Aprende a programar

(con JavaScript)

Autor: Manuel Cabello

Versión 1.0

Fecha: Abril 2016

Digital Learning

Este documento recoge los 6 módulos que componían el curso online "**Aprende a Programar con JavaScript**" que escribí para la compañía Digital Learning SL de la que fui fundador y director durante 15 años.

El curso lo escribí en el año 2016, por lo que algunas cuestiones a los que hacen referencia estos contenidos pueden haber cambiado desde entonces. No obstante, al ser un curso más enfocado a explicar conceptos generales de programación que a enseñar un determinado lenguaje (en este caso JavaScript) no creo que afecte a la esencia y utilidad de los mismos.

El curso lo diseñé como una **introducción a fundamentos básicos de la programación** para personas que no tuvieran formación y experiencia en este campo. Traté de hacer especial hincapié en la claridad de la exposición, tratando de utilizar analogías y ejemplos que fueran fáciles de entender por personas sin perfil técnico.

El curso se complementaba con **videos**, **autotests** y **ejercicios** que podían resolverse mediante el uso de un editor online de código.

Desafortunadamente, nuestra web ya no está accesible, al cesar la actividad de la compañía, pero puedes **consultar los videos en la plataforma Vimeo**, en el enlace: https://vimeo.com/showcase/11896716

No he podido incorporar los ejercicios a esta documentación por el formato en que estaban. Quizás pueda hacerlo más adelante en una nueva versión del documento.

En un mundo donde el uso de la Inteligencia Artificial empieza a ser la opción preferida para el auto-aprendizaje -y empezando a competir con los cursos tutorizados-, posiblemente este tipo de recursos se tengan en cuenta cada vez menos, pero si a alguien le resulta de utilidad, estupendo, habrá merecido su publicación.

Manuel Cabello
(Septiembre 2025)

Aprende a programar

(con JavaScript)

Módulo 1. Introducción

Autor: Manuel Cabello

Versión 1.0

Fecha: Abril 2016

Digital Learning

Qué es programar

Si buscamos una descripción informal y en muy pocas palabras sobre **qué es programar**, aparte de la muy obvia: "es crear programas usando un lenguaje de programación", podríamos encontrar otras como: "darle instrucciones al ordenador" o "enseñarle al ordenador a hacer algo". Son definiciones algo esquemáticas, pero si las unimos podemos obtener una mejor descripción: "**es crear programas** (software) **que enseñan al ordenador a hacer algo, a través de una secuencia de instrucciones que debe seguir, y que hemos escrito usando un lenguaje** (de programación) **específico para ello**".

Este es un curso práctico donde no pretendemos ahondar en las bases teóricas de la programación. Aunque seguramente ya tienes una idea, creemos que es mejor que vayas descubriendo en qué consiste programar, realizando precisamente esa actividad. En esta sección de introducción, hablaremos de forma breve, sobre algunos conceptos esenciales, algo así como el 'abc', que nos permitan comenzar a andar.

Algoritmos, programas y lenguajes de programación

Para ayudar a entender la programación a un nivel básico se suele utilizar símiles, como las instrucciones de montaje de un mueble o una receta de cocina. En ellas explicamos cómo realizar algo a través de una serie de pasos detallados. Por ejemplo, al escribir una receta, primero hemos tenido que descomponer mentalmente el proceso de cocinar un plato en una serie de tareas con un orden lógico:

- Limpiar el pescado
- Echarle dos pizcas de sal
- Picar 20 gr. de cebolla
- Calentar 2 cucharas de aceite en una sartén
- Dorar la cebolla
- etc...

Luego escribiremos esos pasos. Podría ser en español, en inglés o cualquier otro idioma, pero las instrucciones seguirían siendo las mismas.

Pues bien, al desglose de un proceso en pasos detallados y ordenados le denominamos

algoritmo y el fichero donde transcribimos estas instrucciones usando un **lenguaje de programación** concreto (Javascript, PHP, Python, Java...) para que pueda ser ejecutado por un ordenador, le llamamos **programa** (*).

La sintaxis de estos lenguajes de programación es bastante más simple que nuestros idiomas y utilizan un vocabulario y un conjunto de reglas mucho más reducido. Eso sí, son muy estrictas y debemos seguirlas a rajatabla para que el ordenador pueda interpretarlas sin que produzca un error.

En resumen, estos **programas** son un **conjunto de sentencias** escritas en un **lenguaje de programación** que **le dicen al ordenador qué tareas debe realizar y en qué orden**, a través de una serie de **instrucciones** que detallan completamente ese proceso sin ambigüedad.

Saber más (*): hay lenguajes **interpretados** y **compilados**.

En los primeros, como Javascript, un **programa** llamado **intérprete ejecuta las sentencias** a la vez que las lee del fichero de texto donde están escritas. En estos casos, a los programas también se le suele denominar ***scripts*** o guiones.

En un compilado, como Java, debemos **previamente convertir el fichero de texto** a una 'traducción' mediante un **programa** llamado **compilador**. Ese fichero resultante es el que se ejecutará en el ordenador.

Nota: se podría detallar más la definición de programación y describir otras tareas, como el diseño y creación de interfaces gráficas de usuario (la manera en que el usuario interactuará con nuestro programa: ventanas, formularios, botones, etc) o la implementación del acceso/gestión de información externa (ficheros, base de datos,...).

No vamos a entrar en el curso en estas áreas. Nuestro objetivo es ayudarte a dar los primeros pasos para programar, centrándonos en cuestiones fundamentales que debes conocer. Con esa base, podrás seguir aprendiendo y completando tu visión sobre la programación, y tu capacidad para hacer cosas con ella.

Lenguaje de programación Javascript

Hay muchos **lenguajes de programación**, que se han ido desarrollando según distintos objetivos, cómo implementar características particulares o buscar soluciones a situaciones o ámbitos concretos. Por ejemplo:

- Posibilidad de ejecución en múltiples plataformas
- Rendimiento: utilizar menos recursos del ordenador, conseguir una velocidad más alta de ejecución...
- Sencillez de aprendizaje y uso
- Enfoque a una finalidad específica: cálculos científicos, aplicaciones de negocio, inteligencia artificial, desarrollo Web...

Desde el punto de vista formal, los lenguajes se distinguen también por el paradigma de programación que implementan, algo así como la 'filosofía' que siguen: imperativa, funcional, orientación a objetos,.... No vamos entrar en estas cuestiones teóricas, que no nos son imprescindibles a este nivel, pero trataremos de 'ubicar' al lenguaje que hemos escogido para este curso: Javascript.

Breve presentación de Javascript

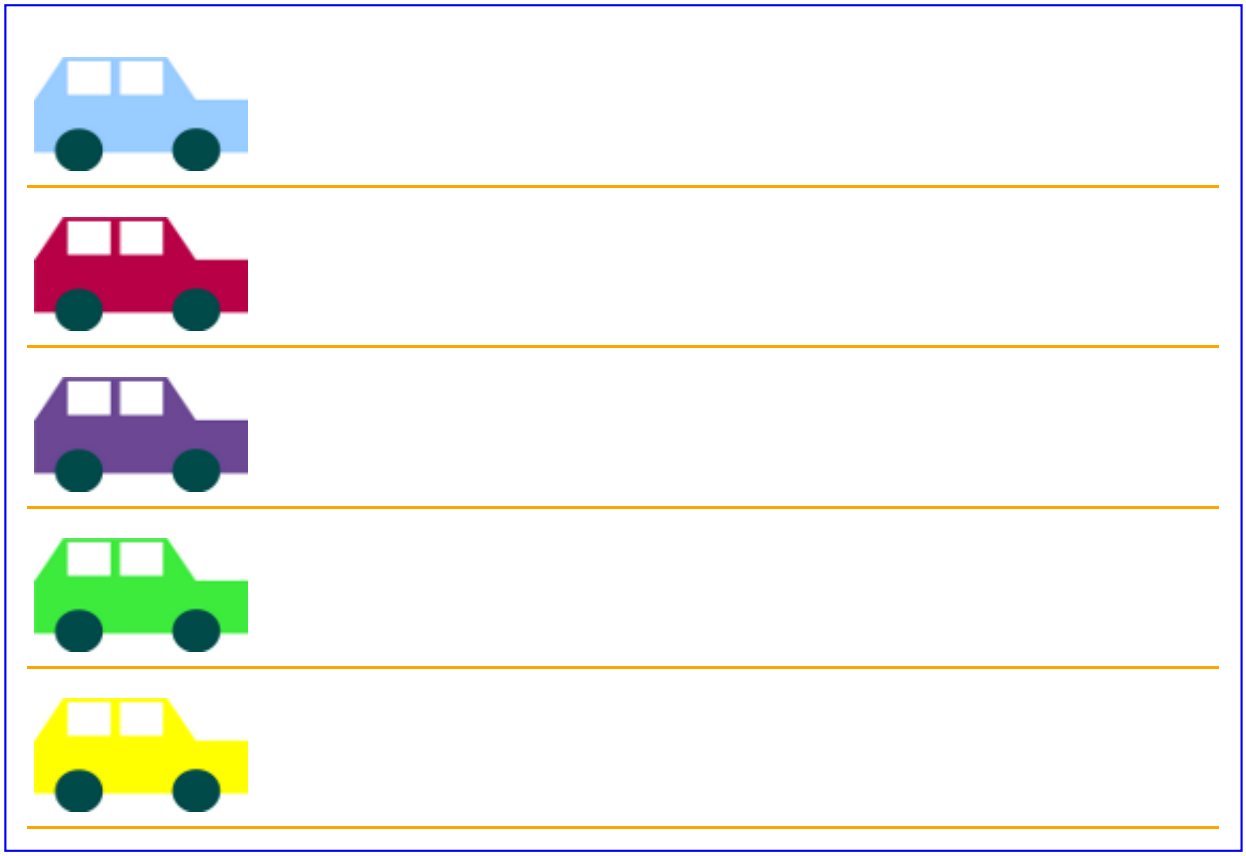
Javascript es un lenguaje que se creó para ser ejecutado solo en los navegadores (*) , con la finalidad de dar **interactividad y dinamismo a las páginas web**, lo que a su vez, limita las cosas que puede hacer (por ejemplo no puede acceder a la mayoría de recursos de tu ordenador, como hacen otros lenguajes).

Si estás impaciente por ver algo de acción, aquí te mostramos un simpático ejemplo que desarrollamos en nuestro curso de Javascript, una carrera donde los coches avanzan aleatoriamente, por lo que llegan en distintas posiciones cada vez que jugamos (si te gusta este lenguaje, esta formación podría ser tu siguiente paso):

Result Edit in JSFiddle

Carreras de coches!!

Curso online de **Digital Learning**: Programación Web con Javascript y jQuery



¡Comenzar!

Una aclaración, **no tiene relación con Java**, que es un lenguaje del que posiblemente hayas oido hablar. La semejanza en el nombre fue una cuestión de marketing, lo que no significa que no tenga algunos parecidos (como ocurre entre muchos lenguajes) a nivel de instrucciones y escritura.

Dentro de los lenguajes de la Web, tenemos dos grupos de lenguajes:

- **Lenguajes en la parte del servidor**, es decir, se ejecutan en el equipo al que solicitamos la página web y dependiendo de si hemos realizado alguna acción (por ejemplo, rellenar un formulario o hacer una consulta) nos pueden envían una página web configurada de acuerdo a esa acción previa.
- **Lenguajes en la parte cliente**, es decir, se ejecutan en el entorno de nuestro propio navegador y equipo

En la parte **servidor**, los lenguajes por antonomasia para la Web han sido PHP y Perl, pero existen otros muy populares de uso general como Python, Ruby, Java, etc. Son muy potentes pero exigen una cierta infraestructura para que funcionen (básicamente un servidor web con las funcionalidades adecuadas). Es decir, no podríamos probarlos en nuestro PC, con nuestro navegador sin más.

En la parte **cliente**, ha habido principalmente 4 opciones:

- **Javascript**: el estándar (**) y el que se ha impuesto sobre todas las demás opciones
- **JScript**: es una variante de Javascript implementada por Microsoft para su navegador IE, de poco uso al no ser soportada por el resto de navegadores.
- **Applets de Java**: son programas desarrollados en el lenguaje Java que se ejecutan en nuestro navegador gracias a un plugin (pequeño programa) que debemos instalarle. Fueron populares en un inicio pero también ha caído en desuso.
- **Flash:** es un programa propietario de la compañía Adobe y necesita también que instalemos un plugin para funcionar. Ha mantenido mucho más su popularidad que los Applets de Java, y se ha utilizado con profusión para crear contenido dinámico e interactivo utilizando el lenguaje ActionScript, con similitudes con Javascript. Ultimamente ha decaído ante el empuje de Javascript y el no ser soportado por los iPhone/iPad de Apple y otros entornos.

Es decir, **el único lenguaje de programación que puede ejecutarse en todos los navegadores modernos sin necesidad de instalar ningún programa o plugin adicional es Javascript**. Si queremos programar en la Web (además de sacar partido al nuevo estándar HTML5) es imprescindible conocerlo, y por ese hecho y por su sencillez de ejecución (no necesitamos más que un editor de texto y un navegador) es una buena opción para iniciarnos en el mundo de la programación.

Saber más (*): ya se puede usar Javascript en la parte del servidor web, utilizando entornos como Node.js, pero no vamos a incidir en esta cuestión. Simplemente es buena noticia para el que quiera profundizar más adelante en este lenguaje porque le da mucha mayor potencia y posibilidades.

Saber más (**): realmente el estándar se llama ECMAScript (publicado por el organismo de estándares ECMA) a partir del cuál se basan las distintas versiones de Javascript que van apareciendo. La última especificación es la ECMA-262 y la última versión Javascript es la 1.8.5 de 2010.

Editor online de código

Para facilitar las pruebas y mostrar ejemplos, vamos a utilizar un **editor online como el que te mostramos aquí:**

Código

+ -

Resultado

+ -

Ver resultado

Te explicamos su funcionamiento en el siguiente video, así como en el texto que viene a continuación de dicho video.
Más abajo, como aún no hemos empezado, encontrarás un pequeño ejemplo de código Javascript con el que podrás probar este editor online .

En el primer recuadro 'Código' de este editor, podemos escribir código Javascript simplemente incluyéndolo entre las etiquetas **<script></script>**.
Vemos como utilizar el editor:

- En el primer cuadro, con el título **"Código"**, incluiremos el código Javascript entre las etiquetas **<script>.....</script>**
- En el segundo cuadro, con el título " **Resultado"**, tras dar al botón '**Ver resultado**' que aparece debajo del mismo, obtendremos el resultado de ejecutar dicho código al igual que si lo hiciera un navegador.

Importante: si el código tiene **errores** y no puede ejecutarse, **no obtendrás nada** en el cuadro de resultados, ni siquiera mensajes de error

Copiar código al editor online: a lo largo de las páginas incluiremos instrucciones o scripts completos de Javascript de ejemplo. Aparecen en una especie de recuadro o visor, con cada linea numerada (los números: 1,2,3... no forman parte del código, solo nos facilitan la lectura y la localización de las sentencias) como vemos en esta imagen:

```
1 <script>
2 document.write("Hola, este es el resultado de mi primer script");
3 </script>
```

Para copiar esas lineas al cuadro del editor online, tienes varias opciones:

Puedes seleccionar todo el código con el ratón. Luego pulsar simultáneamente las teclas 'Control' y 'C' (copiar el texto que has seleccionado) y luego situándote en el cuadro primero del editor online denominado 'Código', pulsa las teclas 'Control' y 'V' (pegar el texto anteriormente seleccionado).
También puedes hacerlo a través de las opciones 'Copiar' y 'Pegar' del menú contextual que te aparece pulsando el botón derecho del ratón.

