la variable llamada mensaje, está completamente en letras minúsculas, pero al aplicarle la función toUpperCase las convertimos a mayúsculas.

Asi se ve por pantalla:

HOLA MUNDO

No solamente podemos aplicar esta función para modificar una salida por pantalla, también podemos asignar esa transformación de minúsculas a mayúsculas al contenido de una misma variable o pasar el contenido a otra.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje1 = 'hola mundo';
            var mensaje2 = mensaje1.toUpperCase();
            mensaje1 = mensaje1.toUpperCase();

            document.write('Contenido de mensaje1: '+mensaje1+'<br/>');
            document.write('Contenido de mensaje2: '+mensaje2);
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 8

En este caso, el código de la imagen 5-8, hacemos dos cosas.
Primero, asignamos a la variable mensaje2, el contenido de la variable mensaje1 pero pasado a mayúsculas, y a continuación asignamos a la variable mensaje1 su propio contenido, también pasado a mayúsculas.
Esta es la salida por pantalla:

Contenido de mensaje1: HOLA MUNDO
Contenido de mensaje2: HOLA MUNDO

FUNCION toLowerCase( ): convierte todos los caracteres de una cadena de texto en minúsculas.

Su uso es idéntico al de la función vista anteriormente, to UpperCase, por lo que solo veremos un ejemplo de uso ya que los ejemplos vistos anteriormente aplican a esta función.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje1 = 'HOLA MUNDO';
            var mensaje2 = mensaje1.toLowerCase();
            mensaje1 = mensaje1.toLowerCase();

            document.write('Contenido de mensaje1: '+mensaje1+'<br/>');
            document.write('Contenido de mensaje2: '+mensaje2);
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 9

Es exactamente el mismo caso que el de la imagen 5-8 pero con la salvedad que aquí pasamos de mayúsculas a minúsculas.

Esto se ve por pantalla:

Contenido de mensaje1: hola mundo
Contenido de mensaje2: hola mundo

FUNCION charAt( ): obtiene el carácter que se encuentra en una determinada posición, dentro de una cadena de texto. Dicha posición se deberá indicar entre los paréntesis de la función charAt y debemos tener en cuenta que, al igual que sucede con el array, la primera posición de la cadena, es el cero.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var mensaje1 = 'Esta es una cadena de texto.';

            document.write('La letra en la posicon 10 es: '+mensaje1.charAt(10));
        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 10

Y esto es lo que vemos por pantalla:

La letra en la posicon 10 es: a

FUNCION indexOf( ): calcula la posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si el carácter se incluye varias veces dentro de la cadena de texto, se devuelve su primera posición empezando a buscar desde la izquierda. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
<script type="text/javascript">
    var mensaje1 = 'Hola Mundo';

    document.write('Posici\u00F3n letra M: '+mensaje1.indexOf('M')+'<br/>');
    document.write('Posici\u00F3n letra J: '+mensaje1.indexOf('J'));
</script>
```

Imagen 5 - 11

Sobre la frase 'Hola Mundo', efectuamos la búsqueda de dos letras, la M y la J.

Vemos que la letra M se encuentra en la posición 5 (recordemos que se empieza a contar desde la posición cero) pero la letra J no se encuentra dentro del texto, por lo que la función debería devolver el valor -1.
Vemos la ejecución del código y como es la salida por pantalla:

Posición letra M: 5
Posición letra J: -1

Un detalle agregado en el código dentro de document.write, en el mensaje a mostrar, utilizamos el carácter de escape \u junto al código de la letra o minúscula acentuada.
Esto es para poder mostrar los acentos por pantalla.

FUNCION lastIndexOf( ): calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
<script type="text/javascript">
    var mensaje1 = 'Manual de Java';

    document.write('Posici\u00F3n letra a: '+mensaje1.lastIndexOf('a')+'<br/>');
    document.write('Posici\u00F3n letra x: '+mensaje1.lastIndexOf('x'));
</script>
```

Imagen 5 - 12

Es en ejemplo de la imagen 5-12, buscamos la última posición de la letra 'a' (recordemos nuevamente que siempre se empieza a contar desde la posición cero) y la última posición de la letra 'x', que al no ser parte del texto, la función nos debería devolver el valor -1.

