Capítulo 1

JavaScript

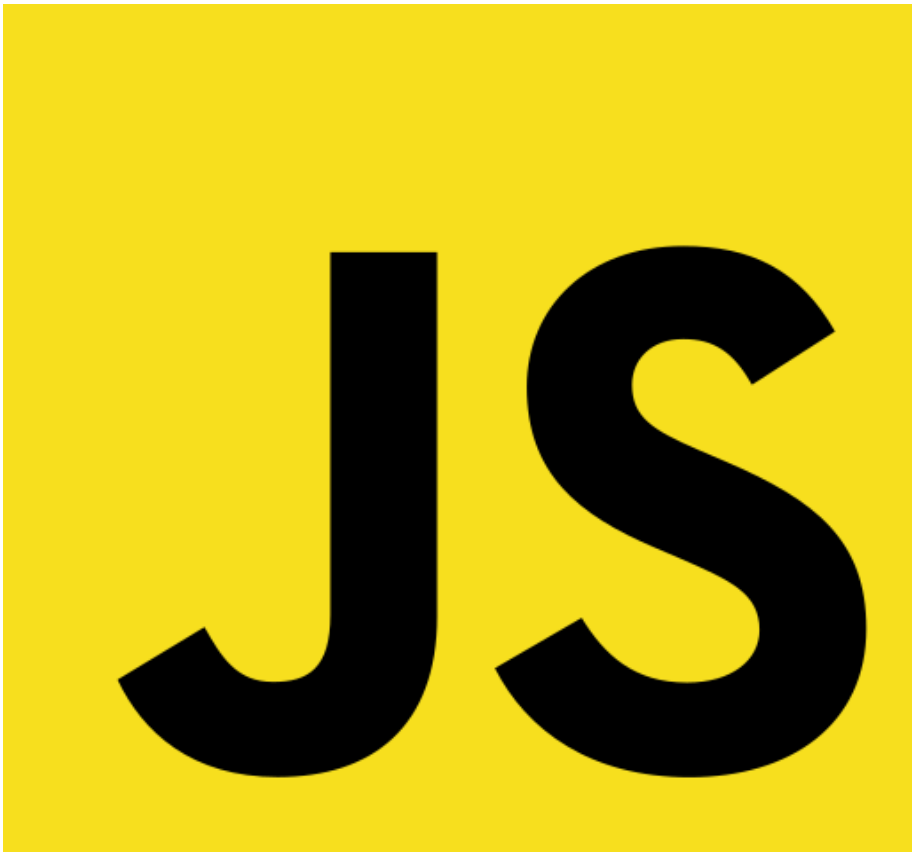

Figura 1.1: JavaScript

1.1. ¿ Qué es javascript?

JavaScript es un lenguaje de programación creado por Brendan Eich en tan solo diez días en mayo de 1995 para dar vida a las páginas web.

Historia de JavaScript https://es.wikipedia.org/wiki/JavaScript

Los programas escritos en este lenguaje se llaman **scripts**. Se pueden escribir directamente en el HTML de una página web y ejecutarse automáticamente a medida que se carga la página.

Los scripts se proporcionan y ejecutan como **texto plano**. No necesitan preparación especial o compilación para correr.

JavaScript es diferente a Java. Cuando JavaScript fue creado, inicialmente se llamaba LiveScript. Pero Java era muy popular en ese momento, así que se decidió que el posicionamiento de un nuevo lenguaje como un *Hermano menor* de Java ayudaría. A medida que evolucionaba, JavaScript se convirtió en un lenguaje completamente independiente con su propia especificación llamada ECMAScript https://es.wikipedia.org/wiki/ECMAScript. No tiene ninguna relación con Java.

Hoy, JavaScript puede ejecutarse no solo en los navegadores, sino también en **servidores** o incluso en cualquier dispositivo que cuente con un programa especial llamado *El motor o intérprete de JavaScript*.

El navegador tiene un motor embebido a veces llamado *Máquina virtual de JavaScript*. Los diferentes motores tienen diferentes nombres como V8 para Chrome, Opera y Edge https://es.wikipedia.org/wiki/Chrome_V8

Los motores son complicados, pero los fundamentos son fáciles.

1. El motor (embebido si es un navegador) lee (analiza) el script.

2. Luego convierte (compila) el script a lenguaje de máquina.

3. Por último, el código máquina se ejecuta.

El motor aplica **optimizaciones** en cada paso del proceso. Incluso observa como el script compilado se ejecuta, analiza los datos que fluyen a través de él y aplica optimizaciones al código maquina basadas en ese conocimiento.

Las principales características de JavaScript son:

- **'Lenguaje del lado del cliente'**

  Cuando se dice que un lenguaje es del lado del cliente, nos referimos a que se ejecuta en la máquina del propio cliente a través de un navegador.

  JavaScript ahora mismo es un lenguaje muy popular. De hecho, desde hace años se ha creado una versión que es capaz de ser ejecutada también en el lado del **servidor** (Node JS). Por tanto, ahora mismo se ejecuta JavaScript en los navegadores y en los servidores.

- **Lenguaje orientado a objetos**

- **De tipado débil o no tipado.**

  Que un lenguaje sea de tipado débil quiere decir que no es necesario especificar el tipo de dato que se va almacenar al declarar una variable. Esta característica supone una gran ventaja a la hora de ganar rapidez programando, pero puede provocar que cometamos más errores.

- **De alto nivel**

  Su sintaxis es fácilmente comprensible por su similitud al lenguaje de las personas.

1.2. ¿ Cómo incluir JS en documentos HTML?

La integración de JS y HTML es muy flexible y existen al menos tres formas para incluir código en las páginas web.

1.2.1. En el mismo documento HTML

El código JS se encierra entre etiquetas **script** y se incluye en cualquier parte del documento. Si lo ponemos dentro de la etiqueta head, primero se ejecutará el código JS y luego se cargará la página. Si incluimos las etiquetas script en el body; primero se cargará la página y luego se ejecutará el código.

Se pueden poner tantos bloques de código como necesitemos.