Si te sitúas sobre el código, te aparecerá una linea arriba justo de él. Pincha el icono 'Copy' según mostramos en la imagen de abajo. Te seleccionará automáticamente todo el texto y haz copia-pega tal como hemos explicado antes.

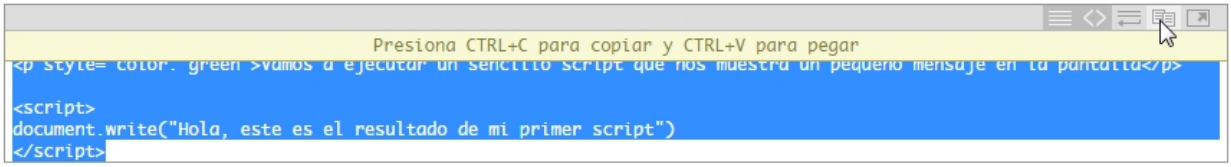


Imagen: icono 'copy' que nos facilita el copia-pega del código

Ejemplo del uso del editor online de código

Aquí tienes un sencillo código de ejemplo. Copia ese código en el primer cuadro del editor online de arriba y comprueba el resultado pulsando el botón 'Ver resultado':

```
1 <script>
2 document.write("Hola, este es el resultado de mi primer scrip
3 t");
  </script>
```

Incluir también HTML en el editor online

Puedes escribir también etiquetas HTML e incluso estilos internos CSS *ver nota en el recuadro de abajo*), como si estuvieras elaborando una página web en el editor online. De hecho, la forma en que escribimos Javascript en este editor no es más que una de las 3 formas posibles de incluir dicho código en una página web y que son muy similares, si conoces HTML a las que empleamos con los estilos CSS:

- **Dentro de una etiqueta HTML**. En vez de usar el atributo 'style' <p style="*reglas de estilo CSS*"> se emplea un manejador de eventos, como por ejemplo 'onclick' (`<button onclick="instrucciones Javascript">`)
- **Entre etiquetas**. En vez de <style></style>, en este caso lo incluiremos entre las etiquetas `<script></script>` Esta opción es la que hemos elegido en este curso!
- En un **fichero de texto externo** que contendrá el código Javascript. En vez de enlazarlo con <style src="fichero.css"> utilizaremos `<script src="fichero.js">`

Si no estás familiarizado con HTML, no te preocupes, porque no vamos a emplearlo en el curso (salvo la etiqueta
 que nos permite hacer salto de linea). Solo es para que sepas que esa posibilidad existe por si tienes conocimientos de ese lenguaje y quieres complementar algunos de los ejemplos/ejercicios que hacemos. Prueba a copiar este código al editor y comprúebalo tú mismo.

```
1 <h2>Primera prueba con el editor online</h2>
2 <p style="color: green">Vamos a ejecutar un sencillo script que
  nos muestra un pequeño mensaje en la pantalla</p>
3
4 <script>
5 document.write("Hola, este es el resultado de mi primer scrip
6 t");
  </script>
```

Si estás interesado en la creación de Webs, en Mayo de 2016 lanzaremos un curso de la serie 'Aprende a...' sobre WordPress y esperamos elaborar otro sobre HTML y CSS a continuacón.

Sintaxis y vocabulario de Javascript

Antes ya de empezar a practicar con código real, vamos a comentar brevemente cómo debemos escribir dicho código en Javascript y qué reglas debemos seguir.

- Como dijimos, los **scripts** (programas) se componen de una serie de **instrucciones**.
- Cada instrucción o paso lo denominamos **sentencia** (en inglés, *statement*).
- Cada sentencia la debemos acabar con un **punto y coma** (**;**) que indica su final.
  (*el uso del punto y coma es más una recomendación, que algo obligatorio. Si hacemos salto de linea al acabar una sentencia el intérprete del navegador entenderá el código, pero puede haber otros casos, por ejemplo con varias sentencias en la misma linea, donde pueda dar lugar a un error. Mejor acostumbrarnos a utilizarlo siempre*).

De esta forma, para que nuestro código sea claro, escribiremos cada instrucción en una nueva línea, de esta forma:

```
<script>
linea 1 de código Javascript;
linea 2 de código Javascript;
.....
linea N de código Javascript;
</script>
```

Lo vemos con un ejemplo muy simple donde calculamos un incremento de temperatura y lo mostramos en una ventana emergente en grados centígrados:

```
1 <script>
2 var grados1 = 23;
3 var grados2 = 30;
4 var incremento = grados2 - grados1;
5 alert("La temperatura subió: " + incremento + " °C");
6 </script>
```

Recordamos que la numeración de las líneas (1,2,3,4...) no forman parte del código. Los incluyen muchos editores y visualizadores de código para facilitarnos la lectura y poder identificar sentencias (por ejemplo, un depurador puede dar el mensaje: "error en línea X...").

No te preocupes si no entiendes aún el código. En próximos apartados explicaremos este tipo de sentencias y verás que comprendes todo. Por ahora, para hacerte una idea, puedes copiar el código al **editor online** que hemos incluido abajo y ver el resultado (en esta ocasión, ya te hemos incluido las etiquetas **<script></script>)**

Incluir comentarios en el código Javascript

Sea cual sea el lenguaje que utilicemos conviene incluir comentarios en nuestro código para hacerlo más claro. Tenemos dos opciones:

1. Comentar (en) una linea mediante ' **//** ' (doble barra inclinada ó *slash*)
2. Comentar varias líneas mediante ' **/*** ' y ' ***/** '

Ejemplo donde empleamos ambas formas:

```
<script>

var grados1 = 23;   //declaramos la variable 'grados1' y le asi
gnamos el valor '23'
var grados2 = 30;  //lo mismo para la variable grados2

/* comentario de varias líneas: ahora vamos a hacer una operaci
ón, restando ambas variables,
y asignado el resultado a una tercera variable,
que llamaremos 'incremento' */

var incremento = grados2 - grados1;

//a continuación imprimimos el resultado en un popup
alert("La temperatura subió: " + incremento + " °C");

</script>
```

Vamos a analizar estas sentencias para ver que contienen:

- **Palabras clave** (keywords) y **operadores** que forman parte del **vocabulario** de Javascript. Los explicaremos en próximos apartados pero para que te hagas una primera idea, en este ejemplo, las **palabras claves** serían '**var**' y '**alert**' que sirven para declarar una variable y presentar un mensaje respectivamente.
  Los **operadores** serían los signos '=', '–' , '+' que sirven para asignar un valor, realizar una

resta y para concatenar (unir) cadenas de texto respectivamente.

- **Nombres** y **valores**, que asignamos nosotros.
  En ese ejemplo, serían los **nombres** de las variables '**grado1**', '**grado2**', '**incremento**' y los **valores** '**23**' y '**30**' (en todos los casos podríamos haber elegido otros diferentes)
- **Expresiones**, que hacen algún tipo de operación con los operadores.
  En este ejemplo serían: la asignación del valor a una variable, la operación matemática '**grados2 – grados1**' y la unión de textos dentro del paréntesis de 'alert'

Algunas sentencias pueden contener a su vez un **bloque de instrucciones**. Estas instrucciones las incluimos entre dos llaves de apertura y cierre ' **{** ' y ' **}** ' que indican el inicio y final del bloque. Las llaves no llevarán el 'punto y coma' final de sentencia, pero sí las instrucciones que estén allí contenidas. Esto lo veremos al explicar las instrucciones condicionales "if" y de bucle "for" en unos próximos apartados

Saber más: podemos escribir **todo el código** de un script **en una sola línea**, incluso sin espacios en blanco (en el ejemplo que hemos visto habría que separar solo 'var' y 'gradosX', para distinguir ambas palabras), pero el código quedaría menos claro para leerlo y no se suele hacer así. Los 'punto y coma' marcarían la separación entre las sentencias:

```
var grados1=23;var grados2=30;var incremento=grados2-grados1;alert("La temperatura subió: " + incremento + " °C");
```

Módulo 2. Mostrar y solicitar información

Autor: Manuel Cabello

Versión 1.0

Fecha: Abril 2016

Digital Learning

Función alert()

En todo lenguaje de programación tenemos instrucciones que nos permiten imprimir información en pantalla, siendo de las primeras instrucciones en describirse, ya que nos permite mostrar el resultado de los programas que vamos creando. Al aprender un lenguaje, el primer programa que suele crearse es el famoso "Hola mundo" que nos presenta ese mensaje en nuestro monitor.

Vamos a ver por tanto dos instrucciones Javascript que nos serán de utilidad para mostrar información o resultados por pantalla. A diferencia de otros muchos lenguajes, la palabra clave no es "print" ni ninguna variación similar.

Comenzamos con **alert.**

Como ya vimos, `alert` es una **palabra clave** de Javascript (*ver nota abajo*) que hace **aparecer una ventana** o cuadro **emergente**, también llamado 'pop-up', donde podemos incluir un **mensaje**.

Vemos cómo debemos escribirlo:

```
alert("Este es un mensaje mostrado con alert");
```

Si nos fijamos, a '**alert**' le sigue un paréntesis **(...)** que encierra un texto entrecomillado con comillas dobles (**" "**).

Pruébalo en el editor online (*recuerda, entre las etiquetas* ***<script> </script>****. Si tienes algún bloqueador de pop-ups debes desactivarlo para permitir que aparezca*):

En cuanto al modo de escribir la sentencia habría valido igualmente que el texto estuviera entrecomillado con comillas simples (**' '**).
También es indiferente si dejamos espacios en blanco entre 'alert' y el inicio del paréntesis o entre los paréntesis y las comillas ya que Javascript no tiene en cuenta esos espacios, pero es más habitual escribirlo como lo hemos hecho.

Funciones *built-in* (internas): tanto **alert** como **prompt** (que veremos luego) son **funciones que incorpora Javascript**. Explicaremos más adelante el concepto de función pero basta saber por ahora que al incluirlas en nuestro programa, ejecutan una tarea determinada.

El **uso** de alert o prompt, muy extendido, **no se considera** actualmente una **buena práctica** ya que utilizan ventanas emergentes que bloquean el acceso a otras partes de la página, mientras no son aceptadas por el usuario. Hay alternativas pero no obstante las usaremos **instrumentalmente** para conseguir interactividad en los ejemplos, sin necesidad de complicar el código en estos primeros pasos.

Método document.write

La 2º instrucción que vamos a ver para mostrar información por pantalla es `document.write`
En este caso, la **información se mostraría en la misma página**, y no en un pop-up. Vemos como debemos escribirlo:

```
document.write("Este es un mensaje mostrado con document.write": ¡Hola mundo!);
```

Ya ves que es una manera muy similar a la instrucción `alert` y se le aplica las mismas consideraciones que hemos hecho antes (se puede emplear comillas simples y no importan los espacios en blanco).

Si lo probamos en el editor online, el texto se mostrará esta vez en el cuadro 'Resultados' (*recuerda incluirlo entre las etiquetas* **<script></script>**):

Ejemplo de cómo muestran mensajes en pantalla las funciones alert() y document.write()

Incluir etiquetado HTML

Si conoces HTML, te interesará saber que dentro de los paréntesis de alert y document.write, podemos escribir **etiquetas HTML entrecomilladas** que serán tomadas en cuenta por el navegador como tal código. Un ejemplo muy sencillo donde utilizamos la etiqueta
 (salto de línea):

```
1 document.write("Primera linea");
2 document.write("<br>");
3 document.write("Segunda línea");
```

Es decir, al ir entrecomillado, ha incluido el texto <*br*> y el navegador ha interpretado esa etiqueta HTML, haciendo el salto de línea.

Vemos otro ejemplo algo más elaborado. En este caso, mostramos parte del código de una página web donde tenemos etiquetado HTML (un párrafo y una lista) y además hemos incluido una sentencia document.write (entre las etiquetas <script></script> como siempre) en cuyo texto van etiquetas HTML.

Si lo pruebas en el editor online, verás que salvo por las aclaraciones que hacemos en el propio texto, no se habría podido distinguir cuándo hemos empleado HTML directamente o cuándo Javascript con HTML en su 'interior' (*utiliza el scroll del cuadro si no ves el resultado completamente)*:

```
1 <p>Vamos a escribir primero una lista con HTML directamente:
2 </p>
3 <ul>
4 <li>item 1 lista HTML</li>
5 <li>item 2 lista HTML</li>
6 </ul>
7 <script>
  document.write("<p>Esta lista en cambio, la hemos incluido a
8 través de un document.write (dw):</p><ul><li>item 1 lista dw
  -HTML</li><li>item 2 lista dw-HTML</li></ul>")
  </script>
```

Si te das cuenta, en este caso hemos llamado '**método**' a la instrucción **document.write** en vez de 'función' como hicimos con alert ó prompt. Son conceptos muy parecidos que explicaremos más adelante (de hecho tanto alert como prompt son también métodos). Por ahora basta con entenderlos intuitivamente: son instrucciones que al incluirlas en nuestros programas realizan determinadas tareas.

La función prompt()

La instrucción `prompt` nos permite solicitar información al visitante de la página. Vemos cómo se escribe:

```
prompt("Indique su edad");
```

Si lo probamos en el editor online (*entre etiquetas* ***<script></script>***), nos saldrá un pop-up con el texto que hayamos incluido y una casilla para introducir la información solicitada (datos númericos o texto).

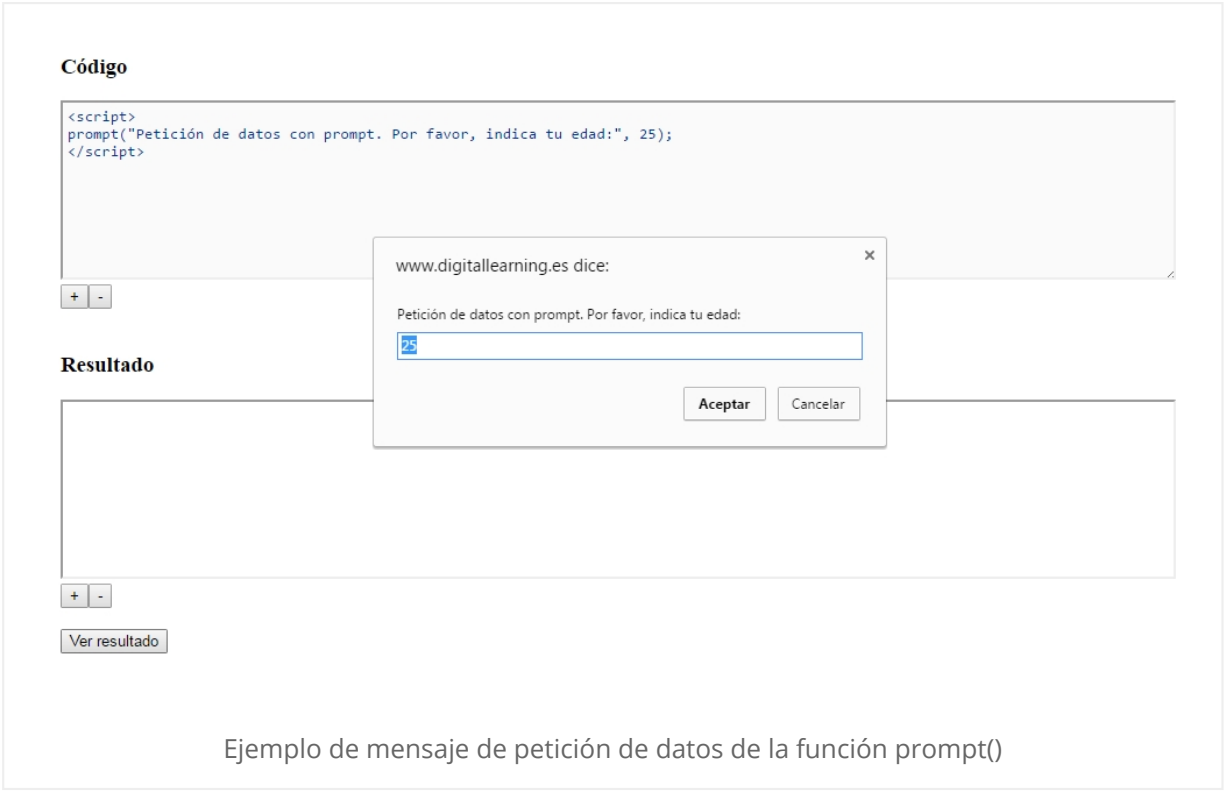
Se puede incluir una respuesta por defecto. Esta se mostrará en la casilla y puede ser modificada por el visitante. Ejemplo:

```
prompt("Indique su edad", 25);
```

Normalmente asignaremos la respuesta del usuario a una variable, para que según haya sido aquella, realicemos alguna acción definida en nuestro programa. El concepto de variable lo explicaremos a continuación y veremos como nos permitirá hacer muchas más cosas con nuestros scripts. Será cuando empecemos a pensar que estamos programando realmente.