Salida por pantalla:

Posición letra a: 13
Posición letra x: -1

Esta función comienza su búsqueda desde el final de la cadena hacia adelante, aunque la posición devuelta es la correcta contando de izquierda a derecha.

FUNCION substring (inicio, final): extrae una porción de una cadena de texto.

El segundo parámetro es opcional. Si sólo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final.

Si solo se indica el primer parámetro, y es un número negativo, la función devuelve el texto completo.

Si se indican ambos parámetros, la función toma como parámetro de inicio al más pequeño de los dos, y solo devuelve los caracteres comprendidos entre las posiciones de inicio y la inmediatamente anterior a la posición final.

Ejemplos:

```
<script type="text/javascript">
    var mensaje = 'Manual de Java';

    document.write(mensaje.substring(4)+'<br/>'); //ej.1
    document.write(mensaje.substring(-3)+'<br/>'); //ej.2
    document.write(mensaje.substring(0,2)+'<br/>'); //ej.3
    document.write(mensaje.substring(2,0)); //ej.4

</script>
```

Imagen 5 - 13

Veamos cada ejemplo del código.
A la variable mensaje le asignamos el texto 'Manual de Java'.
Luego hacemos cuatro extracciones de texto usando los conceptos teóricos explicados para esta función.

En la primer línea de código a continuación de la asignación de la variable, lo que hacemos es extraer de la cadena de texto, todo el mensaje a partir de la posición 4, por lo que en pantalla, para el ej.1 deberíamos ver el texto 'al de Java'.

En la línea siguiente, colocamos como único parámetro, un número negativo, por lo que la función debería devolver el mensaje completo.

En la tercer línea de código, colocamos los dos parámetros, un cero para el inicio y un dos para el final, es decir, le pedimos que extraiga del texto los caracteres que se encuentren entre la posición 0 y la posición 1 (recordemos que al colocar los dos parámetros, la función extrae el texto comprendido entre la posición de inicio y la inmediatamente anterior a la posición final). El resultado devuelto por la función debería ser el texto 'Ma'.

Por último, utilizamos la función con, también con dos parámetros pero los invertimos, damos al inicio un valor mayor al del final. En este caso, según vimos en la descripción teórica de la función, esta deberá tomar como valor de inicio al as pequeño de los dos parámetros y como valor final al mayor. Como utilizamos los mismos valores que en el ejemplo anterior pero invertidos, la función debería devolvernos exactamente la misma porción del texto, 'Ma'.

Vemos la salida por pantalla:

al de Java
Manual de Java
Ma
Ma

FUNCION split (separador): convierte una cadena de texto en un array de cadenas de texto.
La función parte la cadena de texto, determinando sus trozos a partir del carácter separador indicado.
Si como separador utilizamos comillas con un espacio (" ") los elementos del array serán las palabras que forman la cadena de texto.

Si en cambio el separador son las comillas sin el espacio de separación (“”), los elementos del array serán las letras que forman la cadena de texto.

```
<script type="text/javascript">
    var mensaje = 'Manual muy completo de Java';
    var array_palabras = mensaje.split(" ");
    var array_letras = mensaje.split('');

    document.write('Array palabras: '+array_palabras+'<br/>');
    document.write('Array letras: '+array_letras);

</script>
```

Imagen 5 - 14

Al ser ahora un array, podemos acceder a sus elementos a través del índice tal y como lo vimos en capítulos anteriores.