Mediante el atributo **language** o **type** se debe indicar si se trata de un script de JS o de cualquier otro lenguaje (si no se indica se sobreentiende que es un script de JS).

```
1
2  Ejemplo uno:
3
4  <!DOCTYPE html>
5  <html lang="en">
6  <head>
7      <meta charset="UTF-8">
8      <title>Ejemplo uno</title>
9      <script>
10         alert("Un mensaje en head ");
11         // alert es como showMessageDialog
12     </script>
13 </head>
14 <body>
15
16     <script>
17         alert("Un mensaje en body ");
18     </script>
19
20     <p>Párrafo.</p>
21 </body>
22 </html>
```

Podemos encontrar más información sobre la etiqueta script en páginas del tipo http://www.w3schools.com/tags/tag_script.asp o https://developer.mozilla.org/es/docs/Web/SVG/Element/script

1.2.2. En un fichero

Las instrucciones se pueden incluir en un archivo externo de tipo JS que los documentos HTML enlazan mediante la etiqueta script. Se pueden crear

todos los archivos que sean necesarios y cada documento HTML puede enlazar tantos archivos JS como necesite.

Este método requiere utilizar el atributo **src**, que es el que indica la dirección del archivo JavaScript que se quiere enlazar. Una etiqueta script solo puede enlazar un archivo, pero en una misma página se pueden incluir tantas etiquetas script como sean necesarias.

Los archivos de tipo JavaScript son documentos normales de texto con la extensión .js que se pueden crear con cualquier editor de texto.

La principal ventaja de enlazar un archivo JS es que se simplifica y ordena el código de la página; se puede reutilizar y que cualquier modificación realizada en el archivo JS se ve reflejada inmediatamente en todas las páginas que lo enlazan.

```

Ejemplo dos:

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo dos</title>
    <script src="f21.js"></script>
</head>
<body>
    <p>Texto</p>
    <script src="f22.js"></script>
</body>
</html>
```

```

// Contenido del fichero f21
alert("Head -- Código js en un fichero");


/// Contenido del fichero f22
alert("Body -- Código js en un fichero");
```

Es posible incluir archivos locales (mediante la ruta absoluta o relativa) o de otros servidores indicando la URL.

```
<script
    src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
```

1.2.3. Incluir JavaScript en los elementos HTML

Este método consiste en incluir trozos de JS dentro del código de la página.

```

Ejemplo tres:

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo tres</title>
</head>
<body>
    <p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
</body>
</html>
```

El mayor inconveniente de este método es que ensucia el código de la página y complica el mantenimiento del mismo.

1.2.4. Etiqueta noscript

Algunos navegadores no disponen de soporte JavaScript, otros navegadores permiten bloquearlo parcialmente e incluso algunos usuarios bloquean completamente el uso de JavaScript porque creen que así navegan de forma más segura.

En estos casos, es habitual que si la página web requiere JavaScript para su correcto funcionamiento, se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página.

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

Configuración – Privacidad y seguridad – Configuración de sitios – JavaScript para habilitar o deshabilitar la ejecución de JS en Chrome

```
Ejemplo cuatro:

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ejemplo cuatro</title>
    <script>
        alert("Un mensaje en head ");
        // alert es como showMessageDialog
    </script>
    <noscript>Este navegador no soporta javascript</noscript>
</head>
<body>

    <script>
        alert("Un mensaje en body ");
    </script>
```

```
19
20      <p>Párrafo.</p>
21  </body>
22  </html>
```

1.3. El lenguaje JavaScript

Para trabajar con JS se puede utilizar un IDE o directamente un navegador.

Creando snippets o nuevos fragmentos de código dentro de fuentes (herramientas – herramientas para desarrolladores), podemos empezar a desarrollar los ejemplos y ejercicios que nos van a permitir familiarizarnos con este lenguaje.

Estos apuntes no son un manual. En Internet podéis encontrar muchos y muy buenos tutoriales completos.

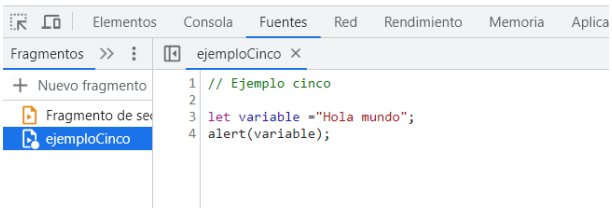

Figura 1.2: Fragmentos de código

*Si queremos restringir algunas de las "libertades"de javascript y usar un JS moderno, es necesario indicar el modo estricto. Hay que escribir **'use strict'** al principio de los scripts.* `https://www.youtube.com/watch?v=7e6ssF78Af4` *(14:22)* `https://www.w3schools.com/js/js_strict.asp`

Para ejecutar directamente código JS desde Visual Studio necesitamos tener node.js instalado.

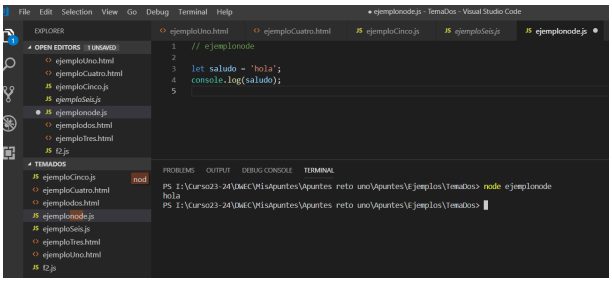

Figura 1.3: Código desde Visual Studio.

Desde vs ejecutando el comando node -v podemos ver si está instalado y en qué versión

Las normas básicas que definen la sintaxis de JS son las siguientes:

- No se tienen en cuenta los espacios en blanco y las nuevas líneas. El intérprete ignora cualquier espacio en blanco sobrante, por lo que el código se puede y debe ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, ...)

- Se distinguen las mayúsculas y minúsculas.
- **No se define el tipo de las variables**. Al crear una variable no es necesario indicar el tipo de dato que almacenará. Una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script. Tipado dinámico.
- Las sentencias acaban con ;. Si se nos olvida no pasa nada ya que el salto de línea es el ; implícito.