Para que te des cuenta de ello, prueba el siguiente ejemplo (muy 'tonto') en el editor:

```
<script>
prompt("Soy capaz de multiplicar.\n Empieza introduciendo un
número. \n Número 1° = ");
prompt("Introduce otro número. \n Número 2° = ");
document.write("Como no se qué números has introducido, no p
uedo calcular el resultado. ¡ Vaya desastre de programa !");
document.write("<br>");
document.write("Vuelve después de estudiar los capítulos de
variables y operadores, y seguro que puedes programarme para
calcular cualquier multiplicación");
</script>
```



Ejemplo de mensaje de petición de datos de la función prompt()

Módulo 3. Variables y tipos de datos

Autor: Manuel Cabello

Versión 1.0

Fecha: Abril 2016

Digital Learning

¿Qué son las variables?

A través de un **lenguaje de programación** podemos **manipular datos** según nuestras necesidades.

Esos datos pueden variar cada vez que se ejecute el programa. Por ejemplo, podemos solicitar al usuario que introduzca datos, en formato numérico o de texto, a través de un formulario. Con esa información, nuestro script realizará el proceso que hayamos programado. Algunos ejemplos:

- Calcular y mostrar la mensualidad que debemos pagar al solicitar un préstamo, tras introducir la cantidad, el tiempo y el tipo de interés.
- Comprobar que el texto registrado en un campo email contiene el caracter '@', y si no, mostrar un mensaje de error.
- Presentar un pasaje de un libro al seleccionar el nombre de un escritor de una lista desplegable.

Las posibilidades son ilimitadas e iremos viendo ejemplos a lo largo de esta introducción.

¿Cómo **almacena** la **información** un lenguaje de programación? Pues utilizando lo que denominamos **variables**, contenedores de información temporal que les permiten guardar datos que pueden ser modificados por acción de nuestro programa.

Muy posiblemente hayas utilizado variables al estudiar matemáticas en el colegio. ¿Te acuerdas de las variables **x**, **y** de una **función**? Si tenías una expresión del tipo:

```
y = x+2
```

tanto '**x**' como '**y**' no eran más que símbolos que podían tomar cualquier valor.

Por ejemplo, si asignamos a '**x**' el valor '**5**', entonces '**y**' toma el valor '**7**':

y = x+2 = 5+2 = 7

Si '**x**' toma el valor '**8**', el valor que tomaría '**y**' sería '**10**' (y = 8+2), y así sucesivamente con cualquier número que asignáramos a **x**.

En definitiva, **x** ó **y** son solo 'nombres', una manera de identificar un número de forma genérica, que puede tomar cualquier valor.

Pues bien, las **variables** en programación son algo muy parecido, aunque en este caso veremos que no solo **pueden contener números**, sino también **textos** u **otro tipo de valores**.

Si aún sigues con dudas, podemos emplear un **símil** muy sencillo, asociando el concepto de variable a nuestro **buzón postal de correo**, aquél donde el cartero nos deposita las cartas que han remitido a nuestra dirección.

El buzón es la variable y la carta son los datos. Puede haber muchísimos buzones, pero podemos identificar a cada uno por su dirección (en el caso de las variables por su nombre), aunque para que esta comparación sea válida, simplificaremos diciendo que en nuestro buzón solo puede depositarse una carta. Si introducimos una nueva, se elimina la anterior.

Según este símil, cada vez que el programa va a consultar el valor de la variable para realizar alguna operación con ella (mostrar su valor en pantalla, realizar un cálculo, ...) verá la última carta que se haya depositado en el buzón.

Veremos más adelante que hay unas variables especiales, los **arrays** (matrices), que permite almacenar muchos valores en un solo contenedor de ese tipo.

Declaración y asignación de una variable

En Javascript construimos estos buzones (en el argot, '**declaramos una variable**') con la sentencia:

```
1 var miPrimeraVariable;
```

que como vemos se compone de la palabra clave '**var**' y de un nombre '**miPrimeraVariable**'.

Podemos elegir el nombre que queramos, tenga o no significado, **dentro de unas sencillas reglas** *ver recuadro abajo*), aunque conviene que sea descriptivo para facilitar la lectura del código del programa.

Ejemplos de algunos nombres que podríamos dar a nuestras variables:

- matricula
- ventas2015
- numeroHabitacion
- modeloCoche

Con esos nombres es fácil saber qué tipo de valores estamos almacenado en esas variables, mientras que con el nombre '**ghfp**', que también es válido, sería difícil, sobre todo si utilizamos muchas variables en nuestro programa.

Si tenemos que **declarar varias variables**, lo podemos hacer en una sola sentencia separando con comas, de la forma:

```
var variable1, variable2, variable3,.....  ;
```

Bien, y una vez que tenemos el buzón creado, es decir, la variable, ¿cómo depositamos una carta en él?, o lo que es lo mismo, **¿cómo almacenamos un dato/información en la variable?** Pues con la siguiente sentencia, utilizando el signo

de '**igual**' (**=**) :

```
1 miPrimeraVariable = 7;
```

En esta sentencia hemos asignado el valor 7 a nuestra variable 'miPrimeravariable'.

Podríamos simplificar y en **una sola instrucción** realizar las dos acciones: **definir la variable y asignarle un valor**:

```
1 var miPrimeraVariable = 7;
```

Vamos a practicar un poco con todo esto con un sencillo ejercicio (pincha el botón):

Normas y consejos para crear nombres de variables:

- No podemos utilizar palabras claves reservadas en Javascript, como 'for', if', 'new'... porque forman parte de la sintaxis del lenguaje. Aquí tienes la lista completa para el estándar ECMAScript 6 (recuerda que ésta es la norma oficial que define al lenguaje Javascript)
- Los nombres solo pueden contener letras, números, signo del dólar ($) y guión bajo (_).
  Es decir, no puede contener espacios en blanco, guiones medios (-), u otros tipo de caracteres/símbolos.
- las letras pueden ser minúsculas y mayúsculas, distinguiéndose entre ambas. Es decir, "NIF" es distinto a "nif" y a "Nif".
- Aunque no es obligatorio, se recomienda emplear solo minúsculas. En el caso de nombres compuestos de varias palabras, al no permitirse el espacio en blanco, se aconseja utilizar el estilo *camelCase* , comenzando con mayúscula las palabras posteriores a la 1ª (asemejando las joróbas de un camello).
  Ejemplos: "tallaCamiseta", "finPlazoMatricula"

Tipo de variables y datos

Las variables pueden contener diversos tipos de datos (***data types***), Según sean dichos tipos pueden tener un tratamiento diferenciando.

Los dos tipos de datos que nos resultarán más familiares son:

Numbers (números)

Números enteros y decimales, positivos y negativos.

En otros lenguajes de programación, se distingue entre distintos tipos de números (int: entero, float: decimal,...), por lo que hay que indicarlo al declarar la variable, pero en Javascript no.

Ejemplo: `var notaMedia = 7.52, numeroAlumnos = 45, variacionAlumnos = -5;`

Nota: los decimales no pueden expresarse como fracciones (ej: 1/2) y deben separarse por puntos (.), según notación inglesa, y no por comas (,).

Practica asignando valores numéricos:

Ejercicio variables numéricas

Strings (cadenas)

Una o varias **cadenas** de caracteres alfanuméricos (letras, números, signos puntuación...).

Las denominamos tipos de datos o variables de **texto**, porque las empleamos para almacenar información de esa naturaleza o porque sus valores los vamos a tratar como texto.

Ejemplos: `var nif = "12332112W", ciudad= "Santiago de Compostela", codigoPostal = "28001", telefono = "003423145627"; ocupacion = "Estudiante de Biología";`

Como vemos en los ejemplos, el código postal o el teléfono, aunque contengan solo números, no nos interesa tratarlos de esa forma (en el sentido de aplicarles operaciones matemáticas: sumarlos, restarlos, comparar si son mayores o menores...). Tiene más sentido tratarlos como cadenas de texto, por lo que asignamos su valor entrecomillado.

Si esto te causa confusión, prueba con el siguiente script de ejemplo, donde los mismos valores '3' y '5', son definidos en unas variables como textos (string) y en otras como números. Al aplicarles el operador '+' verás que produce un resultado distinto según se trate de un caso u otro (explicamos más adelante este operador '+', pero es sencillo de entender: suma los valores cuando se aplica a variables numéricas y los une cuando sea aplica sobre variables de texto):

```
<script>
var numTexto1= "3"; //variable tipo string-texto al asign
ar el valor entrecomillado
var numTexto2= "5"; //igual tipo que la anterior
document.write("Al usar el operador + con las variables s
tring 3 y 5 obtenemos: ");
document.write(numTexto1 + numTexto2);  //te aparecerá la
cadena "35" , al haber sido tratadas como texto
var num1 = 3 ;                  //variable tipo number-númer
o al asignar el valor sin comillas
var num2 = 5;                   //igual tipo que la anterior
document.write("<br>");
document.write("Al usar el operador + con las variables n
umber 3 y 5 obtenemos: ");
document.write(num1 + num2);  //te aparecerá el resultado
de la suma: 8, al haber sido tratado como números
</script>
```

Práctica asignado valores de texto:

Ejercicio variables de texto

Además de los dos anteriores, podemos tener estos otros tipos de datos:

Booleans

El nombre viene de la lógica de Boole. Pueden tomar solo dos valores: '**true**' o '**false**' (verdadero/falso).

Ej: `var maximaPuntuacion= true, maximaPuntuacion = false;`

Se utilizan mucho en las sentencias condicionales, a modo de interruptor, pudiendo equiparar los valores true/false, en ese símil, a 'encendido' /apagado', es decir, que hace que el programa se siga ejecutando por una instrucción o por otra diferente: "Si es verdad, sigue ejecutando X, si es falso, haz Y".

Veremos su utilización práctica en el apartado de flujos del programa.

Objects (objetos)

Este concepto podemos asociarlo, de forma simplificada, a un ente u objeto de la vida real que contiene diversas propiedades características que lo definen.

Explicaremos más sobre este concepto en un apartado posterior , sobre todo para introducir los objetos que incorpora Javascript. La creación de nuestros propios objetos, típica del estilo o paradigma de programación 'orientada a objetos' va más allá del objetivo de esta formación y no lo tratemos en este curso (publicaremos próximamente sobre ello en otros contenidos)

Ejemplo declaración de propiedades de un objeto: `var coche {marca:"Seat", modelo: "Ibiza", matricula:"1400AXZ", color:"rojo"}`

Arrays (matrices)

Pueden **contener múltiples valores** (piensa en un buzón con diversos compartimentos numerados en su interior) y son objetos. Vamos a ver aquí solo su definición y en apartados posteriores, hablaremos más sobre ellas y de cómo podemos manipularlas.

Ejemplo: `var granPoblacion = ["Madrid", "Barcelona", "Valencia", "Sevilla"];`

En su definición date cuenta cómo se emplean:

- corchetes [ ]: para incluir los valores
- comas (,): para separar los valores dentro del array

Undefined (indefinido)

Se da este tipo de datos cuando se declara una variable, pero no se le asigna ningún valor.

Ejemplo: `var cantidad;`

Como 'rarezas', tenemos otro *data type* denominado **Null** (nulo) que se considera un objeto (muchos contemplan este *data type* como un 'fallo' del lenguaje Javascript), y el valor **NaN** ('not a number': no un número), que curiosamente tiene el tipo de dato 'number' y que es el resultado de una operación que no tiene sentido. Ejemplo: `var resta = 25 - "texto";`

Typeof

Si queremos averiguar el tipo de dato que contiene una variable podemos utilizar la función typeof.

Vamos a ver un ejemplo donde asignamos los siguiente valores a una lista de variables:

- notaMedia = 2.5
- coche = “Opel”
- final = true
- notasAlumnos = [9, 8, 5, 7]

y luego vamos a ver con la funcion ‘typeof’ de qué tipo de datos es cada variable:

```
<script>
var notaMedia = 2.5; var coche = "Opel"; var final = true;
var notasAlumnos = [9, 8, 5, 7];
document.write("Variable notaMedia es del tipo: " + typeof
(notaMedia) + "<br>");
document.write("Variable coche es del tipo: " + typeof(coc
he) + "<br>");
document.write("Variable record es del tipo: " + typeof(re
cord) + "<br>");
document.write("Variable notasAlumnos es del tipo: " + typ
eof(notasAlumnos));
</script>
```

Nota: si ejecutas este código en el editor online, date cuenta que el array que hemos incluido (notasAlumnos) da como resultado un tipo ‘objeto’.

Javascript tiene tipo de datos ó ‘tipado’ dinámico

Javascript tiene un tipo de datos dinámico. Esto significa que una variable puede declararse conteniendo un tipo de dato y, más adelante, asignarle otro tipo mediante una simple asignación. Ejemplo:

```
<script>
var cambioTipo = "hola";  //es un string
cambioTipo = 89;  //es un número
</script>
```

Debido a esto, en Javascript no es necesario indicar el tipo de dato que va a tener una variable a la hora de declararla. Es decir, siempre empleamos la palabra clave ‘var’, sin importar si es un número, un texto o cualquier otro tipo.

Tipo de datos estricto: otros lenguajes, como Java, tienen 'tipado' estricto, es decir, al declarar una variable debemos indicar de qué tipo es.
Por ejemplo, si es una cadena de texto, debemos declararlo con la palabra clave 'String': `String nombre = "Sandra"` y si es un número entero, se debe declarar con la palabra clave 'int': `int año = 2015`.

En este tipo de lenguajes, podemos cambiar el valor asignado, siempre que sea del mismo tipo. Es decir, si hemos definido una variable 'String' y le asignamos un dato de otro tipo (por ejemplo, el número 7), el programa daría un error al compilarlo (salvo que hagamos una conversión previa de tipos mediante una función).

Los lenguajes de tipado dinámico, como Javascript, se consideran más flexibles y fáciles de utilizar porque no tenemos que tener en cuenta todas estas cosas, ahora bien, debido a este menor 'control' pueden producirse más errores, que a veces son difíciles de detectar.

Arrays

Como adelantamos en la página anterior, los **Arrays** ó matrices pueden **contener múltiples valores**, como un buzón con diversos compartimentos numerados en su interior. Son del tipo objetos. Vemos como trabajar con ellas:

Ejemplo: `var granPoblacion = ["Madrid", "Barcelona", "Valencia", "Sevilla"];`

En su definición date cuenta cómo se emplean:

- corchetes [ ]: para incluir los valores
- comas (,): para separar los valores dentro del array

En el array, los valores (que pueden ser numéricos, de texto o una combinación de ellos) se encuentran en posiciones numeradas, empezando a contar desde el cero, no desde el uno. Es decir, en el ejemplo anterior los valores de cada elemento del array (a los que se accede con *nombreArray*[n], siendo n= 0,1,2...) serían:

- granPoblacion[0]: Madrid
- granPoblacion[1]: Barcelona
- granPoblacion[2]: Valencia
- granPoblacion[3]: Sevilla

Podrías añadir o cambiar valores dentro del array. Por ejemplo, la sentencia: `granPoblacion[4] = "Zaragoza";` añadiría este valor detrás del valor 'Sevilla'.