Po ejemplo, accedemos al elemento 2 del array llamado array_palabras que debería ser la palabra ‘completo’;

```
<!DOCTYPE html>
<html>
    <head>

        <script type="text/javascript">
            var mensaje = 'Manual muy completo de Java';
            var array_palabras = mensaje.split(' ');
            document.write('elemento 2 del Array: '+array_palabras[2]);
        </script>

    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 15

Comprobamos la salida por pantalla del resultado esperado:

elemento 2 del Array: completo

Funciones útiles para el manejo de Arrays.

FUNCION length ( ): calcula el número de elementos que contiene un array.
Se utiliza de manera similar a como la utilizamos con cadenas de texto.

```
<script type="text/javascript">
    var impares = [1,3,5,7,9];

    document.write('Cantidad de impares: '+impares.length);
</script>
```

Imagen 5 - 16

Salida por pantalla:

Cantidad de impares: 5

FUNCION concat ( ): se utiliza para concatenar los elementos de varios Arrays.

Si al concatenar dos array que tienen elementos en común (es decir que un mismo elemento figura en ambos array), el array concatenado contendrá a ambos, es decir, tendrá duplicados.

```
<script type="text/javascript">
    var lista1 = [1,2,3,1];
    var lista2 = [4,5];

    var lista_final = lista1.concat(lista2);
    document.write(lista_final);

</script>
```

Imagen 5 - 17

En este ejemplo, tenemos dos array que no tienen elementos en común, pero el array lista1, tiene un elemento repetido. Al concatenar ambos vectores, el vector resultante tiene también los elementos repetidos.

Esto se ve por pantalla:

1,2,3,1,4,5

¿Qué pasaría si ahora agregamos a la concatenación un array con un elemento que ya figura en otro de los vectores? Pues se agregaría al vector concatenado, veamos el ejemplo:

```
<script type="text/javascript">
    var lista1 = [1,2,3,1];
    var lista2 = [4,5];
    var lista3 = [5,6];

    var lista_final = lista1.concat(lista2).concat(lista3);
    document.write(lista_final);

</script>
```

Imagen 5 - 18

En este ejemplo, agregamos un vector con dos elementos, el número 5 que ya figuraba en el vector lista2 y un nuevo elemento, el número 6.

Esto es lo que se ve por pantalla al concatenar los tres Arrays:

1,2,3,1,4,5,5,6

Por último, también podemos concatenarle a un array, una lista de elementos sin necesidad de que estén incluidos en otro array.

```
<script type="text/javascript">
    var lista1 = [1,2,3];

    var lista_final = lista1.concat('a','b','c');
    document.write(lista_final);

</script>
```

Imagen 5 - 19

Además vemos en este ejemplo que podemos concatenar diferentes tipos de datos, algo que ya habíamos visto en capítulo de Arrays.

Este es el resultado visto por pantalla:

1,2,3,a,b,c

FUNCION join (separador): es la función contraria a split ( ).
Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador indicado.

```
<script type="text/javascript">
    var lista1 = ['Bienvenido','estimado','usuario'];

    document.write('Vector: '+lista1+'<br/>');
    document.write('Texto con separaci\u00f3n: '+lista1.join(" ")+'<br/>');
    document.write('Texto sin separaci\u00f3n: '+lista1.join(""));

</script>
```

Imagen 5 - 20

Partimos del array llamado lista1.

Tenemos tres salidas por pantalla.
En la primera, lo mostramos como un vector.
En la segunda lo unimos en un texto con separación de sus elementos.
En la tercera, lo unimos sin separar los elementos.

Esto es lo que se ve por pantalla:

Vector: Bienvenido,estimado,usuario
Texto con separación: Bienvenido estimado usuario
Texto sin separación: Bienvenidoestimadousuario

FUNCION pop ( ): elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye en 1 elemento.