1.3.1. Comentarios

Los comentarios de una sola línea se definen añadiendo dos barras oblicuas (//) al principio de la línea.

```
1
2 // Se muestra un mensaje a través de la función alert
3 alert("mensaje de prueba");
```

Los comentarios formados por más de una linea se definen encerrando las mismas entre los símbolos /* y */.

```
1
2 /* Los comentarios de varias líneas son muy útiles cuando se necesita
      incluir mucha información */
3
4 alert("mensaje de prueba");
```

1.3.2. Variables

Una variable es una *caja* que se emplea para almacenar y hacer referencia a un valor. Las variables en JS se definen mediante las palabras reservadas **var** o **let**. Let es la versión moderna, var pertenece al javaScript antiguo. En cualquier caso, no son exactamente iguales; var declara la variable en cualquier ámbito y let lo hace en el ámbito de bloque.

```
1
2 Ejemplo seis:
3
4 var n1 = 3;
5 var n2;
6 n2 = 1;
7 var resultado = n1 + n2;
8 alert(resultado); // 4
9
10 var v1 = 1;
11 if (v1 == 1)
12 {
```

```
13      var v2 = 2; // variable declarada en un bloque con var
14      alert(v1); //1
15      alert(v2); //2
16  }
17  alert(v1); //1
18  alert(v2); //2 Fuera del bloque, existe. Ha sido declarada con var.
19
20
21  var v3 = 1;
22  if (v3 == 1)
23  {
24      let v4 = 2;
25      alert(v3); // 1
26      alert(v4); // 2
27  }
28  alert(v3); // 1
29  alert(v4); // Error v4 is not defined
```

Una de las características más sorprendentes de JS para los programadores habituados a otros lenguajes de programación como java es que **no es necesario declarar las variables**. En otras palabras, se pueden utilizar variables que no se han definido anteriormente.

```
1
2   Ejemplo siete
3
4   let n1 = 3;
5   let n2 = 1;
6   {
7   resultado = n1 + n2; // Ni var ni let
8   alert(resultado);
9   }
10  alert(resultado); // Es como poner var
```

La variable resultado no está declarada, por lo que JS crea una variable global y le asigna el valor correspondiente. En cualquier caso, se recomienda declarar todas las variables que se vayan a utilizar. El hecho de que no sea obligatorio declarar las variables puede dar lugar a errores difíciles de detectar. Esta situación la podemos evitar utilizando el modo estricto.

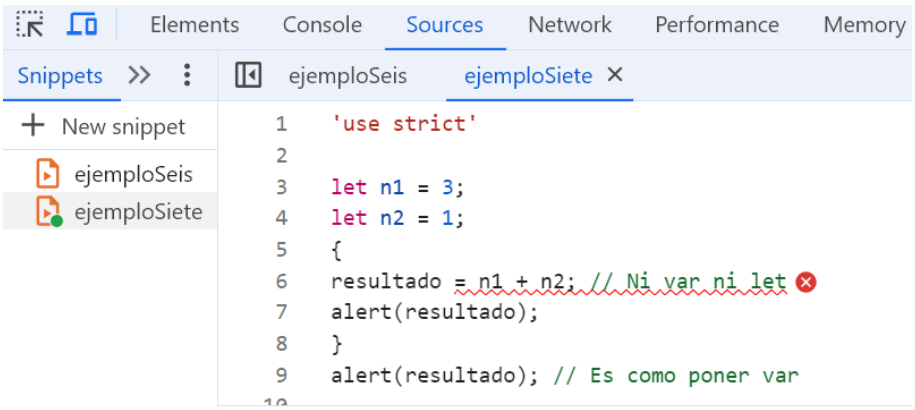

Figura 1.4: Error por no declarar la variable.

El **nombre o identificador de una variable** debe cumplir las siguientes normas:

- Sólo puede estar formado por letras, números y los símbolos dolar y guión bajo.
- El primer carácter no puede ser un número.

```
1 // Correctos
2 let $numero1;
3 let _$letra;
4 let $$$otroNumero;
5 let $_a__$4;
6
7 // Incorrectos
8 let 1numero; // Empieza por un número
9 let numero;1_123; // Contiene un carácter ";"
```

Tipos de datos

Aunque todas las variables se declaran de la misma forma (mediante la palabra reservada let), el valor que se les asigna determina el tipo de dato que almacena (number, string, boolean,...).

Los tipos de datos primitivos que existen en javascript son: **'string'**, **'number'**, **'bigint'** (es un tipo de dato numérico que puede representar números enteros en el formato de precision https://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic), **'boolean'**, **'symbol'** (es un objeto incorporado cuyo constructor devuelve un symbol que está garantizado que es único. Se utilizan a menudo para añadir claves de propiedades únicas a un objeto que no sean iguales a las claves que cualquier otro código pueda añadir al objeto, y que están ocultas de cualquier mecanismo que otro código utilice normalmente para acceder al objeto), **'null'** y **'undefined'**.

Null y undefined sirven para algo muy parecido, que es indicar la ausencia de valor. Lo que ocurre es que undefined es un valor que denota que no hay valor porque no se ha definido todavía, mientras que null se usa para indicar que no hay valor porque así lo hemos querido indicar expresamente.

Figura 1.5: Tipos de datos

```
1
2  // Ejemplo ocho
3
4  let v1;
5  alert(typeof(v1)); // undefined
6
7  v1 = 12.5
8  v1 = -3
9  v1 = 6;
10 alert(v1); // 6
11 alert(typeof(v1));// number
12
13 v1 = parseInt("Hola");
14 alert(v1); // NaN
15 alert(typeof(v1)); // number (no ha hecho la conversión)
16
17 v1=parseInt('3');
18 alert(v1); // NaN
19 alert(typeof(v1)); // number
20
21
22 v1 = 'Hola'
23 v1 = "Adios"
24 v1 = `pepe`
25 alert(v1);
26 alert(typeof(v1)); // string
27
28 v1 = true
29 alert(v1);
30 alert(typeof(v1)); // boolean
31
32 v1 = 23n
33 alert(v1);
34 alert(typeof(v1)); // bigint
35
36 v1 = Symbol(20)
37 alert(v1.toString());
38 alert(typeof(v1)); // symbol
```

Variables numéricas

Se utilizan para almacenar valores numéricos enteros o decimales. El valor se asigna indicando directamente el número entero o real. Los números reales utilizan el carácter . (punto) en vez de , (coma) para separar la parte entera y la parte decimal.

```
1  // Ejemplo nueve
2
3  let iva = 21; // número entero
4  // typeof indica el tipo de dato
5  alert(typeof iva + iva) // number 21
6
7  let total = 234.65; // número real
8  alert(typeof total + total) // number 234.65
9
10 let numero=12e5; // Notación científica.
11 alert(typeof numero + numero) // number 1200000
12
13 // Instanciar la clase Number
14 let cantidad = new Number(); // Constructor vacío
15 alert(typeof cantidad + cantidad) // object 0
16
17 let cantidadDos = new Number(2); // Constructor con dato.
18 alert(typeof cantidadDos + cantidadDos) // object 2
```

Como se puede observar en los ejemplos, también podemos crear **objetos de tipo Number** en vez de variables numéricas. El valor del objeto Number que se crea depende de lo que reciba el constructor de la clase Number. Si el constructor recibe un número, inicializa el objeto con el número que recibe. Si recibe un número entrecomillado lo convierte a valor numérico, devolviendo también dicho número. Devuelve 0 en caso de que no reciba nada. En caso de que reciba un valor no numérico devuelve NaN, que significa "Not a Number"(No es un número). Si recibe false se inicializa a 0 y si recibe true se inicializa a 1.