Vamos a ilustrarlo con un ejemplo. Puedes copiar este script al editor (abajo):

```
<script>
var granPoblacion = ["Madrid", "Barcelona", "Valencia", "S
evilla"];       //declaramos el array
document.write("granPoblacion[2]: " + granPoblacion[2] + "
<br>");         //imprimimos el valor '2'
granPoblacion[2] = "Málag
a";                                                 // cambia
mos el valor '2'
document.write("Cambiamos Valencia por: " + granPoblacion
[2] +"<br>");    //imprimimos el nuevo valor '2'
granPoblacion[4] = "Zaragoz
a";                                        //añadimos el valo
r Zaragoza al array
document.write("Hemos añadido al array: " + granPoblacion
[4]);           //imprimimos el nuevo valor
</script>
```

También podrías crear un array vacío y posteriormente irle agregando elementos tal como hemos visto. Un ejemplo:

```
var ciudades = [];
ciudades[0] = "Cuenca";
ciudades[1] = "Gijón";
ciudades[2] = "Granada"
```

En un apartado posterior, veremos más posibilidades que nos ofrecen los arrays para manipular los valores que almacenan, usando métodos (funciones).

Módulo 4. Expresiones y Operadores

Autor: Manuel Cabello

Versión 1.0

Fecha: Abril 2016

Digital Learning

Expresiones y Operadores

Javascript nos permite realizar operaciones matemáticas como ya hemos visto en un apartado anterior. En ella calculábamos un incremento de temperatura, restando la temperatura final de la inicial: `var incremento = grados2 - grados1;`

En Javascript a una acción u operación que nos da un resultado, como el ejemplo anterior, se le denomina '**Expresión**'.

No solo hay expresiones asociadas a **operaciones aritméticas**; tenemos también **operaciones** donde podemos **comparar valores** (si son iguales/distintos o mayores/menores), **operaciones lógicas** (si una o varias condiciones se cumplen) o incluso **operaciones con textos** (concatenar/unir cadenas de textos).
También se considera una expresión, aunque de un tipo distinto, el **asignar un valor a una variable**. Ya vimos ejemplos en ese apartado.

Para todas estas operaciones empleamos los denominados '**operadores**', de los que ya hemos visto algunos. Listamos a continuación una lista de operadores que se utilizan con mucha frecuencia y en las siguientes páginas veremos ejemplos para entender su funcionamiento:

- Para **asignación de valores** a variables usamos el operador 'igual' (**=**)
- **Aritméticas**: los muy conocidos de suma (**+**) , resta (**–**), división (**/**), multiplicación (*****), más otros tres menos habituales: módulo (**%**), incremento (**++**), decremento (**––**)
- Para operaciones con **textos**, usamos el operador 'más' (**+**), que permite concatenar (unir) varias cadenas de texto
- **Comparativas**: igual (**==**), estrictamente igual (**===**), distinto a (**!=**), estrictamente distinto a (**!==**), mayor que (**>**), mayor o igual que (**>=**), menor que (**<**), menor o igual que (**<=**)
- **Lógicas**: el operador lógico 'Y' (**&&**), el operador 'O' (**||**) y la negación lógica (**!**).
  También es muy habitual nombrarlos por su denominación en inglés: AND, OR, NOT,

respectivamente (el símbolo del operador sería el mismo)

Saber más: los operadores estrictamente igual (===) y estrictamente distinto (!==), no solo comparan que las variables tenga el mismo valor, sino que sean también del mismo tipo. Hay quienes aconsejan utilizar siempre esta versión 'estricta' en vez de los 'normales', pero encontrarás muy a menudo ambas versiones, por lo que nosotros también las hemos utilizado indistintamente a lo largo de los ejemplos.

Operador de texto

Como hemos indicado en el apartado anterior, el operador '+' aplicado a dos variables de texto hace que sus valores se concatenen, es decir, que sus cadenas de texto se unan o junten una detrás de otra. Por ejemplo si tenemos dos variables:

- var texto1 = "cadena1"
- var texto2 = "cadena2"

El resultado de "texto1 + texto2" será: "cadena1cadena2"

Puedes probarlo en el editor online con este otro ejemplo:

```
<script>
var nombre = "Sandra";
var apellido1 = "Guzmán";
var apellido2 = "Castillo";
document.write("Resultado operador '+' con 3 variables de te
xto: nombre + apellido1 + apellido2 = ");
document.write(nombre + apellido1 + apellido2);
document.write("<br><br>" + "Imprimimos ahora un espacio int
ercalado para presentar mejor el nombre completo: ");
document.write(nombre + " " + apellido1 + " " + apellido2)
</script>
```

Veremos a continuación que el operador '+' realiza la suma aritmética cuando se aplica sobre dos variables numéricas. JavaScript distingue cada situación y aplica bien la concatenación o la suma, según el tipo de dato que se trate. Eso sí, si mezclamos ambos tipos, texto y número, siempre hará una concatenación. Compruébalo tú mismo con este ejemplo:

```
<script>
var nombre = "David";
var cifra = 15;
document.write(nombre + cifra);
</script>
```

Nota: en JavaScript, como no declaramos el tipo de dato que contiene una variable, ni forzamos a que solo admita de un determinado tipo (cadena texto, numérico, boolean...), se pueden producir resultados no esperados, que pueden incluso depender del intérprete (navegador) que ejecute el programa.

Por ejemplo si en el anterior ejemplo, a la variable 'nombre' le asignamos el valor "3" (será cadena de texto, al ir entrecomillado), puede ocurrir:

- Si hacemos la operación '**nombre + cifra**', obtener como resultado "315", tratando ambos valores como texto (string)
- Si se ejecuta '**nombre * cifra**' puede que obtengamos 45, tratando a ambos como números, ya que el operador '*****' no existe para las variables tipo texto (string) y el intérprete trata de aplicar la operación que pueda tener sentido.
- En otras combinaciones podemos obtener un 'NaN' (*non a number*: no es un número) que citamos en el apartado de tipos de datos.

Aunque JavaScript nos de esa flexibilidad y 'arregle' por sí mismo algunos casos, como el que se pueda multiplicar un texto por un número, es una **buena práctica controlar y cambiar el tipo de dato** al uso que le vayamos a dar. Por ejemplo, si pedimos al usuario que introduzca un número mediante una instrucción prompt,

éste se va a almacenar como texto. Si lo vamos a utilizar en el programa como un número (hacer operaciones matemáticas, comparar con otros números...) es mejor que le cambiemos el tipo de dato a 'number' antes de empezar a usarlo. Veremos en la próxima sección, que hay **funciones** que nos permiten esa **conversión** de 'string' a 'number' y viceversa.

Operadores ariméticos

Como indicamos al inicio, en JavaScript (al igual que en la mayoría de lenguajes) disponemos de los **operadores aritméticos** usuales: **suma**, **resta**, **división** y **multiplicación**, y algunos otros con los que posiblemente estés menos familiarizado (**módulo**, **incremento/decremento**).

Vamos a ver a través de unos ejemplos el uso de estos operadores. Puedes pegar el código en el editor online para comprobar su resultado y por supuesto modificar el código para probar por ti mismo (en estos ejemplos hemos incluido ya las etiquetas <script>).

```
1 <script>
2 //aunque no es necesario dejar espacios en blanco entre núme
  ros y operadores, por claridad, los hemos separado con uno
3 var operacion1 = 3 - 1 * 4;
4 var operacion2 = 8 + 6 / 2;
5 var operacion3 = 10 % 3;
6
7 document.write("operacion 1 = " + operacion1 + '<br>' + "ope
  racion 2 = " + operacion2 + '<br>' + "operacion 3 = " + oper
8 acion3);
  </script>
```

El operador módulo (**%**) que quizás no conozcas, al aplicarlo entre dos números o variables numéricas, nos da como resultado el resto de la división de ambos números o valores. En el ejemplo, 10 % 3, nos da 1, en 10 % 4, nos daría 2, y en 10 % 5, nos daría 0 al ser una división exacta.

Orden de aplicación de los operadores ariméticos

Si has ejecutado el anterior ejemplo, ¿se han obtenido los resultados que esperabas? En Javascript, como suele suceder en cualquier lenguaje de programación, hay una **prelación** de los **operadores multiplicación y división sobre la suma y la resta**, es decir, hay un **orden de aplicación** por el que las operaciones " **1 * 4** " y " **6 / 2** ", se realizan antes que las demás.
Si queremos alterar ese orden, podemos emplear paréntesis, por ejemplo en la forma:

- (3 – 1) * 4
- (8 + 6) / 2

Si sustituyes en el anterior código verás la diferencia de resultados.

Uso de números y variables numericas

Otra cuestión que ya hemos visto en anteriores ejemplos (como el cálculo de un precio con descuento e IVA), es que podemos poner en la expresión tanto valores numericos directamente (1,2,...), como variables que tengan un valor numérico asignado, o 'mezclar' ambas formas. Podemos utilizar números enteros y decimales (separados por punto ' . '), tanto positivos como negativos. Ejemplo:

```
<script>
//calculamos la velocidad de un coche, pasado un tiempo, debido al efecto de una desaceleración de -2,5 m/s2

var velocidad1 = prompt("Introduce la velocidad inicial (m/s) = ", 30);  //por defecto, 30 m/s
var tiempo= prompt("Introduce el tiempo transcurrido (s) = ", 10); //por defecto 10 s
var velocidad2 = velocidad1 -2.5 * tiempo  //-2,5 debemos ponerlo como -2.5 (punto como separador decimal)

document.write("Velocidad tras " + tiempo + " s = " + velocidad2 + " m/s");
</script>
```

Combinación de operadores

Podemos combinar los operadores aritméticos con el operador de asignación " = " para simplificar expresiones. Mostramos algunos ejemplos de estas equivalencias:

- La expresión `num = num + 5` equivale a: `num += 5`
- La expresión `num = num - 2` equivale a: `num -= 2`
- La expresión `num = num * 7` equivale a: `num *= 7`
- La expresión `num = num / 4` equivale a: `num /= 4`
- La expresión `num = num % 4` equivale a: `num %= 4`

Ten cuidado porque si escribes los operadores en orden inverso, poniendo delante el operador de asignación '**=**', en el caso de '**+**' y '**-**', le vas a dar a la variable el valor positivo o negativo, respectivamente, del número que querías sumar o restar, y en el caso de '*****' y '**/**' se producirá un error y no obtendrías nada.

Un ejemplo para visualizarlo mejor:

```
1  <script>
2  var num = 8;
3  document.write("El valor de num es: " + num + "<br>");
4  document.write("Resultado de num += 3: ");
5  document.write(num += 3);
6  document.write("<br>" + "Resultado de num =+ 3: ");
7  document.write(num =+ 3);
8  document.write("<br>" + "Resultado de num =- 3: ");
9  document.write(num =- 3);
10 </script>
```

Operadores incremento y decremento

También presentamos los **operadores incremento** (**++**) y **decremento** (**- -**). Como sus nombres indican, **aumentan o disminuyen en una unidad** el **valor** de la **variable** numérica a la que se aplican.
Se pueden **aplicar** a la **izquierda** o a la **derecha** del nombre de la **variable**. En el primer

caso (izquierda), el incremento/disminución se realiza antes de la asignación, y en el segundo (derecha), toma efecto con posterioridad.

Seguro que lo entendemos mejor con un ejemplo:

```
<script>
var num1Inicio = 6; num2Inicio = 6;
var num1Final = num1Inicio++; var num2Final = ++num2Inicio;
document.write("Partimos de una valor = 6" + "<br>");
document.write("Cuando aplicamos num++, obtenemos: " + num1F
inal + "<br>")
document.write("Cuando aplicamos ++num, obtenemos: " + num2F
inal + "<br>");
</script>
```

El incrementar o disminuir en una unidad mediante este operador se utiliza mucho en los contadores para los bucles 'for', donde se va repitiendo una instrucción hasta que ese contador alcanza un valor. Lo veremos en ese apartado con un ejemplo.

Objeto Math

Además de estos operadores, Javascript, como sucede como otros muchos lenguajes de programación, nos ofrece más funciones para operar con números. Con ellos podremos calcular por ejemplo logaritmos, senos/cosenos, elevar a un exponente, redondear cifras, calcular máximos/mínimos... Lo veremos en el apartado posterior: 'Métodos del objeto Math'

Operadores de comparación y lógicos

Estos operadores son **muy utilizados para comprobar si se cumple una determinada condición** (o condiciones). Según se cumpla o no, el programa ejecutará una determinada instrucción.

Los veremos en el apartado de flujos del programa más en detalle, pero vamos a incluir un par de ejemplos para que puedas tener una primera idea.

Ejemplo con operadores de comparación ('mayor que', 'igual que'):

```
<script>
//vamos a pedir dos números por pantalla y asignar cada uno a una variable, input1 e input2 respectivamente

var input1 = prompt("Introduce un número :");
var input2 = prompt("Introduce otro número (igual o distinto al anterior):");

//Vamos a comparar ambos números: si es 1° es mayor que el 2° (>) o si son iguales (==)
//Podríamos utilizar también el comparador estrictamente igual (===)

if (input1 > input2) {
    document.write("El primer número es mayor que el segundo");
    }
    else if (input1 == input2) {
          document.write("Has introducido dos números iguales");
          }
     else {
           document.write("El primer número es menor que el segundo");
          }
</script>
```

Ejemplo con operadores de comparación y lógicos (‘mayor ó igual que’, ‘igual que’, ‘menor que’, ‘distinto de’, ‘Y lógico’):

```
<script>
var numero1 = prompt("Introduce un número, positivo o negat
ivo :");

if (numero1 >= 0 && numero1 % 2 == 0) {
    document.write("El número es positivo y par");    //cons
ideramos al '0' par y positivo
    }
    else if (numero1 >= 0 && numero1 % 2 != 0) {
            document.write("El número es positivo e impa
r");
            }
    else if (numero1 < 0 && numero1 % 2 == 0) {
            document.write("El número es negativo y par");
            }
    else  {
          document.write("El número es negativo e impar");
          }
</script>
```

Nota: en el 2º ejemplo hemos empleado el operador módulo (**%**), que nos da el **resto de una división**. Por ejemplo, si hacemos la división entera de 9 entre 2 (9/2) sin sacar decimales, tendremos que: 9 es el dividendo, 2 es el divisor, 4 es el cociente y el resto es 1. Se cumple: 4 * 2 + 1 = 9

En este caso, al dividir el **valor** de **numero1** por **2:**

- si el resto es igual a 0 (**== 0**), significa que el número es **par.**
- si el resto no es igual a cero (**!= 0**), es **impar**.

El operador ' **&&** ' (**Y** lógico) exige que se cumplan las dos condiciones que hay a cada lado de ese operador.

Por ejemplo: `(numero1 >= 0 && numero1 % 2 == 0)` exige que se cumpla:

- 'numero1' sea mayor o igual (**>=**) que cero

Y (**&&**)

- 'numero1' sea par (**% 2 == 0**)

Módulo 5. Funciones y Métodos

Autor: Manuel Cabello

Versión 1.0

Fecha: Abril 2016

Digital Learning

Funciones

Las **funciones** son un elemento muy utilizado en la programación. Empaquetan y 'aíslan' del resto del programa, una parte de código que realiza alguna tarea específica.

Son por tanto un conjunto de instrucciones que ejecutan una tarea determinada y que hemos encapsulado en un formato estándar para que nos sea muy sencillo de manipular y reutilizar.