```
<script type="text/javascript">
    var lista1 = ['Bienvenido','estimado','usuario'];
    var ultimo_elemento = lista1.pop();

    document.write('Ultimo elemento: '+ultimo_elemento+'<br/>');
    document.write('Array actualizado: '+lista1);

</script>
```

Imagen 5 - 21

En la salida por pantalla vemos como la función pop ( ) elimina del array original el último elemento pero se lo asignamos a una variable llamada ultimo_elemento.

Ultimo elemento: usuario
Array actualizado: Bienvenido,estimado

FUNCION push ( ): añade un elemento al final del array.
El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez).

Veamos un ejemplo de cómo agregar un elemento a un array:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            var lista1 = [1,2,3,4,5];
            document.write('Array original: '+lista1+'<br/>');

            lista1.push(6);
            document.write('Array actualizado: '+lista1);

        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 22

Vemos por pantalla el array original y el array con el nuevo elemento:

Array original: 1,2,3,4,5
Array actualizado: 1,2,3,4,5,6

No solo podemos agregar de a un elemento por vez, podemos agregar de una sola vez una lista de varios elementos, por ejemplo:

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            var lista1 = [1,2,3,4,5];
            document.write('Array original: '+lista1+'<br/>');

            lista1.push(6,7,8,9);
            document.write('Array actualizado: '+lista1);

        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 23

En este ejemplo, el de la imagen 5-23, agregamos cuatro elementos nuevos al array original, esto se ve por pantalla:

Array original: 1,2,3,4,5
Array actualizado: 1,2,3,4,5,6,7,8,9

FUNCION shift ( ): elimina el primer elemento del array y lo devuelve.
El array original se ve modificado y su longitud disminuida en 1 elemento.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            var lista1 = ['Bienvenido','estimado','usuario'];
            var ultimo_elemento = lista1.shift();

            document.write('Ultimo elemento: '+ultimo_elemento+'<br/>');
            document.write('Array actualizado: '+lista1);

        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 24

Vemos el resultado por pantalla, del primer elemento que sacamos del arraz y de cómo queda dicho array sin ese elemento.

Ultimo elemento: Bienvenido
Array actualizado: estimado,usuario

FUNCION unshift ( ): añade un elemento al principio del array.
El array original se modifica y aumenta su longitud en 1 elemento. (También es posible añadir más de un elemento a la vez).

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            var lista1 = [1,2,3,4,5];
            document.write('Array original: '+lista1+'<br/>');

            lista1.unshift(0);
            document.write('Array actualizado: '+lista1);

        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 25

También es posible insertar más de un elemento al principio del array.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            var lista1 = [1,2,3,4,5];
            document.write('Array original: '+lista1+'<br/>');

            lista1.unshift(10,20,30);
            document.write('Array actualizado: '+lista1);

        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 26

Ese es el resultado que vemos por pantalla:

Array original: 1,2,3,4,5
Array actualizado: 10,20,30,1,2,3,4,5

FUNCION reverse ( ): modifica un array colocando sus elementos en el orden inverso a su posición original.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">

            var lista1 = [1,2,3,4,5,6];
            document.write('Array original: '+lista1+'<br/>');

            lista1.reverse();
            document.write('Array inverso: '+lista1);

        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 27

Vemos el resultado por pantalla:

Array original: 1,2,3,4,5,6
Array inverso: 6,5,4,3,2,1

Funciones para manejo de números.

NaN: viene del inglés “Not a number”, JavaScript emplea este tipo de valor para indicar un valor numérico no definido, por ejemplo, la división cero sobre cero.

```
<!DOCTYPE html>
<html>
    <head>
        <script type="text/javascript">
            var numero1 = 0;
            var numero2 = 0;
            document.write('Valor obtenido: '+numero1/numero2);

        </script>
    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 28

Al ejecutar el código veremos por pantalla el mensaje “Valor obtenido: NaN”.

FUNCION IsNaN ( ): permite proteger a la aplicación de posibles valores numéricos no definidos.

```
<script type="text/javascript">
    var numero1 = 0;
    var numero2 = 0;
    if(isNaN(numero1/numero2)) {
     document.write('La division no esta definida para los numeros indicados');
    }
    else {
     document.write('La division es igual a => ' + numero1/numero2);
    }
</script>
```

Imagen 5 - 29

Aprovechando la definición de valor tipo NaN, podemos tratar este tipo de anomalías sin necesidad de que nuestra aplicación se vea desbordada o interrumpida en su funcionamiento.