Algunas urls en las que encontrar información sobre las variables numéricas, los objetos de tipo Number y funciones que se pueden utilizar son las siguientes:

https://www.w3schools.com/jsref/jsref_obj_number.asp

https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_types

También en la clase Math encontraremos métodos que nos permitan realizar operaciones varias con las variables numéricas.

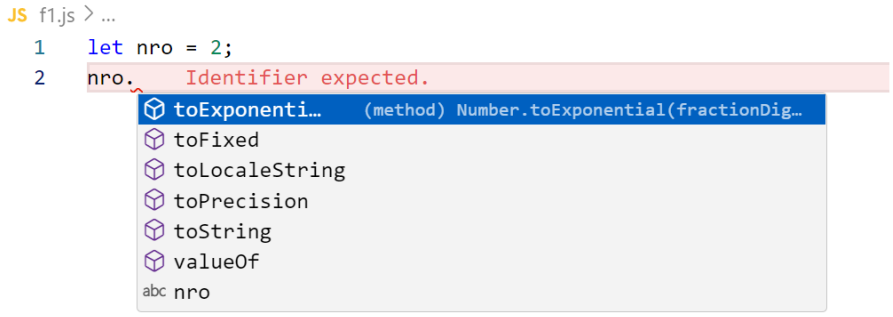

Figura 1.6: Funciones para datos de tipo number

Cadenas de texto o string

Se utilizan para almacenar caracteres alfanuméricos. Para asignar el valor a la variable, se encierra el texto entre comillas dobles o simples.

```
1
2  let mensaje = "Bienvenido";
3
4  let nombreProducto = 'Producto ABC';
5
6  let letraSeleccionada = 'c';
```

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

```
1
2  /* El contenido de texto1 tiene comillas simples, por lo que se encierra
       con comillas dobles */
3
4
5  let texto1 = "Una frase con 'comillas simples' dentro";
6
7
8  /* El contenido de texto2 tiene comillas dobles, por lo que se encierra
       con comillas simples */
9
10
11 let texto2 = 'Una frase con "comillas dobles" dentro';
```

No obstante, a veces las cadenas de texto contienen tanto comillas simples como dobles. Además, existen otros caracteres que son difíciles de incluir en una variable de texto (tabulador, ENTER, etc...). Para resolver estos problemas, JS define un mecanismo para incluir de forma sencilla caracteres especiales y problemáticos dentro de una cadena de texto. El mecanismo consiste en sustituir el carácter por una combinación simple de caracteres. En la imagen se muestra la tabla de conversión que se debe utilizar.

Utilizando estos caracteres, el ejemplo anterior que contenía comillas simples y dobles dentro del texto se puede rehacer de la siguiente forma:

```
1
2  let texto1 = 'Una frase con \'comillas simples\' dentro';
3
4  let texto2 = "Una frase con \"comillas dobles\" dentro";
```

Si se quiere incluir...	Se debe incluir...
Una nueva línea	\n
Un tabulador	\t
Una comilla simple	\'
Una comilla doble	\"
Una barra inclinada	\\

Figura 1.7: Tabla de conversión

Otra manera de crear cadenas de texto consiste en instanciar directamente la clase String. De acuerdo con el estándar ECMA, las cadenas se convierten automáticamente en objetos de tipo String cuando intentas llamar a un método por lo que esta manera de crear cadenas no suele ser la más utilizada.

```
1 let nombre1 = new String();
2 nombre = "pepe";
3 alert(nombre.toUpperCase());
4
5 let nombre2 = new String("Ana");
6 alert(nombre2.toUpperCase());
7
8 let nombre3 = "Manolo";
9 alert(nombre3.toUpperCase());
```

Algunas de las funciones más útiles para el manejo de cadenas de texto las podemos consultar en las direcciones http://www.w3schools.com/jsref/jsref_obj_string.asp y https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String

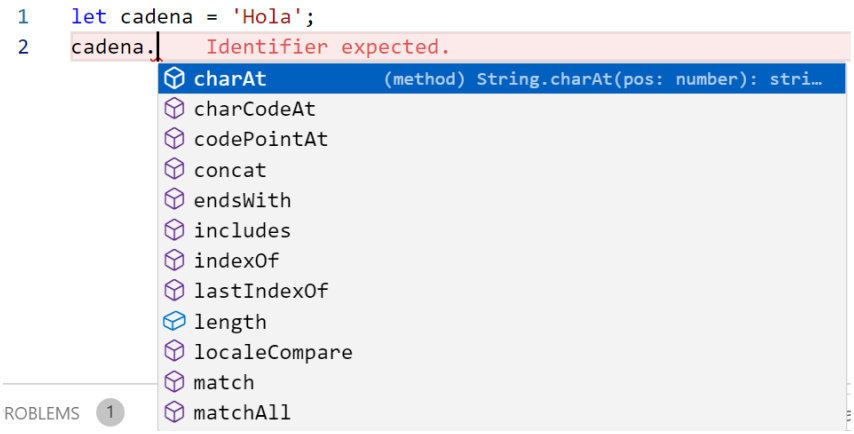

Figura 1.8: Funciones para datos de tipo string

Los operadores typeof e instanceof

Si tenemos en cuenta que una misma variable, en distintos momentos puede almacenar diferentes tipos de datos, cabe la posibilidad de que en un momento determinado necesitemos conocer no el dato, sino el tipo de dato que almacena.

El operador **typeof** devuelve una cadena que indica el tipo del dato almacenado en una variable.

```
1
2  // Ejemplo diez
3
4  let variable = 3
5  alert(typeof variable) //number
6
7  variable = "hola"
8  alert(typeof variable) // string
9
10 variable = new String("adios")
11 alert(typeof variable) // object
12
13 variable = true
14 alert(typeof variable)  // boolean
15
16 let v2
17 alert(typeof v2) // undefined
```

El operador **instanceof** devuelve true si un objeto es una instancia de otro especificado.

```
1
2  // Ejemplo once
3
4  let color1=new String("verde")
5  alert(color1 instanceof String) // devuelve true
6
7  let color2="coral"
8  alert(color2 instanceof String) // devuelve falso color2 no es un objeto
       String
9  //alert(color2 instanceof string) // No funciona. string no es una clase
10
11 color2 = 7;
12 //alert(color2 instanceof number) // No funciona.
13 alert(color2 instanceof Number) // false
```

Booleanos

Una variable de tipo boolean puede tomar dos valores: true (verdadero o 0) o false (falso o 1).

```
1  let clienteRegistrado = false;
```

```
2  let ivaIncluido = true;
3
4  let x = new Boolean();
```

Más información en páginas como https://www.w3schools.com/jsref/jsref_obj_boolean.asp o https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Boolean.