Lo vemos con un ejemplo muy simple, una función que denominamos "saludo( )", que nos da una idea del funcionamiento general:

```
<script>

//definimos la función, que en este caso la llamamos saludo
()

function saludo() {
     document.write("Hola, este es el resultado de la funci
ón saludo");
}

//llamamos a la función saludo() para que ejecute sus instr
ucciones

saludo();

</script>
```

Copia el código en el editor y ejecútalo.

Como puedes ver en ese código de ejemplo, para utilizar una función debemos de hacer dos cosas:

Declarar la función

Corresponde al bloque: `function saludo() {.....}`

Podemos distinguir los siguientes elementos en este bloque de código que define la función:

- La palabra clave '**function**'
- El **nombre** que queremos darle a dicha **función**, en este caso: **saludo**
- Unos **paréntesis** que añadimos al nombre para identificarla como función: **saludo()**
- Un **bloque de instrucciones** que queda encerrado entre llaves **{ }** . En este caso solo hemos incluido una (un document.write), pero podrían ser varias. Las instrucciones deben llevar al final el correspondiente punto y coma (;)

Llamar la función

Lo hacemos con la instrucción: `saludo()`

En este ejemplo, puedes ver que el nombre de la función, tanto al declararla como al llamarla, tiene unos paréntesis vacíos. En este caso, la función siempre devuelve el mismo resultado. En este ejemplo, muestra una frase que siempre es la misma.

Veremos ahora que también podemos pasarle valores a la función, incluyéndoselos dentro de los paréntesis, y dependiendo de éstos, nos podrá devolver resultados distintos.

Función con parámetros

Vamos a crear una función muy simple que nos dará un resultado dependiendo del valor que le pasemos. En este caso calculará el volumen de una esfera según el valor del radio (R). La fórmula de dicho volumen es: $4/3 * \pi * R^3$

Nota: el objeto Math de JavaScript nos puede dar el valor del número pi (π) y también tiene una función para calcular potencias, pero no lo utilizaremos porque aún no lo hemos visto. Para el nº pi, utilizaremos un valor aproximado de 3,14.

Primero veamos el código de este sencillo script sin emplear ninguna función. En él solicitamos al usuario que introduzca el valor del radio de la esfera cuyo volumen desea calcular, hacemos el cálculo y lo mostramos en pantalla:

```
var radio = prompt("Introduce la longitud del radio de la es
fera: ");
var volumen = 4/3 * 3.14 * radio*radio*radio;
document.write("El volumen de la esfera es: " + volumen);
```

Vamos ahora a encapsular el cálculo del volumen en una función que denominaremos volEsfera(), de la siguiente forma:

```
function volEsfera(x) {
    var volumen = 4/3 * 3.14 * x*x*x;      //cálculo del volum
en
    return volumen;                        //valor que devuelv
e la función
}
```

Para utilizar la función en nuestro código, debemos definirla y luego llamarla de la siguiente forma:

```
<script>

function volEsfera(x) {
 var volumen = 4/3 * 3.14 * x*x*x;
 return volumen;
}

/* Aquí y/o antes de la función, podría ir más código, si e
l script realiza más cosas.
En el momento en que quisiéramos utilizar la función, la ll
amaríamos con: volEsfera(argumento) */

var radio = prompt("Introduce la longitud del radio de la e
sfera: ");
document.write("El volumen es: " + volEsfera(radio));

</script>
```

Como ves, en este caso hemos declarado la función, no como volEsfera(), sino como **volEsfera(x)**. Al 'x' dentro del paréntesis le llamamos **parámetro** de la función (no es necesario que se llame 'x', podríamos haberle puesto cualquier otro nombre) y al valor que le pasamos, en este caso el de la variable 'radio', le llamamos **argumento**.

Una **función puede no tener parámetros** (como el primer ejemplo: la función saludo()) **o tener uno** (como la función volEsfera(x)) **o varios** (vemos a continuación un ejemplo con dos).

No tienen por qué coincidir los nombres de los parámetros y el de los argumentos, como hemos visto en el ejemplo (x ≠ radio) pero sí **debe haber el mismo número**. Es decir, si la función tiene 2 parámetros, debemos pasarle 2 argumentos, como vemos en el siguiente ejemplo:

En este caso, es un script con una función de dos parámetros (precio y descuento) que nos calculan el PVP, aplicando además el IVA:

```
<script>
//introducimos un precio y descuento a un producto
var precio = prompt("Introduzca Precio producto (en euros):
");
var descuento = prompt("Introduzca descuento (en euros):
");

//definimos la función calculoPVP, que tiene como parámetro
s los valores de precio y descuento
function calculoPVP(precio, descuento) {
   var IVA = 1.21;
   var PVP = (precio - descuento) * IVA; //en la variable P
VP hemos almacenado el PVP calculado
   return PVP;  //la función devuelve el valor asignado a l
a variable PVP
}

document.write("Precio= " + calculoPVP(precio,descuento) +
 " €");
</script>
```

Return: devolución de un resultado

En el primer ejemplo de la función saludo(), la función ejecutaba un 'document.write', es decir mostraban una frase en pantalla, con lo que ya nos daba un resultado. Pero ¿qué ocurre, si en vez de ésto, lo que calculan es una valor o una serie de valores que queremos utilizar en el resto del programa?

En estos casos, para devolver el valor calculado por la función se utiliza la palabra clave '**return**'. En los dos ejemplos anteriores, nos ha devuelto el valor de la variable 'volumen' y el valor de la variable 'PVP', respectivamente. En ambos casos los hemos mostrado por pantalla, pero también podíamos haberlos utilizado para otras tareas, como por ejemplo realizar otros cálculos a partir de ellos, guardarlos en un fichero, etc....

Variables locales y globales

Las **variables definidas dentro de las funciones**, siempre que utilicemos la palabra clave '**var**' en su declaración, **son denominadas variables locales**, y no pueden ser accedidas desde el exterior de la función. En los ejemplos anteriores, serían variables locales: 'volumen', 'IVA' y 'PVP'.

Esta es una práctica muy recomendable que debemos seguir, para que no entren en conflicto con otras variables del mismo nombre que puedan existir en el resto del script, incluyendo las que pueden estar dentro de otras funciones que también hayamos definido en dicho programa.

Con esta norma, podrían contener valores distintos a pesar de tener el mismo nombre (aunque esto último deberíamos evitarlo para no crear confusiones innecesarias en nuestros scripts) y un cambio en una, no nos afectará a la otra, evitando modificaciones sin darnos cuenta. Por ejemplo, podríamos haber definido la variable 'volumen' en el script del cálculo de la esfera fuera de la función, y a pesar de tener el mismo nombre, no haber afectado a la variable esfera de dentro de esa función.

La contraposición a ésto son las **variables globales**, aquellas que **definimos fuera de las funciones** (o **dentro de ellas sin** la palabra clave **'var'**), que tienen efecto en todo el script. Se dice por tanto que una variable puede tener ámbito (*scope*) local o global.

Beneficios de utilizar funciones

Las funciones permiten crear programas o scripts mejor estructurados y más claros, evitando repeticiones innecesarias y facilitando su mantenimiento. En el ejemplo anterior, las instrucciones para calcular el PVP se escriben solo una vez dentro de la función y no repetidas y repartidas por todo el programa. Si hay una modificación en el IVA por un cambio en la norma fiscal, solo tenemos que modificarlo en la función, y en ese momento, todos los precios se calcularan con el nuevo impuesto.

Las funciones 'empaquetan' y aislan del resto del programa una serie de variables e instrucciones de código que realizan alguna tarea específica. **Solo se ejecutan si son llamadas desde el código principal** y tras procesar sus instrucciones, devuelven un resultado a esa parte del código principal que la invocó. Los valores de esas variables internas (locales) **no entran en conflicto** con el resto del programa.

Como curiosidad, según el lenguaje de programación, a las funciones (o a las estructuras que juegan ese mismo papel) se les puede conocer por otros nombres como (sub)rutinas, procedimientos o subprogramas.

En el entorno de la programación orientada o basada en objetos, vamos a ver que los métodos de dichos objetos son funciones que se han definido dentro de ellos.

Objetos: propiedades y métodos

Objetos

Hemos hablado antes brevemente del **concepto de objeto**, al ver los distintos tipos de variables y datos. Lo definíamos como una especie de modelo con el que representamos una cosa o un concepto de la vida real.

Propiedades de los objetos

Vimos que podíamos definir variables y asociarlas a ese objeto. A esas variables las denominamos **propiedades** (también se les suele llamar **atributos**) y son características propias de ese objeto.

Por ejemplo, en un coche, podían ser la marca, el modelo, potencia del motor, capacidad del maletero, etc. Si lo piensas, seguro que te ocurrirán muchas.

Cuando definimos un objeto en programación, solo elegimos las propiedades que nos interesan para el tipo de proceso o tarea que estamos intentando gestionar o resolver. Por ejemplo, en el contexto de un Taller, puede interesar saber el número de kilómetros que tiene el coche o cuándo ha sido su última reparación o mantenimiento, y por tanto tener definidas propiedades/variables para almacenar esos datos, mientras que esa información no es necesaria o útil si el contexto es otro, como por ejemplo si gestionamos um concesionario de venta de vehículos nuevos o un Rally de coches deportivos.

Métodos de los objetos

Ademas de propiedades, a los objetos podemos definirle una serie de **métodos,** que son instrucciones que pueden cambiar los valores que hemos asignado a esas propiedades. En el caso del Taller podrían ser métodos como 'realizar revisión periódica' o 'arreglar avería', que podrían cambiar las fechas de la última revisión y de la última reparación respectivamente.

Javascript es un lenguaje basado en objetos. De hecho la mayoría de elementos son objetos, o pueden ser tratados como tales. Esto incluye a los tipos de datos 'string' o 'number' o a los 'arrays', a los que podemos aplicar métodos o acciones como explicaremos en los siguientes apartados.

Además, Javascript incluye otra serie de 'objetos *built-in*' (*) como por ejemplo los objetos **Date**, **Math o Regex** (veremos en detalle los dos primeros), o aquellos que tienen que ver con el **documento HTML**, y el **navegador** donde éste se visualiza (**Window**, **Document**, **History**,...), que no abordaremos en este curso al ser una cuestión específica de la programación web (puedes estudiarlos en nuestro curso de Javascript).

Nota (*): los objetos que no definimos nosotros sino que ya vienen incorporados en Javascript se le denominan 'objetos built-in'

Pues bien, como quizás hayas pensado ya, los **métodos** no son más que **funciones**, como las que hemos visto en el apartado anterior. En ellos encapsulamos una serie de instrucciones que podemos luego llamar y aplicar sobre el objeto en el cuál se han definido, de forma parecida a cómo lo hemos hecho con las funciones, eso sí, especificando dicho objeto con la **notación**: `objeto.metodo()`

Vemos unos ejemplos de esa notación, y cuya funcionalidad explicaremos en los apartados siguientes:

(1) `Math.sin(45);`

(2) `numero1.toFixed(2);`

(3) `texto1.search("Digital Learning");`

en estas instrucciones, identificamos los siguientes elementos:

- los **objetos**: (1) **Math**, (2) **numero1** y (3) **texto1**
- los **métodos**: (1) **sin()**, (2) **toFixed()**, (3) **search()**
- los **argumentos** que le pasamos al método: (1) **45**, (2) **2**, (3) **"Digital Learning"**

y realizan respectivamente:

- (1) calcula el seno de 45º
- (2) limita a dos decimales el valor de un número, haciendo el correspondiente redondeo
- (3) busca la cadena "Digital Learning" en un texto

Nota: no te preocupes si no entiendes del todo este subapartado, no te es necesario para seguir avanzando. Como decíamos, solo queremos darte una primera referencia, sin entrar a detallar las explicaciones, porque si no, nos extenderíamos mucho e iríamos más allá de nuestros objetivos de este curso.

Aunque el enfoque a objetos es muy útil, en JavaScript (**basado** en objetos) el tratamiento es un tanto particular respecto a otros lenguajes **orientados** a objetos como Java, C# o Python. Hay muchos conceptos comunes, pero también hay diferencias importantes, como las 'clases' que no existen como tal en JavaScript.

En resumen, si no solo estás interesado en aprender JavaScript sino también los fundamentos de la Programación Orientada a Objetos (POO), vemos más recomendable hacerlo con un lenguaje basado en objetos, como por ejemplo nuestro curso Java para Android y POO.

Creación de objetos

Aunque en este curso no vamos a trabajar con objetos creados por nosotros, creemos que es oportuno que conozcas las formas de crearlos y la interacción básica con ellos, para al menos distinguir esa notación si la ves en un script, por ejemplo al declarar el objeto Date que veremos más adelante.

Hay dos notaciones para crear objetos en Javascript:

Notación literal: vimos esta notación al hablar de los tipos de datos. Utilizaba llaves ({}) para encerrar la declaración de propiedades y métodos del objeto. Vemos un ejemplo:

```
var asignatura = {                          //definimos objeto a
signatura
   nombre: "Historia",                      //definimos sus prop
iedades (notar el uso de ':' y ',')
   docente: "Irene Castillo Moragas",
   horasAnuales: 60,
   horasRestantes: function (horasImpartidas) {     //defini
mos método 'horasRestantes'
      return this.horasAnuales - horasImpartidas;
   }
};
```

Comentamos algunos características de esta notación:

- Vemos que los valores a las **propiedades** se **asignan** con ' **:** ' (**dos puntos**), y no ' = '.
- Vemos que las **propiedades** se **separan** por **comas** (como en los valores de los arrays) y no con punto y coma.
- El **método** que hemos definido en el objeto, lo hemos llamado 'horasRestantes', y como ves es básicamente una **función**, pero **asociado a dicho objeto**.
- La palabra clave '**this**' reemplaza el nombre del objeto, y se utiliza por razones prácticas en las que no vamos a extendernos aquí. Es decir, podríamos haber escrito 'asignatura.horasAnuales'.

Notación constructor: empleamos la expresión `new Objeto()` . En este caso creamos un objeto vacío al que luego podemos asociar propiedades con sus valores correspondientes, y métodos, a través de las notaciones: `objeto.propiedad =` y `objeto.metodo =` .

En el caso de los métodos empleamos lo que se denominan 'funciones anónimas', es decir, que no tienen nombre y no se les puede llamar como a las funciones 'normales', sino que simplemente se ejecutan cuando el intérprete llega a ese punto del programa, como si fueran una expresión.

Vemos un ejemplo:

```
1  var asignatura = new Object();         //creamos el objeto
2  'asignatura'
3
4  //ahora le añadimos las propiedades: nombre, docente, horaA
5  nuales y horas Impartidas
6      asignatura.nombre = "Historía";
7      asignatura.docente = "Irene Castillo Moragas";
8      asignatura.horasAnuales = 60;

9  //definimos un método denominado 'horas restantes' para el
10 objeto 'asignatura', utilizando una función anónima
11     asignatura.horasRestantes = function () {
           return this.horasAnuales - this.horasImpartidas;
       };
```

Para manipular el objeto que hemos definido (cambiar el valor de una propiedad y utilizar su método), puedes probar con este ejemplo. Empleamos la notación literal para definirlo, pero es exactamente igual si hubiéramos empleado la notación constructor:

```
<script>

//definición del objeto con sus propiedades y método
asignatura = {
   nombre: "Historia",
   docente: "Irene Castillo Moragas",
   horasAnuales: 60,
   horasRestantes: function(horasImpartidas) {
          return this.horasAnuales - horasImpartidas;
   }
};

//cambiamos el valor de la propiedad 'docente'
asignatura.docente = "Antonio campos Velilla";
document.write(asignatura.docente + "<br>");

//utilizamos el método horasRestantes llamándole con 'asign
atura.horasRestantes(argumento)
var horas = prompt("Introduzca las horas ya impartidas de l
a asignatura: ");
document.write("Horas restantes de la asignatura: " + asign
atura.horasRestantes(horas));

</script>
```

Saber más: este es un ejemplo muy simple, solo para hacernos una idea de la notación. En este caso podríamos también haber creado un *template* 'Asignatura', es decir una plantilla a partir del cual crearíamos objetos asignatura específicos: 'asignaturaHistoria', 'asignaturaGeografia', etc... Como hemos dicho, no vamos a desarrollar este tema al exceder el objetivo de estos contenidos.