En el ejemplo de la imagen 5-29, lo que hacemos, es mediante una estructura de decisión detectar el valor NaN y proceder de acuerdo a lo que sea mas conveniente.

Este mensaje se verá por pantalla:

La division no esta definida para los numeros indicados

FUNCION Infinity: hace referencia a un valor numérico infinito y positivo (también existe el valor – Infinity para los infinitos negativos).

```
<!DOCTYPE html>
<html>
    <head>

        <script type="text/javascript">
            var numero1 = 10;
            var numero2 = 0;

            document.write('Resultado = '+numero1/numero2);

        </script>

    </head>
    <body>

    </body>
</html>
```

Imagen 5 - 30

El resultado visto por pantalla es el siguiente:

Resultado = Infinity

FUNCION toFixed (dígitos): devuelve el número original con tantos decimales como los indicados por el parámetro dígitos y realiza los redondeos necesarios. Se trata de una función muy útil por ejemplo para mostrar precios.

```
<script type="text/javascript">
    var numero1 = 3456.6789;

    document.write('Redondear a 2 digitos = '+numero1.toFixed(2)+'<br/>');
    document.write('Redondear a 3 difitos = '+numero1.toFixed(3)+'<br/>');
    document.write('Redondear sin decimales = '+numero1.toFixed()+'<br/>')

</script>
```

Esto se ve por pantalla:

Redondear a 2 digitos = 3456.68
Redondear a 3 difitos = 3456.679
Redondear sin decimales = 3457

Capítulo 6. Funciones de usuario (parámetro y argumento).

Funciones de usuario.

Cuando se desarrolla una aplicación compleja, es muy habitual utilizar una y otra vez las mismas instrucciones. Un script para una tienda de venta por internet, por ejemplo, tiene que calcular el precio total de los productos varias veces, para añadir los impuestos y los gastos de envío.

Cuando una serie de instrucciones se repiten una y otra vez, se complica demasiado el código fuente de la aplicación, ya que el código de la aplicación es mucho más largo porque muchas instrucciones están repetidas.

Si se quiere modificar alguna de las instrucciones repetidas, se deben hacer tantas modificaciones como veces se haya escrito esa instrucción, lo que se convierte en un trabajo muy pesado y muy propenso a cometer errores y hace de nuestra aplicación, muy difícil de mantener.

Las funciones son la solución a todos estos problemas, tanto en JavaScript como en el resto de lenguajes de programación.

Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y que se pueden reutilizar fácilmente.

Supongamos el caso en que debamos sumar dos números un determinado número de veces, podríamos escribir tantas líneas de código que realicen al suma como se necesario o armar una estructura que repita la ejecución de dicha suma un número determinado de veces.

Ambas soluciones son ineficientes, la primera porque engrosaría nuestro código inútilmente y la segunda porque estaría siempre a punto de fallar si alguno de los parámetros necesarios se saliera de rango.

Las funciones en cambio, se escriben una sola vez y se las puede reutilizar tantas veces como sea necesario.

La sintaxis básica de una función es:

```
function nombre_de_funcion ( ) {
        …
}
```

Dentro de los corchetes pondremos el código a ejecutarse cada vez que invoquemos a la función.

El nombre de la función se utiliza para llamar a esa función cuando sea necesario. El concepto es el mismo que con las variables, a las que se les asigna un nombre único para poder utilizarlas dentro del código. Después del nombre de la función, se incluyen dos paréntesis cuyo significado se detalla más adelante. Por último, los símbolos { y } se utilizan para encerrar todas las instrucciones que pertenecen a la función (de forma similar a como se encierran las instrucciones en las estructuras if o for).

Las funciones más sencillas no necesitan ninguna información para producir sus resultados. Sin embargo, la mayoría de funciones de las aplicaciones reales deben acceder al valor de algunas variables para producir sus resultados, es decir, deben recibir algún argumento o valor para procesarlo y devolvernos un resultado.