Conversiones

La mayoría de las veces, los operadores y funciones convierten automáticamente los valores que se les pasan al tipo correcto. Esto es llamado *conversión de tipo*. Por ejemplo, 'alert' convierte automáticamente cualquier valor a string para mostrarlo. Las operaciones matemáticas convierten los valores a números.

También hay casos donde necesitamos convertir de manera explícita un valor y para ello disponemos de las siguientes funciones:

parseFloat(cadena) transforma en número real una cadena de caracteres. Devuelve NaN (Not a Number) cuando la cadena de caracteres empieza con caracteres que no se pueden interpretar como parte de un número. Si la cadena contiene caracteres no numéricos hacia el final, el número se interpreta sólo hasta donde empiezan los caracteres no numéricos, y se devuelve como valor la parte interpretada como número.

parseInt(cadena) funciona de forma similar a la función anterior, pero el valor devuelto siempre es un entero.

Number es similar.

El **símbolo +** también convierte un string en un número.

```
1
2  // Ejemplo doce
3
4  let dato = "3.14"; // string
5
6  let datoNuevo = parseInt(dato);
7  alert(typeof(datoNuevo) + datoNuevo); // number 3
8
9  datoNuevo = parseFloat(dato);
10 alert(typeof(datoNuevo) + datoNuevo); // number 3.14
11
12 datoNuevo = Number(dato);
13 alert(typeof(datoNuevo) + datoNuevo); // number 3.14
14
15 datoNuevo=+dato
16 alert(typeof datoNuevo + datoNuevo) // number 3.14
17
18
19 dato = "Hola 3.15"; // Empieza con letras
20 alert(parseInt(dato)); // NaN
21 alert(parseFloat(dato));  // NaN
22 alert(Number(dato)); // NaN
23 alert(+dato) // NaN
24
25
26 dato = "3.14 Hola"; // Empieza con números.
27 alert(parseInt(dato)); // 3
28 alert(parseFloat(dato));  // 3.14
29 alert(Number(dato)); // NaN
```

```
alert(+dato) // NaN
```

También existe **toString** para convertir números en cadenas de caracteres, aunque también se puede hacer concatenando con ' ' o con String.

```

// Ejemplo trece

let a = 7
alert(typeof a) // number

a=a+''
alert(typeof a) // string

let b=7
alert(typeof b) // number

b=b.toString()
alert(typeof b) // string

let c=7
alert(typeof c) // number
c=String(c)
alert(typeof c) // string
```

También se puede convertir un número o un string en un dato de tipo boolean. Los valores vacíos, null, undefined y NaN se convierten en false. Otros valores se convierten en true.

```

// Ejemplo catorce

alert( Boolean(1) ); // true
alert( Boolean(0) ); // false

alert( Boolean("hola") ); // string con datos - true
alert( Boolean("") ); // string vacío - false

let n;
alert( Boolean(n) ); // undefined - false
n=0;
alert( Boolean(n) ); // number cero - false
n=7;
alert( Boolean(n)); // number - true

let dato;
alert(Boolean(dato)); // undefined - false

dato = parseInt("Pepe");
alert(Boolean(dato)); // NaN - false
```

Asignación

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

```
1
2  // Ejemplo dieciseis
3
4  let numero1 = 3;
5  let numero2 = 3.78;
6  let nombre='Pepe';
7  let verdadero=true;
8
9  let mayor = numero1 > numero2;
10 alert(mayor) // false
11
12 let nombreNuevo = nombre;
13
14 let suma = numero1 + 2;
```

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

Definición de constantes

A las constantes en JS siempre se las ha echado en falta y se han visto obligados a crear pseudo-constantes simplemente estableciendo el nombre de la variable en mayúsculas (var PSEUDOCONSTANTE) y dando por hecho que nadie la modificará posteriormente. En el JS moderno existe **const** para que no se pueda modificar el valor, .

```
1
2 const IVA = 21;
```

Clase Date

Sobre la clase Date recae todo el trabajo con fechas y horas. Date no es un tipo de datos primitivo pero sí es uno de los tipos de datos más usados además de los números y de las cadenas de caracteres.

http://www.w3schools.com/jsref/jsref_obj_date.asp

https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Date

Para crear un objeto fecha con el día y hora actuales colocamos los paréntesis vacíos al llamar al constructor de la clase Date.

```
1
2  let hoy = new Date();
3
4  alert(hoy) // Wed Aug 23 2023 17:38:04 GMT+0200 (hora de verano de Europa
       central)
5
6  hoy = Date.now(); // new Date().getTime()
7
8  alert(hoy) // 1692805042064 timesmap milisegunodos desde 1 de enero de
       1970
```

Para crear un objeto fecha con un día y hora distintos de los actuales tenemos que indicar entre paréntesis el momento con el que inicializar el objeto.

```
1
2  // Ejemplo dieciseite
3
4  // Fecha y hora
5  let fecha1 = new Date(2030,01,26,13,10,30);
6  alert(fecha1) // Tue Feb 26 2030 13:10:30 GMT+0100 (hora estándar de
       Europa central)
7
8  // Solo fecha (año,mes,dia)
9  let fecha2 = new Date(2030,01,26);
10 alert(fecha2) // Tue Feb 26 2030 00:00:00 GMT+0100 (hora estándar de
       Europa central)
11
12 // Fecha a partir de un string
13 let fecha3 = new Date("2030-01-26");
14 alert(fecha3) // Sat Jan 26 2030 01:00:00 GMT+0100 (hora estándar de
       Europa central)
15
16 // Fecha no correcta -> autocorrección
17 // Solo fecha (año,mes,dia)
18 let fecha4 = new Date(2030,13,26);
19 alert(fecha4) // Wed Feb 26 2031 00:00:00 GMT+0100 (hora estándar de
       Europa central)
```

La clase Date nos proporciona diferentes métodos.

Ejemplos:

El mes comienza por 0, es decir, enero es el mes 0. Los días de la semana también empiezan en 0 (0 Domingo). Si no indicamos la hora, el objeto fecha se crea con hora 00:00:00.