Métodos para tratamiento de números

Como hemos indicado anteriormente, casi todo en Javascript es un objeto o se puede tratar como tal, como es el caso de los tipos de datos 'number' y 'string' (*).

En este apartado vamos a ver el tratamiento de variables númericas como objetos, a los que podemos aplicarles algunos métodos para manipular sus valores. Esta parte, la completaremos con el siguiente apartado, donde veremos como el objeto Math nos dará más posibilidades para realizar cálculos y aplicar funciones matemáticas.

Saber más (*): en Javascript existen **tipos de datos simples** (también llamados '**primitivos**'), como 'number' y 'string' y **tipos de datos complejos** como 'objeto' ('array' se incluiría aquí también). Pues bien, Javascript tiene además los **objetos 'Number' y 'String'** que aunque son elementos distintos a los tipos de datos del mismo nombre, nos permiten tratar a estos últimos como objetos.

Aunque a este nivel de iniciación puede ser algo confuso de entender, basta por ahora con saber que se da esta circunstancia y que podemos utilizarla, como vamos a ver a través de una serie de propiedades y métodos que podemos asociarles.

Tratamiento de números

Si haces una operación con números decimales en Javascript, puede que en el resultado obtengas un gran número de decimales. Javascript dispone de diversos métodos para redondear cifras o acortar decimales. Pueden sernos muy útil para formatear resultados y cantidades varias: precios, porcentajes, etc... En todos los casos, el resultado que nos devuelven estos métodos es un 'string'.

Los vemos:

Método toFixed

Permite fijar el número de decimales que tendrá el número, haciendo un redondeo.

Se aplica de la siguiente forma: `variableNumerica.toFixed(x)` , siendo x el nº de decimales al que queremos redondear (0, 1, 2....).

Método toPrecision

En este caso nos permitirá fijar el número de dígitos (x) que podrá tener el número, haciendo el redondeo correspondiente. Notación: `variableNumerica.toPosition(x)`

Método toExponential

Nos devuelve el número representado en notación exponencial con un número de decimales determinado (x). Notación: `variableNumerica.toExponential(x)`

Ejemplo

Vemos un ejemplo con los diferentes métodos. Comprueba el resultado en el editor online. Luego puedes variar los argumentos que se pasan a los diferentes métodos para ver como se modifican los resultados:

```
<script>
var numeroEjemplo = 32.73412;  //recordar que los decimales
se separan con punto ( . )
var numero0 = numeroEjemplo.toFixed(0);
var numero1 = numeroEjemplo.toFixed(1);
var numero2 = numeroEjemplo.toPrecision(2);
var numero3 = numeroEjemplo.toExponential(3);

document.write("Al aplicar toFixed(0) a " + numeroEjemplo +
" obtenemos: " + numero0 +  "<br>");
document.write("Al aplicar toFixed(1) a " + numeroEjemplo +
" obtenemos: " + numero1 + "<br>");
document.write("El aplicar toPrecision(2) a " + numeroEjemp
lo + " obtenemos: " + numero2 + "<br>");
```

```
    document.write("Al aplicar toExponential(3) a " + numeroEje
    mplo + " obtenemos: " + numero3);

    </script>
```

Conversión de tipos de datos: númericos <–> cadena de texto

Recomendamos en el apartado de operadores numéricos, el asignar a una variable con tipo de datos de texto (string), el tipo de dato numérico (number), según el tratamiento que fuéramos a hacer de ese valor. Bien por ese motivo o porque nos sea simplemente necesario hacer cambio entre esos dos *data type*, conviene saber que disponemos de una serie de funciones generales (no asociados a ningún objeto) y métodos que nos permiten realizar tal conversión. Vamos a ver algunas:

parseInt("*string*")

Convierte las cadenas de números a enteros. Si tiene parte decimal, la elimina sin redondear. Ejemplo:

`var numEntero = parseInt("7.824");` —> el valor de 'numEntero' es 7

Nota: recordar que el 'punto', en vez de la 'coma', es la separación de decimales en notación anglosajona

parseFloat("*string*")

En este caso, sí preserva la parte decimal:

`var numDecimal = parseFloat("7.824");` —> el valor de 'numDecimal' es 7,824

Number("*string*")

En este caso convierte al número idéntico (sea entero o decimal) que contiene la cadena

`var numero1 = Number("33");` –> numero1 tiene el valor 33

`var numero2 = Number("33.1512");` –> numero2 tiene el valor 33,1512

***variableNumerica*.toString()**

Método que nos convierte un número a su cadena de texto equivalente. Ejemplo:

```
1 var numEjemplo = 2017;
2 var numEjComoTexto = numEjemplo.toString();
```

la variable 'numEjComotexto' toma el valor "2017".

Métodos del objeto Math

Javascript tiene un **objeto** llamado "**Math**" que nos permite realizar numerosas **operaciones y funciones matemáticas** a través de los **métodos** que tiene definidos. De una manera coloquial, podríamos decir que tenemos incorporada en Javascript una calculadora científica para utilizar cuando queramos en nuestro código.

Estas operaciones incluyen el **cálculo** de:

- funciones trigonométricas: seno, coseno, tangente,...
- máximo y mínimos de un conjunto de números
- logaritmos, raices y potencias
- redondeo y generación de un número aleatorio

Para utilizar estas funciones, debemos escribir: `Math.metodo()` , siendo **Math** el nombre de este objeto, seguido por un punto y el nombre del método que vamos a

aplicar.

Mostramos un ejemplos con los métodos: **sqrt()**, **cos()**, **pow()**, **max()**, **random()**, **round()** (y las variantes de éste último: **ceil()** y **floor()**) explicando en el mismo script qué realiza cada uno:

```
<script>

var num1 = Math.sqrt(25);              //cálculo de la raiz c
uadrada de un número x: sqrt(x)
var num2 = Math.cos(30);               //cálculo del coseno d
e una angulo x: cos(x)
var num3 = Math.pow(5,2);              //cálculo de un número
x elevado a y: pow(x,y)
var num4 = Math.max(150, 30, -200);    //cálculo del máximo d
e un conjunto de números: max(x,y,z,....)
var num5 = Math.random();              //generación de un núm
ero aleatorio entre 0 y 1
var num6 = Math.round(num5);           //redondeo del valor d
e 'num5' a entero más cercano
var num7 = Math.ceil(num5);            //redondeo del valor d
e 'num5' al entero superior
var num8 = Math.floor(num5);           //redondeo del valor d
e 'num5' al entero inferior

document.write("Resultado al aplicar varios métodos del obj
eto Math : " + "<br><br>");
document.write("Raiz cuadrada de 25 = " + num1 + "<br>" +
"Coseno de 30 = " + num2 + "<br>" + "5 al cuadrado = " + nu
m3 + "<br>" + "Máximo de 150, 30 y -200: " + num4 + "<br>"
+ "N° aleatorio entre 0 y 1: " + num5 + "<br>");
document.write("Redondeo del n° aleatorio anterior: " + num
6 + "<br>" + "Redondeo del mismo n° al entero superior: " +
num7 + "<br>" + "Redondeo del n° al entero anterior: " + nu
m8 + "<br>");

</script>
```

Números aleatorios: hay muchos programas que necesitan **generar un número al azar**, por ejemplo si estamos simulando un sorteo, desarrollando un juego donde el usuario tiene que adivinar algo, si queremos presentar elementos sin un orden establecido,....

Para ello tenemos el método `Math.random()` en JavaScript, que en versiones parecidas está presente en casi todos los lenguaje de programación. Como hemos visto en el ejemplo, este método nos da un número decimal **entre 0 y 1**, así que para **generar números aleatorios enteros** (0, 1, 2, 3...) necesitamos **multiplicar ese resultado por un número entero,** y luego **quedarnos con un redondeo**, aplicando alguno de los métodos del objeto Math que hemos visto en ese ejemplo: **round()**, **floor()** ó **ceil()**.

Para entenderlo mejor, vamos a multiplicar por ejemplo por una potencia de 10 (10, 100, 1000,...) y probarlo directamente:

- **Copia** primero el ejemplo de **script** anterior al editor online y **pulsa repetidamente** el botón '**Ver resultado**', para ver como cambia el nº aleatorio y sus redondeos.
- Después, modifica el ejemplo **cambiando** la instrucción `var num5 = Math.random();` por la instrucción `var num5 = Math.random() * 10;`
- Vuelve a **pulsar repetidamente** el botón '**Ver resultado**'. Verás como en los **redondeos** hechos con los métodos '**floor()**' y '**ceil()**' te van generando números entre **0** y **9** ó **1** y **10**, respectivamente. El método 'round' te genera entre 0 – 10, pero si lo piensas, el '0' y el '10' tienen menos probabilidades de salir que los números del 1 al 9.
- Prueba a multiplicar por otras potencias: Math.random() * 100, Math.random * 1000,... y comprueba nuevamente los resultados. Verás que los rangos serán entre **0-99 / 1-100, 0-999 / 1-1000**, etc...

Constantes

El objeto **Math** también nos proporciona algunas **constantes matemáticas** muy importantes. Son **propiedades** de dicho objeto y podemos acceder a su valor con la notación que ya conocemos: **Math.*propiedad***. Por ejemplo:

`Math.E` : nos da el número e (2,718...)

`Math.PI` : nos da el número π (3,14...)

Saber más: en esta página puedes encontrar el conjunto completo de propiedades y métodos del objeto Math.

Métodos del objeto String

Tratamiento de textos

Ya vimos que ***string*** era un tipo de dato que permitía almacenar cadenas de caracteres alfanuméricos (letras, números, signos de puntuación...). Las utilizamos para almacenar texto o valores que queremos tratar como tal (por ejemplo números que no manipularemos como cantidades en operaciones matemáticas).

Al igual que vimos con las variables de tipo numérico, las variables de texto también pueden tratarse como objetos, utilizando las propiedades y métodos del objeto 'String' que incorpora Javascript. Esto nos permite **obtener información o manipular el valor almacenado** en la variable de muchas formas. Para aplicarlos, deberemos escribirlo con la siguiente estructura: `nombreVariable.metodo()`

Vemos una propiedad y ocho métodos destacados que nos pueden ser de mucha utilidad:

Propiedad 'length'

Nos da la longitud de una variable **string**, indicándonos el **número** de caracteres que tiene. Se llamaría de la siguiente forma: `nombreVariable.length`

```
<script>
var texto1= prompt("Escribe un texto de una o varias palabras:");

var longTexto = texto1.length;   //el método length, nos calcula el nº de caracteres del texto almacenado en 'texto1'

document.write("El texto '" + texto1 + "' tiene: " + longTexto + " caracteres (incluyendo espacios en blanco)");
</script>
```

Nota: en el texto del ejemplo, dentro de document.write hemos utilizado el carácter de escape "" antes de las comillas simples (') para que éstas sean tratadas como texto normal, y no como caracteres especiales de Javascript.

Método toUpperCase()

Cambia los caracteres almacenados en la variable a mayúsculas. Se escribe: `nombreVariable.toUpperCase()`

Prueba las siguientes líneas de código para ver un ejemplo, donde cambiamos la cadena de texto almacenada en una variable a mayúsculas:

```
<script>
texto2= prompt("Escribe un texto que contengan minúsculas : ");
texto2Mayusculas = texto2.toUpperCase();              //cambiamos a mayúsculas la cadena de texto almacenada en la variable texto2
document.write(texto2Mayusculas);
</script>
```

Método toLowerCase()

Hace justo lo contrario al método anterior, pasando los caracteres a minúscula. Compruébalo directamente en el anterior ejemplo cambiando en el script un método por otro e introduciendo un texto que contenga mayúsculas:

```
texto2.toLowercase();
```

Métodos indexOf() y lastIndexOf()

Búsqueda de una cadena de texto (por ejemplo: un carácter, una palabra o una expresión) dentro del valor almacenado en la variable de texto. Si lo encuentra, nos da la posición del primer carácter de esa cadena en su primera aparición (indexOf) o en su última (lastIndexOf) dentro del texto.

Lo aplicamos de la siguiente forma: `nombreVariable.indexOf("cadena a buscar")`

```
<script>
var texto2= "Ejemplo de búsqueda con un método Javascript en estos contenidos de introducción a Javascript de Digital Learning ";
var posicionPrimera = texto2.indexOf("Javascript");
var posicionUltima = texto2.lastIndexOf("Javascript");

document.write("En el texto: " + "<br><em>" + texto2 + "</em><br>" + "la primera aparición de la palabra Javascript, comienza en la posición: " + posicionPrimera + "<br>");
document.write("La última aparición de la palabra Javascript, comienza en la posición: " + posicionUltima);
</script>
```

Método search()

Similar al anterior, pero más potente, ya que nos permite buscar utilizando **expresiones regulares**, es decir localizar coincidencias usando un **patrón de texto**.

Las expresiones regulares es un método muy utilizado en el tratamiento de textos, y se incluye en muchos lenguajes de programación. Describirlo requeriría casi un curso específico, que intentaremos desarrollar en un futuro próximo, pero para hacernos una primera idea, aquí te dejamos dos ejemplos:

- Encontrar si hay una cifra en el texto del ejemplo anterior: `texto2.search(/[0-9]/)`
- Encontrar si el texto contiene una 'x', 'y' o una 'z': `texto2.search(/[x,y,z]/)`

Como ves, empleamos el símbolo de barra inclinada (/) para delimitar el patrón que vamos a buscar.
En el primer ejemplo, la expresión '0-9' entre corchetes ([ ]) se traduce por: cualquier número comprendido en el rango de 0 a 9, ambos incluidos, es decir, cualquier número. Si hubiésemos puesto /[a-m]/ sería: cualquier letra minúscula comprendida en el rango de la 'a' a la 'm'.
En el segundo ejemplo, incluimos una lista de letras que queremos que se tengan en cuenta en la búsqueda, separadas por comas (,).

Si quieres conocer más sobre las las expresiones regulares, puedes empezar por este artículo de Wikipedia

Método charAt()

Este método hace en cierta manera lo contrario a indexOf(): indicamos una posición y nos devuelve el carácter que se encuentra allí.

Ejemplo:

`texto2.charAt(34)` : nos devuelve el caracter 'J'

Método substring()

Este método es similar al anterior, pero en este caso, nos devuelve los caracteres que se encuentran en un rango de posiciones.

Ejemplos:

`texto2.substring(34,45)` : nos devuelve el conjunto de caracteres entre la posición 34 y la 44 (la 45 no se tiene en cuenta), es decir: 'Javascript'

Método replace()

Este método es una especie de 'buscar y cambiar'. Nos permite reemplazar la cadena de caracteres (uno o varios) que definimos en el primer argumento por la que definimos en el segundo argumento. Solo cambia la primera coincidencia.

Ejemplo:

`texto2.replace("Javascript","JS")` : reemplaza la palabra Javascript por JS (solo la 1ª que aparece en el texto, no la 2ª)

Métodos del objeto Array

Ya vimos que los arrays nos permitían almacenar listas con un número variable de valores.

También vimos que podíamos cambiar o añadir valores, indicando la posición en la lista (el índice, que recordemos, empieza en '0') donde haremos el cambio o añadiremos el nuevo valor, como en el siguiente ejemplo:

```
1 var ciudades = ["Cuenca", "Gijón", "Granada"];
2 ciudades[0] = "Salamanca"                //cambiamos el valor
3 Cuenca por Salamanca
  ciudades[4] = "Toledo"                  // añadimos esta ciud
  ad al final de la lista
```

Pues bien, un array en Javascript es un tipo de **objeto** que tiene asociados diversos **métodos** que nos permiten **manipular sus valores almacenados** de muchas más formas.