Las variables que necesitan las funciones se llaman argumentos. Antes de que pueda utilizarlos, la función debe indicar cuántos argumentos necesita y cuál es el nombre de cada argumento. Además, al invocar la función, se deben incluir los valores que se le van a pasar a la función. Los argumentos se indican dentro de los paréntesis que van detrás del nombre de la función y se separan con una coma (,).

Veamos un ejemplo simple de una función que sirva para sumar dos números cualquiera que le pasaremos por argumento.

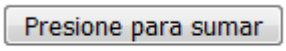
```
<!DOCTYPE html>
<html>
    <head>

        <script type="text/javascript">

            function sumar(num1,num2){
                resultado=num1+num2;
                alert('El resultado de la suma es: '+resultado);
            }

        </script>

    </head>
    <body>
        <button onclick="sumar(3,7)">Presione para sumar</button>
    </body>
</html>
```

Imagen 6 - 1

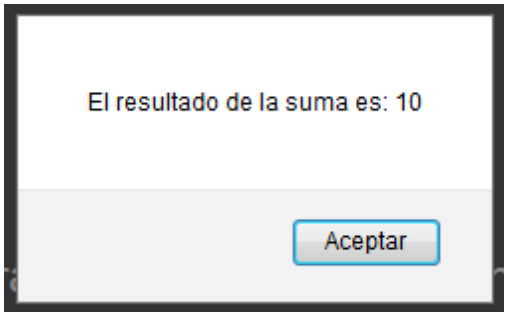
Esta función recibe dos argumentos (los que están dentro del paréntesis) que serán los números a sumar. Dentro de los corchetes, se encuentra el código que se va a ejecutar cada vez que llamemos a la función, en nuestro ejemplo, se sumaran los dos números que le pasemos y se mostrará el resultado por pantalla mediante un mensaje de alerta.
Lo interesante de usar funciones, es que no debemos reescribir el mismo código varias veces, sino que podemos reutilizarlo.

Veamos un ejemplo de cómo llamar a la función, pasándole los números que necesitamos que sume, utilizando etiquetas HTML, en este caso un botón.
En el código de la imagen 6-1 declaramos una función llamada sumar, que recibe dos valores, los cuales va a sumar y guardar en una variable resultado y luego mostrar dicho resultado en un mensaje de alerta.

Al ejecutarlo, vemos un botón en pantalla que deberemos presionar para que la suma se efectué:



Si lo presionamos, aparecerá un mensaje de alerta con el resultado de haber sumado los números 3 y 7:



Si quisiéramos sumar algunos pares de números más, no tendríamos que reescribir nuevamente el código, solo tendríamos que reutilizar la función que acabamos de definir.

Un detalle importante es que para poder utilizar los argumentos dentro de la función, debemos emplear los mismos nombres con que los definimos entre los paréntesis.

Veamos otro ejemplo donde invocamos la función dentro de las etiquetas de <script>.
Las funciones, además de recibir valores, también pueden devolverlos, en el caso de nuestro ejemplo de una función que realiza una suma, lo que haremos es devolver el resultado de dicha suma para poder utilizarlo fuera de la función.
Para ello tenemos la palabra RETURN.

```
<!DOCTYPE html>
<html>
    <head>

        <script type="text/javascript">

            function sumar(num1,num2){
                resultado=num1+num2;
                return resultado;
            }

            var suma = sumar(2,4);
            document.write('El resultado es = '+suma);

        </script>

    </head>
    <body>

    </body>
</html>
```

Imagen 6 - 2

Vemos que al final de la función, agregamos la instrucción RETURN que lo que hace, es devolver o sacar fuera de la función, el valor de la variable resultado.

Fuera de la función, puedo o no utilizar el nombre de dicha variable, en este caso, elegí usar el nombre suma en lugar de usar el nombre resultado, ya que la función no devuelve una variable, devuelve su contenido.