MÉTODO	QUÉ HACE
getDate()	Devuelve el día del mes. Número entre 1 y 31
getDay()	Devuelve el día de la semana. Entre 0 (domingo) y 6 (sábado)
getFullYear()	Devuelve el año con 4 dígitos
getMilliseconds()	Devuelve los milisegundos entre 0 y 9999
getMinutes()	Devuelve los minutos. Entre 0 y 59
getMonth()	Devuelve el mes. Entre 0 (enero) y 11 (diciembre)
getSeconds()	Devuelve los segundos. Entre 0 y 59
getTime()	Devuelve los milisegundos transcurridos entre el día 1 de enero de 1970 y la fecha correspondiente al objeto al que se le pasa el mensaje
parse()	Analiza una fecha y devuelve el número de milisegundos pasados desde el 1 de enero de 1970 hasta la fecha analizada
setDate()	Actualiza el día del mes
setFullYear()	Cambia el año de la fecha al número que recibe por parámetro
setHours()	Actualiza la hora
setMilliseconds()	Establece el valor de los milisegundos
setMinutes()	Cambia los minutos
setMonth()	Cambia el mes (atención al mes que empieza por 0)
setSeconds()	Cambia los segundos
setTime()	Actualiza la fecha completa. Recibe un número de milisegundos desde el 1 de enero de 1970
toTimeString()	Convierte la parte de tiempo de un objeto Date en una cadena

Figura 1.9: Métodos de la clase Date

```
1
2  // Ejemplo dieciocho
3
4  let fecha;
5  fecha=new Date();
6  document.write('Hoy es ');
7  document.write(fecha.getDate()+'/'); // Día del mes
8  document.write((fecha.getMonth()+1)+'/'); // Mes más uno
9  document.write(fecha.getFullYear()); // Año
10 document.write('<br>');
11 document.write('Es la hora ');
12 document.write(fecha.getHours()+':'); // Hora
13 document.write(fecha.getMinutes()+':'); // Minutos
14 document.write(fecha.getSeconds()); //Segundos
15
16 document.write('<br>');
17 fecha.setDate(18); // Modificar el día del mes
18 document.write(fecha);
19
20 document.write('<br>');
21 fecha.setMonth(fecha.getMonth() + 1); // Añadir un mes
22 document.write(fecha);
```

1.4. Operaciones de entrada/salida

1.4.1. alert()

La función alert() permite mostrar al usuario información literal y/o el contenido de variables en una ventana independiente. La ventana contendrá la información a mostrar y el botón Aceptar.

```
1
2  alert("Bienvenido");
3
4  let nombre="María";
5  alert(nombre + "\n bienvenida");
```

También existe el **método write del objeto document** que nos permite escribir en la página.

El objeto document es el que tiene el contenido de toda la página que se está visualizando en el navegador. Esto incluye el texto, imágenes, enlaces, formularios, ... Gracias a este objeto vamos a poder interactuar dinámicamente con los elementos de la página.

```
1
2  document.write("Bienvenidx");
```

También para visualizar mensajes se puede utilizar la **consola**.

```
1
2  // Ejemplo diecinueve
3
4   let nombre = "Pepe";
5
6   console.log("Hola"); // Hola
7   console.log("Hola " + nombre); // Hola pepe
8   console.log('Hola ' + nombre + ' buenos días'); // Hola pepe buenos días
9   console.log('Hola %s buenos días', nombre); // Hola pepe buenos días
```

%d o %i formatea el valor como un número entero.

%f formatea el valor como un número de punto flotante.

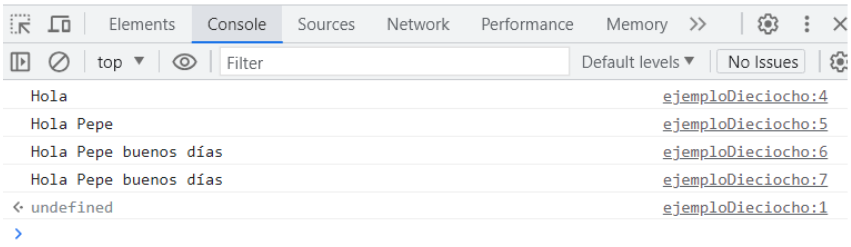

Figura 1.10: Consola

1.4.2. confirm()

A través de la función confirm() se activa un cuadro de diálogo que contiene los botones Aceptar y Cancelar. Cuando el usuario pulsa el botón Aceptar, este método devuelve el valor true; cuando pulsa Cancelar devuelve el valor false.

```
1
2 // Ejemplo veinte
3
4 let respuesta=confirm ("¿ Desea cancelar la subscripción?");
5 document.write("Usted ha contestado que " + respuesta)
6
7 // true aceptar, false cancelar.
```

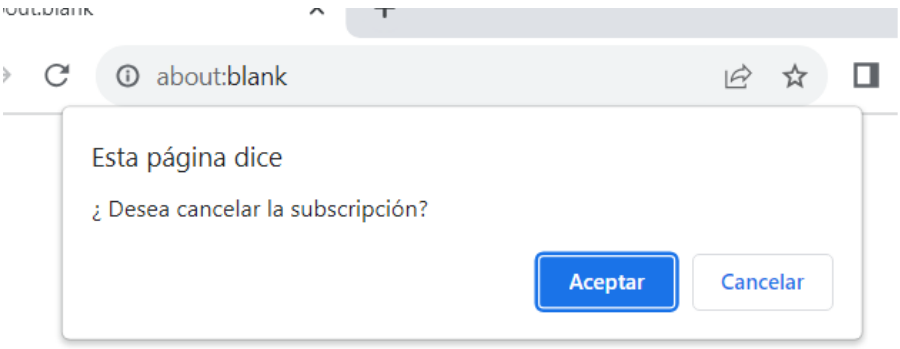

Figura 1.11: Cuadro de diálogo de confirmación

1.4.3. prompt()

La función prompt() abre un cuadro de diálogo en pantalla que pide al usuario que introduzca un dato. Si se pulsa el botón Cancelar, el valor de devolución es null. Pulsando en Aceptar se obtiene la **cadena de caracteres** introducida.