Como es habitual, para aplicarlos se escriben con la siguiente estructura:
`nombreArray.metodo()`

Vamos a describir algunos de estos métodos o funciones:

Método pop()

Este método elimina el último elemento del array.

Se escribiría ***nombreArray*.pop()**.

Si aplicamos esta función sobre el array 'ciudades' que creamos antes:

`ciudades.pop()` : eliminaría su último valor, es decir 'Granada'

Método push()

Este método añadiría nuevos valores al final del array.

Se escribiría: ***nombreArray*.push(*valor1, valor2, ...*)**

Como ves, podemos añadir uno o varios valores al array con este método. Siguiendo con el ejemplo del array anterior:

```
ciudades.push("Lugo", "Castellón", "Oviedo")
```

Métodos shift() y unshift()

Si queremos eliminar o añadir valores al principio del array, en vez de al final como hemos visto con pop() y push(), tendríamos las funciones **shift()** y **unshift()** respectivamente. Ejemplos:

`ciudades.shift()` : elimina el primer valor del array. Cuenca

`ciudades.unshift("León", "Ciudad Real")` : añade al principio del array

estas dos ciudades

Método splice()

No solo estamos limitados al principio o final del array, la función splice() nos permite insertar y/o eliminar elementos en cualquier parte del array. Lo vemos mejor con dos ejemplos:

`ciudades.splice(2,3)` : borraría a partir del valor ciudades[2], los tres siguientes valores.

`ciudades.splice(1,0, "Cádiz", "Vitoria", "Tarragona")` : añadiría a partir del valor ciudades[1], esas tres nuevas ciudades. Al tener el 2º parámetro el valor '0', no borraría ninguno)

Método indexOf()

Este método Nos da la posición o índice que ocupa en el array un valor dado, como un texto o un número. Busca solo la primera coincidencia en caso de que esté repetido.

```
ciudades.indexOf("Vitoria")
```

Propiedad length

Esta propiedad que ya vimos con string nos da la longitud del array. Ejemplo:
`ciudades.length`

Nota: recordar que las propiedades no lleva paréntesis en la notación, como ocurre con los métodos.

Adelantándonos a un apartado posterior, los **bucles**, veremos que con estas instrucciones podemos recorrer fácilmente todos los valores de un array utilizando esa instrucción y la propiedad length. Aunque explicaremos más adelante los bucles, puedes ir haciéndote una idea de sus posibilidades, comprobando el resultado del siguiente ejemplo:

```
<script>
var granPoblacion = ["Madrid", "Barcelona", "Valencia", "Sev
illa"];
document.write("La longitud el array es: " + granPoblacion.l
ength + "<br>");
for (var i = 0;  i < granPoblacion.length; i++) {
document.write("El valor " + i + " del array es: " + granPob
lacion[i] + "<br>");
}
</script>
```

Saber más: para los que quieran ampliar esta información, aquí pueden consultar una lista completa de los métodos del objeto array

Métodos de objeto Date

Podemos trabajar con horas y fechas en Javascript utilizando el objeto Date. En este caso, a diferencia de los objetos anteriores que hemos visto, debemos crearlo primero mediante la instrucción: `var hoy = new Date();`

En ella, hemos **asignado a una variable** que hemos llamado 'hoy', **el objeto Date**, que hemos creado con la expresión **new Date()**.

Saber más: recordar que la expresión new *Objeto*() la vimos al presentar la notación constructor. En este caso, como el objeto Date ya existe (se dice que creamos una instancia del mismo), ya tiene propiedades y métodos definidos, aunque podemos modificar lo valores de las propiedades que tienen por defecto, como veremos en este apartado.

Vemos los métodos que nos permiten extraer información o establecerle nuevos valores al objeto date, con un ejemplo que puedes ejecutar en el editor online para ver el resultado.

> **Nota**: la fecha y hora que proporciona el objeto Date las toma del ordenador que está visualizando la página. Si el visitante está en otra zona horaria a la nuestra, o el reloj de su computador está mal configurado, puede haber diferencias en lo que se le muestra.

Métodos getFullYear()

Nos devuelve el año con 4 dígitos.

Para aplicarlo, si seguimos con el ejemplo del inicio donde hemos asignado el valor del objeto Date a la variable 'hoy':

`var hoy = new Date()` , sería:

`hoy.getFullYear()` .

En el resto de métodos la notación sería similar

Método getMonth()

Nos devuelve el número de mes del 0 (enero) al 11 (diciembre)

Método getDate()

Nos devuelve el día del mes, del 1 al 31

Método getDay()

Nos devuelve el día de la semana del 0 (domingo) al 6 (sábado)

Método getHours()

Nos devuelve la hora actual, desde 0 a 23 (formato 24 horas)

Método getMinutes()

Nos devuelve los minutos actuales, desde 0 a 59

Métodos getSeconds() / getMilliseconds() / getTime()

Nos devuelve respectivamente los segundos (0-59), los milisegundos (0-999) y los milisegundos transcurridos desde el 01/01/1970 (es el denominado 'tiempo Unix')

Métodos toString() / toDateString() / toTimeString()

Nos devuelven respectivamente la fecha completa (fecha y hora), la fecha y la hora como un string (texto), lo que nos permite manipularlos por ejemplo con los métodos que vimos para las variables de texto.

Métodos para establecer otros valores al objeto Date

Además de poder recuperar la fecha y hora del objeto Date, podemos establecerle otros valores distinto a los que proporciona el reloj del ordenador. Para ello tenemos dos formas, bien creando un objeto con esos nuevos valores como argumentos:

```
1 new Date(año, mes, dia, hora, minutos, segundos)
```

o aplicando los métodos **setFullYear()**, **setMonth()**, **setDate()**, **setHours()**, **setMinutes()**, **setSeconds()**, **setMilliseconds**, **setTime()**, según el valor que queramos modificar.

Ejemplo con métodos Date

ejecuta este ejemplo en el editor online y podrás comprobar el resultado de diferentes métodos del objeto Date:

```
<script>
var hoy = new Date();
document.write("resultado Date = " + hoy + "<br>");
document.write("resultado getFullYear = " + hoy.getFullYear
() + "<br>");
document.write("resultado getMonth = " + hoy.getMonth() + "
<br>");
document.write("resultado getDate = " + hoy.getDate() + "<b
r>");
document.write("resultado getDay = " + hoy.getDay() + "<br
>");
document.write("resultado getHour = " + hoy.getHours() + "<
br>");
document.write("resultado getMinutes = " + hoy.getMinutes()
+ "<br>");
document.write("resultado toString = " + hoy.toString() + "
<br>");
</script>
```

Módulo 6. Control del flujo del programa: condicionales y bucles

Autor: Manuel Cabello

Versión 1.0

Fecha: Abril 2016

Digital Learning

Orden de instrucciones

Una de las características fundamentales de los lenguajes de programación es la de ofrecernos la posibilidad de controlar el flujo del programa, es decir, en qué orden y según que condiciones se van a ejecutar las instrucciones del mismo.

En Javascript. como en la mayoría de los otros lenguajes, hay dos tipos de instrucciones que nos permiten controlar este orden o avance del programa:

- **Instrucciones condicionales**: nos permiten determinar qué instrucciones se ejecutarán en función de que se cumpla una o varias condiciones determinadas. En este grupo incluiríamos a las instrucciones "**if/else**" y "**switch/case**"
- **Instrucciones de bucle**: con ellas podemos repetir la ejecución de una o varias instrucciones un determinado número de veces, mientras se cumpla una condición. En este grupo incluiríamos las instrucciones "**for**" y "**do/while**"

Vamos a describir en las siguientes páginas de este apartado cada una de estas instrucciones, con ejemplos que nos ayudarán a entender mejor estos elementos tan importantes en la programación.

Saber más: los nombres de estas instrucciones, e incluso su sintaxis, es muy similar en la mayoría de los lenguajes más utilizados. Las palabras claves, if, else, for, do, while, switch, case, las vas a encontrar exactamente igual en casi todos ellos, por lo que una vez que comprendes su funcionamiento en un lenguaje, como Javascript en este caso, es muy fácil aplicarlo en cualquier otro (Java, C#, PHP, Python...)

Condicional if

La palabra "**if**" es el '**si**' **condicional en inglés**. La utilizamos en frases como: "saldría a pasear **si** no lloviera". Es decir, si se cumpliera la condición "no llover", entonces realizaríamos la acción "salir a pasear". En Javascript (y en general en los lenguajes de programación) esta instrucción "if" juega el mismo papel.

Cuando elaboramos un algoritmo, es decir, desglosamos en pasos detallados y ordenados una tarea o proceso, a menudo tenemos la necesidad de introducir condiciones para contemplar todas las posibilidades que pueden darse en dicho proceso. Lo más directo es verlo con un ejemplo.

Supongamos que desarrollamos un script para mostrar el precio de una entrada a un concierto que se vende online a través de nuestra Web. La entrada cuesta 50 €, pero si el comprador tiene un código de descuento ("Spring7864") y lo introduce en el formulario de compra, se le aplicará un descuento de 15 €. El algoritmo en ese punto sería:

- Procesamos los datos del formulario
  - ¿Ha introducido el código de descuento "Spring7864" –> se aplicará un descuento de 15€
- Calcular el precio final: Precio – descuento

que en código Javascript podría ser:

```
<script>
var PVP= 50;
var descuento = 0;
var precioFinal;
var codDescuento = prompt("Si tienes un código de descuento, escríbelo aquí: ");
//Ahora incluímos un condicional, que se ejecutará solo si se cumple la condición que hemos puesto
if (codDescuento == "Spring7864") {
    descuento = 15;
    }
precioFinal = PVP - descuento;  //si no se cumplió (codDescuento == "Spring7864"), el valor de descuento será '0'
document.write("Precio= " + precioFinal + " €");
</script>
```

Para simplificar, el formulario se ha reducido a solicitar el código descuento a través de una sentencia 'prompt'.

> **Nota**: prueba este código en el editor online introduciendo diferentes valores. Verás que cualquier pequeña diferencia a la hora de escribir el código (teclear 's' en vez de 'S' o por el contrario, todo en mayúsculas), hará que no se aplique el código de descuento. Hay varias formas de evitar ésto si no queremos ser tan estrictos. Consulta el cuadro al final de la página donde te damos pistas y la solución para hacerlo de dos maneras distintas.

Como ves en el ejemplo, tras el **if**, comprobamos la condición dentro de un paréntesis y si se cumple, se ejecuta la instrucción 'descuento = 15'. El interprete de Javascript traduce esta sentencia por:

"si el valor de codDescuento es igual (==) a 'Spring7864', haz los pasos que están entre las dos llaves {.....}, es decir, asigna el valor '15' a la variable descuento y continua luego con el resto de instrucciones del programa (en este caso, el cálculo de precioFinal).
Si no cumple la condición, no tengas en cuenta las instrucciones que hay entre las llaves {...} y sigue con el resto de instrucciones del programa (cálculo del precioFinal)."

Sintaxis de if

De forma general, si incluimos más de una instrucción a ejecutar, la sentencia if tiene la siguiente sintaxis:

```
1 if (condición) {
2     instruccion_1;
3     instrucción_2;
4     ..................
5     instrucción_n;
6 }
```

En la que observamos que:

- La condición se encierra entre paréntesis (), separada por un espacio en blanco de la palabra 'if' y sin finalizar con un ' ; '
- El bloque de instrucciones, que se ejecuta solo si se cumple la condición, lo incluimos entre dos llaves de apertura y cierre ' **{** ' y ' **}** ' para indicar el inicio y final del bloque. Las llaves en sí no llevarán el ' ; ' de final de sentencia, pero sí cada instrucción que esté allí contenida.

No hay un estilo único para escribir esta sentencia. Nosotros hemos utilizado un formato muy habitual para que el código quede más claro, sobre todo si la condición es extensa y/o hay varias instrucciones. Hemos dado algo de sangrado (espacios) al bloque de instrucciones, escribiendo cada una en una nueva linea y colocando la llave final en una línea aparte. No es obligatorio y habría funcionado igual escribiendo todo en una sola linea:

```
if (codDescuento == "Spring7864") {descuento = 15);}
```

Ejercicio

Comprobar validez del código descuento: ¿Has intentado comprobar el código de descuento de forma que aunque el usuario escriba minúsculas o mayúsculas en "**Spring**7864" siga cumpliéndose la condición de igualdad?

Si no lo resuelves en la página de la Web te damos dos pistas, y si tampoco lo consigues con ellas, puedes ver la solución allí.

Condicional if: opciones else, else if

Es muy habitual que nos interese plantear una alternativa si no se cumple la condición que hemos planteando en la sentencia if. Si es verdadera esa condición, se ejecutan una o varias instrucciones determinadas, pero si no, se ejecutan otras diferentes. de forma que el programa puede bifurcarse por dos 'caminos' distintos. Veámoslo con un ejemplo:

Imagínate que el anterior código es un código promocional que además de servir como descuento, da derecho exclusivo a comprar por la web. El que no lo tiene, no puede hacerlo y debe adquirir la entrada en taquilla. En este caso es más útil usar la sentencia 'if/else' de la siguiente forma:

```
<script>
var PVP= 50;
var descuento = 0;
var precioFinal;
var codPromocion = prompt("Si tienes un código para comprar en la Web con descuento especial, escríbelo aquí: ");

if (codPromocion == "Spring7864") {
    descuento = 15;
    precioFinal = PVP - descuento;
    document.write("Precio= " + precioFinal + " €");
}
else {
    document.write("Lo sentimos, pero debes adquirir tu entrada en taquilla. Precio: " + PVP + " €");
}

</script>
```

Ahora bien, las alternativas pueden ser una sola o varias, es decir, que nos puede

interesar desdoblar ese camino o flujo del programa por varias rutas distintas.

Imaginemos que hemos creado dos códigos: uno es el de la anterior promoción que da derecho a comprar por la Web con descuento de 15€, y otro (10Web) que ofrece un descuento de 10 € por la Web. El que no tiene ninguno de los dos tendrá que adquirir la entrada en taquilla. Para ello utilizaremos la sentencia '**if /else if /else**' de la siguiente forma:

```
<script>
var PVP= 50;
var descuento = 0;
var precioFinal;
var codPromocion = prompt("Si tienes un código para comprar en la Web con descuento especial, escríbelo aquí: ");

if (codPromocion == "Spring7864") {
    descuento = 15;
    precioFinal = PVP - descuento;
    document.write("Precio= " + precioFinal + " €");
}
else if (codPromocion == "10Web") {
    descuento = 10;
    precioFinal = PVP - descuento;
    document.write("Precio= " + precioFinal + " €");
}
else {
    document.write("Lo sentimos, pero debes adquirir tu entrada en taquilla. Precio: " + PVP + " €");
}

</script>
```

Podríamos generar más códigos promocionales con ofertas diferentes. Para ello, podríamos incluir más '**else if**' que contemplaran esas nuevas condiciones/instrucciones , o bien emplear la sentencia '**switch/case**' que vamos a describir en el siguiente apartado.

Múltiples opciones. instrucción switch...case

Podemos tener la situación en que el flujo del programa puede depender de diversas opciones para continuar por un camino u otro. Vimos en el apartado anterior que para resolver ésto, podíamos emplear varios '**else if**', tantos como opciones tengamos, pero hay una instrucción algo más concisa para conseguir el mismo resultado, según esta notación general:

```
switch(nomVariable) {
case "opcion1" :
    instrucciones;
   break;
case "opcion2" :
    instrucciones;
    break;
.....................
case "opcionN" :
    instrucciones;
    break;
default :
    instrucciones;
```

Analizamos un poco más en detalle la sintaxis de este código:

- Al lado de la palabra clave switch, pondremos entre paréntesis el nombre de la variable cuyo valor queremos comprobar. Según sea dicho valor, se debe de ejecutar una serie de instrucciones.