Esto es lo que veremos por pantalla:

El resultado es = 6

Como dijimos, un aspecto interesante de las funciones es poder reutilizar el código sin necesidad de reescribirlo.
Veamos el caso en que necesitemos sumar más varios pares de números:

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

    function sumar(num1,num2){
        resultado=num1+num2;
        return resultado;
    }

    var suma = sumar(2,4);
    document.write('El resultado es = '+suma+'<br/>');

    var suma = sumar(3,6);
    document.write('El resultado es = '+suma+'<br/>');

    var suma = sumar(1,7);
    document.write('El resultado es = '+suma+'<br/>');

</script>
```

Imagen 6 - 3

Para sumar tres pares diferentes de números, solo tuve que llamar o invocar a la misma función una vez por cada suma que necesité hacer.

Este es el resultado visto por pantalla:

El resultado es = 6
El resultado es = 9
El resultado es = 8

Otro ejemplo de uso de funciones, es que podemos tener tantas funciones como sean necesarias y aplicarlas sobre un mismo grupo de variables.

Veamos el ejemplo en que tenemos dos funciones para el cálculo del salario de un empleado.

La primera función se encarga de calcular el sueldo del empleado en base a los días que trabajó, y la segunda función se encarga de aplicar un descuento a dicho sueldo del 10%.

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

    function sueldo(dias){
        var sueldo = 350*dias;
        return sueldo;
    }

    function descuento(sueldo){
        var descuento = (sueldo*10)/100;
        return descuento;
    }

    var sueldo = sueldo(14);
    var descuento = descuento(sueldo);
    var sueldo_final = sueldo - descuento;
    document.write('Sueldo base: '+sueldo+'<br/>');
    document.write('Descuento del 10% :'+descuento+'<br/><br/>');
    document.write('Sueldo final: '+sueldo_final);


</script>
```

Imagen 6 - 4

Este es el resultado visto por pantalla:

Sueldo base: 4900
Descuento del 10% :490

Sueldo final: 4410

El código es simple, la primera función recibe el número de días que el empleado trabajó y lo multiplica por 350 que es el pago diario por día trabajado.
Luego, mediante la palabra RETURN, devuelve el valor calculado.

La segunda función, lo que hace es calcular el valor del descuento del 10% para el sueldo que le ingresemos.

Una vez definidas as funciones, declaramos tres variables, sueldo, descuento y

sueldo_final que las usaremos para mostrar los datos por pantalla.

Algunas consideraciones a tener en cuenta cuando trabajamos con funciones:

- Aunque casi siempre se utilizan variables para pasar los datos a la función, se podría haber utilizado directamente el valor de esas variables.

- El número de argumentos que se pasa a una función debería ser el mismo que el número de argumentos que ha indicado la función. No obstante, JavaScript no muestra ningún error si se pasan más o menos argumentos de los necesarios.

- El orden de los argumentos es fundamental, ya que el primer dato que se indica en la llamada, será el primer valor que espera la función; el segundo valor indicado en la llamada, es el segundo valor que espera la función y así sucesivamente.

- Se puede utilizar un número ilimitado de argumentos, aunque si su número es muy grande, se complica en exceso la llamada a la función.

- No es obligatorio que coincida el nombre de los argumentos que utiliza la función y el nombre de los argumentos que se le pasan.

- Cuando una función devuelve un valor, se debe declarar una variable para guardar dicho valor en el punto en que se realiza la llamada. Si no se indica el nombre de ninguna variable, JavaScript no muestra ningún error y el valor devuelto por la función simplemente se pierde y por tanto, no se utilizará en el resto del programa. En este caso, tampoco es obligatorio que el nombre de la variable devuelta por la función coincida con el nombre de la variable en la que se va a almacenar ese valor.

- Si la función llega a una instrucción de tipo return, se devuelve el valor indicado y finaliza la ejecución de la función. Por tanto, todas las instrucciones que se incluyen después de un return se ignoran y por ese motivo la instrucción return suele ser la última de la mayoría de funciones.

Capítulo 7. DOM.