Su sintaxis es prompt(pregunta, respuesta), siendo pregunta la información que se muestra y respuesta la información por defecto que la orden tomará de entrada.

```
1
2  // Ejemplo 21
3
4    let provincia=prompt("Introduzca la provincia ","Asturias");
5    document.write("Usted ha introducido la siguiente información
         "+provincia)
```

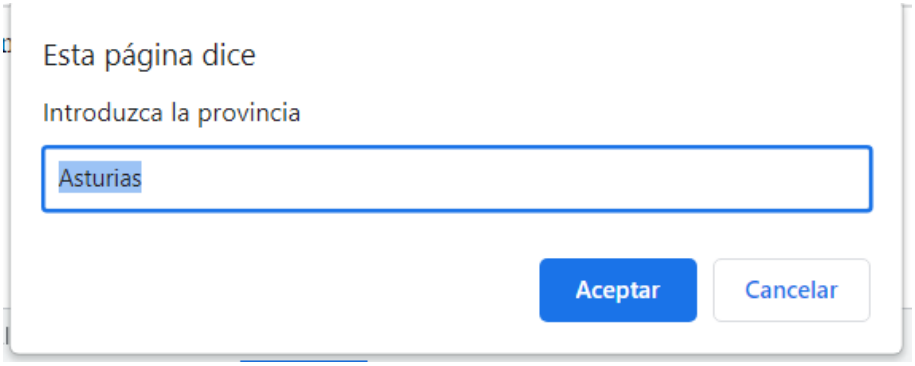

Figura 1.12: Cuadro de diálogo de entrada

1.5. Operadores

Los operadores se dividen en:

- Operador de asignación (=)
- Operadores aritméticos
- Operadores relacionales
- Operadores lógicos
- Operadores a nivel de bit.
- Operadores de cadenas.

Los operadores pueden ser unitarios o binarios; los primeros precisan un único operando y los segundos dos.

Los **operadores aritméticos** son binarios o unitarios. Los operadores unitarios modifican el valor al que se aplican y son:

Incremento ++

Disminución –

Menos unitario -

Los operadores unitarios matemáticos pueden colocarse antes (prefijos) o después (sufijos) del operando y su valor varía según esta posición, ya que el operador prefijo modifica el operando antes de utilizar su valor, mientras el operador sufijo modifica el operando después de haber utilizado el valor.

```
1
2  // Ejemplo 22
3
4  x=10;
5  y=x++; // y=10 y x=11
6  document.write(x + ' ' + y + '<br>')
7
8  x=10;
9  y=++x; // y=11 y x=11
10 document.write(x + ' ' + y)
```

Los operadores binarios matemáticos no cambian el valor de los operandos, sino que guardan el resultado en un tercer operando. Incluyen las principales operaciones aritméticas:

Suma +

Resta -

Multiplicación *

División /

Resto (módulo) %

Exponenciación **

No es posible utilizar el operador de cálculo del módulo cuando los operandos tienen decimales. El operador de división aplicado a variables de tipo entero produce un resultado entero.

El operador de asignación puede ser compactado, es decir, puede combinarse con un operador aritmético. En general, cuando las expresiones son del tipo:

variable = variable operador expresión

pueden cambiarse en:

variable operador = expresión

```
1
2  x += y   x = x + y
3  x -= y   x = x - y
4  x *= y   x = x * y
5  x /= y   x = x / y
6  x %= y   x = x % y
```

La clase Math proporciona los mecanismos para realizar operaciones matemáticas más complejas en JS. Información más completa la podemos encontrar en http://www.w3schools.com/js/js_math.asp, https://www.w3schools.com/jsref/jsref_obj_math.asp o https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Math

Por **operador relacional** entendemos la relación que tiene un valor respecto a otro. Se basan en el concepto de verdadero o falso y en todos los casos operan con sólo dos estados (0/1, verdadero/falso).

```
> Mayor que

>=  Mayor o igual

< Menor que

<=  Menor o igual

==  Igual

!=  Distinto

=== Igualdad estricta (no sólo compara valor sino que además compara el
    tipo).

!== Desigualdad estricta
```

Los operadores relacionales, como ya hemos dicho, producen un resultado 1 si la relación es verdadera, y 0 si la relación es falsa.

Si comparamos diferentes tipos de datos, javascript los convierte en números. Para evitarlo hay que utilizar la igualdad estricta.

```

alert( 0 == false ); // true

alert( 0 === false ); // falso, porque los tipos son diferentes

alert( '' == false ); // true

alert( null === undefined ); // false

alert( null == undefined ); // true
```

```

alert( '2' > 1 ); // true, la cadena '2' se convierte en el número 2
alert( '01' == 1 ); // true, la cadena '01' se convierte en el número 1
```

Los **operadores lógicos** son muy parecidos a los relacionales, en el sentido de que dan también como salida sólo dos valores que, en este caso, son: 0 si la expresión lógica es verdadera y 1 si es falsa.

AND &&

OR ||

NOT !

Los operadores lógicos también se pueden utilizar para obtener el primer valor verdadero de una lista.

```
1
2  // Ejemplo 23
3
4  let firstName = "";
5  let lastName = "";
6  let nickName = "SuperCoder";
7  alert( firstName || lastName || nickName || "Anonymous"); // SuperCoder
8
9  alert(1 && 2 && null && 3); // null
10
11 // Cuando todos los valores son verdaderos, el último valor es retornado:
12 alert(1 && 2 && 3); // 3, el último.
```

Los **operadores sobre cadenas** son:

+ Suma

+= Adjunta

1.6. Estructuras de control de flujo

Una de las estructuras más utilizadas en la mayoría de lenguajes de programación es la estructura if o el operador ?. Se emplea para tomar decisiones en función de una condición. Su definición formal es:

```

if (condición o condiciones)
{
        ...
}
```

Si la condición o condiciones se cumplen, es decir, si su valor es true; se ejecutan todas las instrucciones que se encuentran dentro de las llaves. Si la condición no se cumple, es decir, si su valor es false; no se ejecuta ninguna instrucción y el programa continua ejecutando el resto de instrucciones.

```

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

La condición que controla el if puede combinar los diferentes operadores lógicos y relacionales mostrados anteriormente (and, or, not) y así encadenar varias condiciones.