- Cada valor contra el que vamos a comparar se incluye tras la palabra clave ‘case’, separada por dos puntos (:) y el valor encerrado en comillas
  Por ejemplo, si nuestro programa solicitara un número del 1 al 10, y según eligiera el usuario, el programa le va a dar un resultado concreto, escribiríamos un ‘case’ para cada nº: 1, 2, ….10.
- Incluimos un ‘break’ en cada cláusula ‘case’ para que el programa no se pare ahí, sino que salga de ese punto (*break* es romper en inglés) y continúe después del final de la instrucción switch con las instrucciones siguientes que contenga el programa.
- La última clausula ‘default’ realiza el mismo comentido que el último ‘else’ que ponemos tras una serie de ‘else if’. Es decir, si no se ha cumplido ninguna de las condiciones y queremso que se ejecute algo concreto, lo ponemos en esa clausula ‘default’

Veamos cómo debemos escribirla. Imaginemos que en el ejemplo que habíamos introducido antes, hay 5 códigos promocionales distintos: ‘PlusOro’, ‘Platino33’, ‘100Max’, ‘SuperSilver’ y ‘TotalBronce’ que descuentan respectivamente 25, 20, 15, 10 y 5 € . No habría problema en escribir varios ‘else if’, pero tenemos también esta opción del código ejemplo siguiente:

```
<script>
var PVP= 50;
var descuento = 0;
var precioFinal;
var codPromocion = prompt("Si tienes un código para comprar en la Web con descuento especial, escríbelo aquí: ");
//pasamos a mayúsculas para evitar diferencias al comparar por haber usado minúsculas:
codPromocion = codPromocion.toUpperCase();
switch(codPromocion) {
case "PLUSORO":
  descuento = 25;
  break;
case "PLATINO33" :
  descuento = 20;
  break;
case "100MAX" :
  descuento = 15;
  break;
case "SUPERSILVER" :
  descuento = 10;
  break;
case "TOTALBRONCE" :
  descuento = 5;
  break;
default:
  alert("No ha introducido código promocional, o es erróneo");
}
precioFinal = PVP - descuento;
document.write("Precio= " + precioFinal + " €");
</script>
```

Si ves en el ejemplo, no hemos hecho los cálculos del precio final en cada cláusula 'case' para hacer el código más simple. Solo modificamos en cada 'case' el valor del descuento, que de partida le habíamos asignado el valor '0', y al finalizar la instrucción 'switch', calculamos el precio final. Si el usuario no ha introducido ningún código válido, el descuento seguiría siendo '0' y el precio final '50€'.

Bucle for

Con frecuencia ideamos algoritmos o programas en los que necesitamos **que una serie de acciones se repitan varias veces** hasta que una condición se cumpla (o deje de cumplirse) para que pueda completarse. Estamos hablando del uso de **bucles** (o 'loops') en programación.

¿Qué queremos decir con eso? Utilizando un **símil** sencillo, pensemos en el proceso de hinchar el neumático de un coche con una compresor en una gasolinera. Los pasos serían básicamente darle aire y comprobar por la aguja del compresor en cuánto queda la presión de la rueda; si sigue baja, volver a darle aire y volver a comprobar y así sucesivamente hasta llegar a la presión adecuada (simplificamos este ejemplo, pensando que no sobrepasamos nunca esa presión). Básicamente estamos repitiendo varias veces un paso, 'dar aire', hasta que no cumpla una condición, 'alcanzar la presión correcta'.

Veamos ahora como funciona en **Javascript** la sentencia que nos permite repetir una o varias instrucciones mientras se cumpla una condición: el **bucle 'for'**

Comenzamos con un ejemplo muy básico, sin mucha utilidad, para ver su sintaxis y cómo funciona. Vamos a escribir una misma frase cinco veces en pantalla:

```
<script>
for (var i = 0; i < 5; i++) {
     document.write("Esto es un bucle for (i = " + i + ")" +
"<br>");
}
document.write("El bucle ha finalizado");
</script>
```

Como puedes ver, la estructura tiene ciertas similitudes a la sentencia 'if':

- Después de la palabra clave 'for', dejamos espacio en blanco y entre paréntesis encerramos 3 instrucciones:
  - En la 1ª, (**var i = 0;**) declaramos una variable 'i' que le damos el valor cero

(podríamos haberla llamado de cualquier otra forma)

- Luego, ponemos una condición (**i < 5;**), es decir que este valor sea menor que cinco
- Por último, incrementamos el valor de i en '1' (**i++**)

- Como ocurría con 'if', si esa condición es válida se ejecuta lo que hay entre las llaves '{'...'}' . En este caso es la instrucción document.write (linea 3) que imprime una frase y el valor de 'i'.
- Al finalizar estas instrucciones se vuelve a comprobar la condición primera, y como en este caso 'i' vale 1, sigue cumpliéndose que es menor que 5 y las ejecuta otra vez. En el siguiente paso 'i' vale 2, y se vuelve a ejecutar, así hasta que 'i' vale 5, finalizándose el bucle. La siguiente instrucción tras salir del bucle (linea 5) nos avisa con un mensaje.

La variable 'i' actúa como contador, ya que se incrementa en una unidad en cada ejecución del bucle, hasta alcanzar el valor límite impuesto en la condición.

Ejercicio

Prueba a hacer el sistema del contador al revés, es decir, que comience en 5 y el bucle se pare al llegar este contador a 0. Pincha aquí para ver la solución

Sintaxis de 'for'

De forma general, si incluimos más de una instrucción a ejecutar, la sentencia for tiene la siguiente sintaxis:

```
1 for (var i = 0; condición a cumplir; i++ ) {
2     instruccion_1;
3     instrucción_2;
4     ..................
```

```
5      instrucción_n;
6  }
```

[/vc_column_text]

Ejemplo con bucle 'for' y un condicional 'if'

Veamos ahora otro ejemplo, algo más elaborado que el anterior.

Imaginemos que vamos a escribir un script para pasar un test online. En ese test, el usuario debe acertar una pregunta antes de agotar 3 intentos. El script consistiría básicamente en:

- Definimos un contador de intentos a través de una variable y le damos un valor de partida '0' (var contador = 0;)
- Comparamos el valor del contador con un valor límite de intentos, en este caso '3'
  - Si el valor del contador está por debajo del límite:
    - Mostramos una pregunta a través de un una instrucción 'prompt', para que él pueda introducir ahí su respuesta.
      - ¿Es la respuesta correcta? (utilizamos un 'if' comparando su contestación con la respuesta correcta)
        - Sí –> se muestra un **mensaje de acierto y se acaba el test**
        - No –> Se incrementa el contador en '1' y se vuelve al segundo paso, para compararlo de nuevo con el valor límite
  - Si el valor del contador no es menor que el límite:
    - Mostramos un **mensaje de fallo y se acaba el test**

Esto, traducido a Javascript utilizando la sentencia for, podría escribirse:

```
<script>
var respuesta = "HELSINKI";        //esta es la respuesta co
rrecta
var acierto = false;               //este es un 'marcador' q
ue cambiaremos si acierta la pregunta al valor 'true'
for (var contador = 0; contador < 3 ; contador++) {
     var respUsuario = prompt("Capital de Finlandia: ");
     if (respUsuario.toUpperCase() === respuesta) {   //pas
amos la respuesta a mayúsculas para comparar con la respues
ta correcta
          var acierto = true;
          document.write("Enhorabuena, has acertado");
          break;
     }
}
if (!acierto) {document.write("Lo sentimos, has consumido t
us 3 intentos")};  //(ver nota abajo)
</script>
```

Notas:

1) Hemos utilizado la palabra clave '**break**' que permite **salir de un bucle 'for'** antes de que se completen todos sus ciclos. Lo hemos utilizado porque si se está evaluando las instrucciones dentro del 'if', es que el usuario ha acertado ya la pregunta y no tiene sentido seguir preguntándole.

2) En la última sentencia podríamos haber escrito la condición como: `(acierto === false)`, sin embargo, hemos empleado el operador "**negación lógica**" (!) antes del nombre de la variable: `(!acierto)`.
Es decir, si **acierto** tiene el valor booleano '**true**' (verdad), entonces **!acierto** tiene el valor booleano '**false**' (falso) y viceversa. Esta última es la forma más habitual de hacerlo.

Ejercicio

Intenta modificar el anterior ejemplo para que no sea necesario utilizar un 'break', es decir, que aunque no lo incluyamos, el bucle acabará igualmente si el usuario escribe la respuesta correcta.

Pincha aquí para ver la solución en la Web

Bucles while y do...while

Bucle while

El bucle **while** es muy similar al bucle **for**. While es la palabra inglesa para 'mientras' y cuando la incluimos en nuestro código la podríamos traducir al castellano por: "**mientras se cumpla una condición, ejecuta las siguientes instrucciones....**".

Vemos su sintaxis con un sencillo ejemplo:

```
1 <script>
2 var i=0;
3 while (i <= 5) {
4     document.write("Valor de i = " + i + "<br>");
5     i++;
6 }
7 </script>
```

Básicamente consiste en:

- Declarar una variable 'i' que nos servirá de contador, asignándole un valor '0' (*es lo*

habitual, como vimos en 'for', pero podemos poner cualquier nombre a la variable y asignarle cualquier valor)

- Iniciar el bucle que se ejecutará mientras se cumpla la condición que hemos indicado en este caso (i <= 5)
- Ejecutar las instrucciones encerradas entre llaves ({.....}); en este caso imprimir el valor de i y aumentar el contador 'i' en una unidad

Bucle do...while

Hay una variación que podemos utilizar, el bucle ***do ... while***, muy similar al anterior pero con una diferencia de comportamiento y no solo de escritura. Vemos el ejemplo anterior con este tipo de bucle:

```
1 <script>
2 var i=0;
3 do {
4     document.write("Valor de i = " + i + "<br>");
5     i++;
6 } while (i <= 5);
7 </script>
```

Como podemos observar, declaramos igualmente el contador, pero en vez de indicar la condición que ha de cumplirse a continuación, incluimos la palabra clave 'do' (hacer), que ejecuta las instrucciones que incluimos entre las llaves y ya después de ese bloque, incluimos la condición expresada con un while.

¿Qué diferencia representa éste bucle? Pues que las **instrucciones** incluidas en el bloque '**do**' siempre **se ejecuta al menos una vez**, independientemente de la condición final que pongamos, ya que ésta se comprueba después, y no antes de dichas instrucciones.

Comprueba el resultado de ambos ejemplos en el editor online y verás que son iguales. Haz luego una prueba cambiando la condición "**i <= 5**" y poniendo por ejemplo: "**i < 0**" (condición que no se cumple nunca porque el valor de "i" es siempre igual o mayor que cero) y comprueba de nuevo el resultado para ver la diferencia:

Ejercicio

Para ver la similitud de los bucle **for** y **while**, prueba a trasladar el script ejemplo que vimos en el apartado anterior (un sencillo test online), y en vez de utilizar un bulce for, **emplea en esta ocasión un bucle while**.
Te volvemos a copiar el ejemplo aquí para más comodidad:

```
<script>
var respuesta = "HELSINKI";
var acierto = false;
for (var contador = 0; contador < 3 && acierto === false; c
ontador++) {
     var respUsuario = prompt("Capital de Finlandia: ");
     if (respUsuario.toUpperCase() === respuesta) {
         var acierto = true;
         document.write("Enhorabuena, has acertado");
     }
}
if (!acierto) {document.write("Lo sentimos, has consumido t
us 3 intentos")};
</script>
```

Pincha aquí para ver la solución en la Web

Encadenamiento de condicionales y bucles

Ya hemos visto en un ejemplo (test online) de los apartados anteriores, cómo podíamos **incluir** un **condicional if** dentro de un **bucle for** o un **bucle while** . De forma parecida, también podemos hacer encadenamientos, es decir, incluir condicionales dentro de otros condicionales, o bucles dentro de otros bucles, creando distintos niveles dentro de esos bloques.

Encadenamiento de if

Cuando queramos que se cumplan condiciones complejas en un condicional, en vez de expresarlo en una sola expresión, puede sernos más sencillo o claro emplear un encadenamiento de condiciones if. Lo vemos con un ejemplo:

Condicional sin encadenamiento:

```
<script>
var a = 1; var b = 2; var c = 3; var d = 4; var e;
if ((b-a === 1 || d-c === 1) && c !== b ) {
 e = c+d;
}
else {
 e = d-c;
}
document.write("e = " + e);
</script>
```

Podríamos escribir el script solo con instrucciones que contuvieran condiciones simples (solo se ha de cumplir una condición cada vez), encadenando o incluyendo uno dentro de otros:

```
<script>
var a = 1; var b = 2; var c = 3; var d = 4; var e;
if (c !== b) {
    if (b-a === 1) {
        e = c+d;
    }
    else if (d-c === 1){
        e = c+d;
    }
    else {
        e = d-c;
   }
}
else {
 e = (d-c);
}
document.write("e = " + e);
</script>
```

Aunque a nivel de condiciones, quizás sea más fácil de entender la 2ª versión, nosotros preferiríamos utilizar la 1ª versión por ser más compacta, lo que al final y viendo el script en conjunto, creemos que lo hace más claro. No obstante, puede haber casos realmente complejos donde nos sea más fácil descomponer una condición en varios if. Lo importante, en cualquier caso, es que sepamos que podemos hacerlo de esta forma.

Encadenamiento de bucles

El incluir un bucle dentro de otro puede ser algo más común porque hay procesos que requieren que por cada valor del contador del bucle externo se recorra un rango de valores de un bucle interno. Suena quizás algo enrevesado pero con un ejemplo se

puede visualizar mucho mejor:

```
<script>
var diaSemana = ["Lunes", "Martes", "Miércoles", "Jueves",
"Viernes", "Sábado", "Domingo"];

for (var dia = 0; dia < diaSemana.length; dia++)
{              //bucle externo
     document.write(diaSemana[dia] + "<br>")

     for (var hora = 0; hora < 24; hora++)
{                    //bucle interno
           if (hora !== 23) {document.write(hora + "h. ")}
           else {document.write(hora + "h." + "<br>")}
     }
 }
</script>
```

Si ejecutas el ejemplo en el editor online verás como para cada día de las semana que hemos definido en un array (diaSemana[ ]), mostramos en pantalla las 24 horas de ese día. El bucle externo recorre los días de las semana desde lunes hasta domingo, y por cada uno de estos días, el bucle interno recorre las horas desde 0 a 23. Vamos a describirlo paso a paso:

- El bucle exterior toma en primer lugar el diaSemana[0], es decir, el primer valor del array: “Lunes”.
- Se ejecuta la primera instrucción de ese bucle externo, que es un document.write, que muestra en pantalla dicho día y realiza un salto de linea con la etiqueta “
”
- A continuación se ejecuta el bucle interno que comienza a pintar en pantalla las horas del día.
- Hemos incluido un condicional para formatear mejor el resultado. Comprueba si la hora que va a mostrar en pantalla no es la última del día, es decir, 23: (hora !== 23). Si no lo es, muestra el nº de hora seguido por el texto “h.”, y si lo es, hace lo mismo

pero forzando un salto de línea con “
” para que empiece a mostrarse el siguiente día en otra línea.

- El bucle interno termina cada vez que se alcanza el limite de 23. En ese momento el exterior retoma el flujo del programa, seleccionando el dia siguiente dentro del array, y volviendo a realizarse todo el ciclo.

Ejercicio

Prueba a hacer algo similar, pero en esta ocasión muestra en pantalla los meses del año y en cada uno los días que tiene: 1, 2..... En una versión más simplificada puedes poner 30 días a todos los meses, y en otra, trata de poner los días que tienen exactamente cada uno: 28 (Febrero), 30 ó 31.