```

var mostrado = false;
if(!mostrado)    // Mejor que mostrado == false
{
  alert("Es la primera vez que se muestra el mensaje");
}

var mostrado = false;
var usuarioPermiteMensajes = true;
if(!mostrado && usuarioPermiteMensajes)
{
  alert("Es la primera vez que se muestra el mensaje");
}
```

1.6.1. Estructura if...else

```

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

```
{
...
}
```

Si la condición se cumple se ejecutan todas las instrucciones que se encuentran dentro del if. Si la condición no se cumple se ejecutan todas las instrucciones contenidas en la rama del else.

```

let edad = 18;
if(edad >= 18)
{
      alert("Eres mayor de edad");
}
else
{
     alert("Todavía eres menor de edad");
}
```

```

let nombre = "";
if(nombre === "")
    alert("Aún no nos has dicho tu nombre");
else
    alert("Hemos guardado tu nombre");
```

Las estructuras if o if...else se pueden encadenar para realizar varias comprobaciones seguidas.

```

if(edad < 12)
{
  alert("Todavía eres muy pequeño");
}
else
    // ya sabemos que edad >=12
    if(edad < 19)
    {
    alert("Eres un adolescente");
  }
  else
  {
       // ya sabemos que edad >=19
      if(edad < 35)
      {
          alert("Aún sigues siendo joven");
      }
    else
    {
      alert("Eres mayor");
    }
  }
```

1.6.2. Operador ternario

El operador ternario nos permite escribir una sentencia if-else en una única línea. Este operador está formado por tres partes. La primera es la condición, la segunda que hacer si se cumple la condición y la tercera que hacer si no se cumple la condición. Las distintas partes están separados por ? y : respectivamente.

CONDICION ? ResultadoCierto : ResultadoFalse;

```
1
2  let edad = 22;
3
4  if (edad >= 18)
5    alert("Puede votar");
6  else
7    alert("No puede votar");
8
9
10
11 // Operador ternario
12 // (CONDICION) ? RESULTADO_CIERTO : RESULTADO_FALSO
13
14 let puedeVotar = (edad >= 18) ? "Puede votar" : "No puede votar";
15 alert (puedeVotar);
```

1.6.3. Operador Nullish Coalescing '??'

Si utilizamos este operador, una expresión es definida cuando no es 'null' ni 'undefined'. Es decir, ?? devuelve el primer argumento que no es 'null' ni 'undefined'.

El resultado de a ?? b:

- si 'a' está definida, será 'a',

- si 'a' no está definida, será 'b'.

```
1
2  let a;
3  let b;
4  alert(a ?? b) // undefined
5  a = 6;
6  alert(a ?? b) // 6
7  b=4;
8  alert(a ?? b) // 6
9  let c;
10 alert(c ?? b) // 4
```

1.6.4. Estructura switch

La instrucción switch es una alternativa para reemplazar los if/else if cuando la condición verifica si es igual a cierto valor.

```
1
2  switch(x) {
3    case 'valor1':  // if (x === 'valor1')
4      ...
5      break
6
7    case 'valor2':  // if (x === 'valor2')
8      ...
9      break
10
11   default:
12     ...
13     [break]
14 }
```

La variable que analizamos debe ir después de la instrucción switch entre paréntesis. Cada valor que se analiza debe ir después de la palabra clave 'case'.

Hay que poner 'break' al finalizar cada caso. Las instrucciones que hay después de la palabra clave default se ejecutan en caso que la variable no coincida en algún case. El default es opcional.

```
1
2    let valor;
3    valor=parseint(prompt('Ingrese un valor comprendido entre 1 y 5:',''));
4
5    switch (valor)
6     {
7        case 1: document.write('uno');
8                  break;
9        case 2:
10       case 3:
11         document.write('dos o tres');
12                 break;
13       case '4':
14         // No se ejecuta nunca, porque no coinciden los tipos de datos.
15         document.write('cuatro');
16               break;
17       case 5: document.write('cinco');
18               break;
19       default:
20             document.write('debe ingresar un valor comprendido entre 1 y
                   5.');
21   }
```

1.6.5. Estructura for

```
1
2  for(inicializacion; condicion; actualizacion)
3  {
4           ...
5  }
```

La idea del funcionamiento de un bucle for es la siguiente: mientras la condición indicada se cumpla se repite la ejecución de las instrucciones definidas dentro del for. Ademas, después de cada repetición, actualiza el valor de la variable que se utiliza en la condición.

```
1
2  let mensaje = "Hola, estoy dentro de un bucle";
3
4  // let i=0;
5  for(let i = 0; i < 5; i++)
6  {
7         alert(mensaje);
8  }
9
10 // Dos variables.
11 for(x=1,y=10; x <= 10 && y!=0; x++,y--)
```

1.6.6. Estructura repetitiva (do/while)

La sentencia do/while es otra estructura repetitiva que ejecuta al menos una vez su bloque de instrucciones.

La condición de la estructura está después del bloque a repetir, a diferencia del while que está en la parte superior.

Finaliza la ejecución del bloque repetitivo cuando la condición retorna falso igual que el while y el for.

```
1
2   let valor;
3    do
4    {
5        valor=prompt('Ingrese un valor entre 0 y 999:','');
6        valor=parseInt(valor);
7        if (valor != 0)
8        {
9          if (valor<10)
10         {
11           alert('El valor '+ valor + ' tiene  1 dígito');
12             }
13       else
14       {
15           if (valor<100)
16           {
```

```
17              alert('El valor '+ valor +' tiene 2 dígitos');
18          }
19          else
20          {
21              if (valor < 1000)
22              {
23                  alert('El valor '+ valor + ' tiene 3 dígitos');
24                  }
25                    else
26                    {
27                       alert("Dato no válido");
28                    }
29                }
30      }
31      }
32  }
33  while(valor!=0);
```

1.6.7. Estructura repetitiva while

El bucle while (mientras) tiene la siguiente sintaxis:

```
1
2  while (condicion) {
3    // código
4    // llamado "cuerpo del bucle"
5  }
```

Mientras la condición sea verdadera, el código del cuerpo del bucle será ejecutado.

Por ejemplo, el bucle debajo imprime i mientras se cumpla i ¡3:

```
1
2  let i = 0;
3  while (i < 3) { // muestra 0, luego 1, luego 2
4    alert( i );
5    i++;
6  }
```

1.6.8. Break y continue

La sentencia break se puede usar para salir de un bucle.

```
1
2  let sum = 0;
3
4  while (true) {
5
6    let value = +prompt("Ingresa un número", ''); // + delante de prompt
7
```

```
8     if (!value) break;
9
10    sum += value;
11
12  }
13  alert( 'Suma: ' + sum );
```

Técnicamente es más correcto salir de una repetitiva cuando la condición ha dejado de cumplirse.

La sentencia continue se puede usar para reiniciar una sentencia while, do-while o for.

```
1
2   i = 0;
3   n = 0;
4   while (i < 5)
5   {
6     i++;
7     if (i == 3)
8     {
9         continue;
10    }
11    n += i;
12  }
```