

Desarrollo Web en entorno cliente

UT 2. Variables y estructuras de control en JS



T.S. en Desarrollo de Aplicaciones Web

IES Escultor J.L. Sánchez

Contenido

1. Comenzando a trabajar con JS	3
Edición y ejecución de JS.....	4
2. Variables, tipos de datos y operadores en JS.....	7
2.1. Variables.....	7
2.2. Tipos de datos	10
2.3. Operadores y expresiones.....	13
3. Estructuras de control en JS	17
3.1. Estructuras condicionales	17
3.2. Estructuras repetitivas (bucles).....	19
4. Referencias bibliográficas	20

1. Comenzando a trabajar con JS

La programación de Javascript (en el cliente web) se realiza dentro del propio documento HTML; el código Javascript debe ir encerrado entre las etiquetas `<script> </script>`

Cuando el navegador lee la página y encuentra un script, va interpretando las líneas de código y las va ejecutando una después de otra.

Con la etiqueta `<script>` se utiliza el atributo *type* para indicar qué tipo de codificación de script estamos haciendo y el lenguaje utilizado.

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

La colocación de estos scripts no es indiferente; si el código fuente utiliza elementos del DOM, es necesario que éstos elementos hayan sido procesados previamente. Existen distintas formas de solucionar esta situación para evitar problemas en la ejecución.

Otra manera de incluir scripts en páginas web, implementada a partir de Javascript 1.1, consiste en definir el código fuente en archivos externos, con extensión `.js`. Para incluir el archivo externo en la página HTML, la sintaxis es:

```
<SCRIPT type="text/javascript" src="archivo_externo.js">
</SCRIPT>
```

El archivo `.js` externo (en este ejemplo el *archivo_externo.js*) debe contener únicamente sentencias Javascript. No debemos incluir código HTML de ningún tipo, ni tan siquiera las etiquetas `<script>` y `</script>`

También se puede escribir Javascript dentro de determinados atributos de la página, como el atributo *onclick*, aunque es la opción menos recomendable ya que **no se debe mezclar contenido (formateado con HTML), aspecto (maquetación CSS) y comportamiento (dinamismo con Javascript) en una etiqueta HTML.**

Existen dos tipos de **comentarios** en el lenguaje:

- Uno de ellos, la doble barra (`//`), sirve para comentar una línea de código.
- El otro comentario se utiliza para comentar varias líneas y se indica con los signos `/*` para empezar el comentario y `*/` para terminarlo.

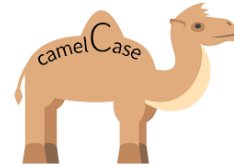
```
<SCRIPT>
//Este es un comentario de una línea
/*Este comentario se puede extender
por varias líneas.*/
</SCRIPT>
```

Javascript es **sensible a mayúsculas y minúsculas**; por ejemplo, *numero* y *Numero* son dos identificadores distintos.

Respecto a la forma que deben tener los identificadores (nombres de variables, funciones, objetos, ...), se siguen los mismos estándares que en otros lenguajes de programación: los nombres de variables y funciones comienzan por minúscula (si están compuestos por dos palabras, la segunda comienza por mayúscula), los nombres de clases/objetos comienza por mayúscula y las constantes se definen en mayúscula. A esta convención se le denomina *Lower Camel Case*:

Por ejemplo:

- *alert()* es una función
- *bgColor()* es una función
- *Math* es un objeto predefinido en el lenguaje
- *PI* es una constante



Las instrucciones en Javascript pueden finalizar de dos formas:

- con punto y coma (;)
- o bien con un salto de línea

A pesar de no ser imprescindible, es recomendable finalizar las instrucciones con ;

Existen, entre otros, varios tipos de **literales** (valores fijos) en Javascript:

- *numéricos* (enteros o reales): 100, 50.5
- de tipo *cadena de caracteres* (encerrados entre comillas simples o dobles): 'hola', "hola"
- booleanos: *true* y *false*

Respecto a los caracteres, Javascript utiliza el conjunto **Unicode**:

<https://unicode-table.com/es/>

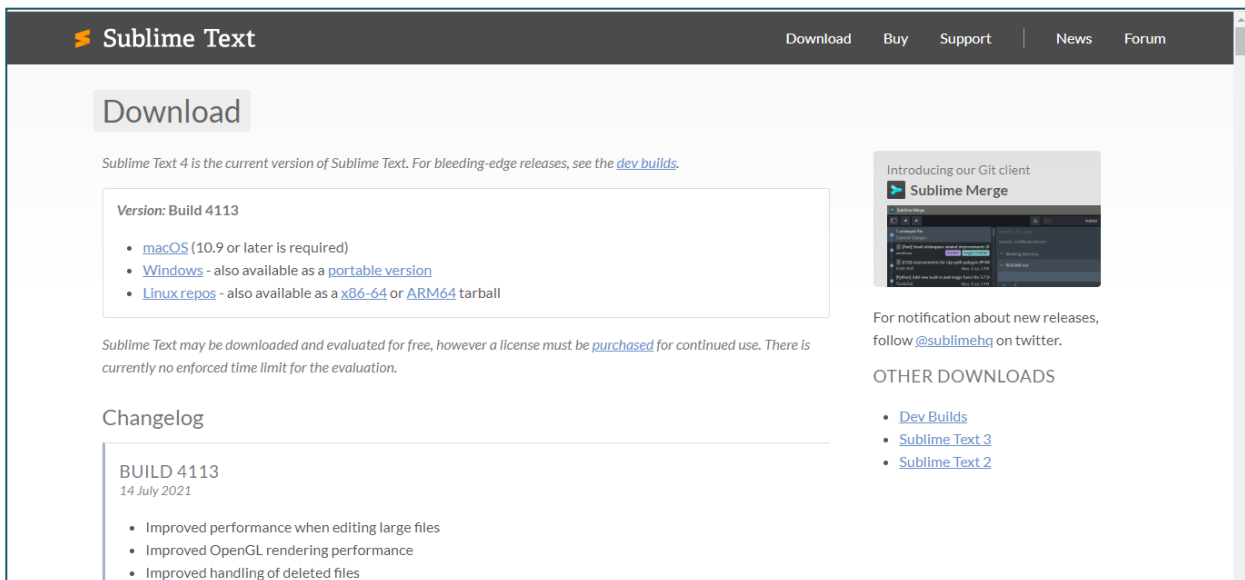
https://www.w3schools.com/charsets/ref_html_utf8.asp

Edición y ejecución de JS

Para editar código Javascript es suficiente con un editor de textos.

En nuestro caso, vamos a utilizar como entorno de desarrollo **Sublime Text**. Se puede descargar e instalar en el ordenador, o bien utilizar la versión portable:

<https://www.sublimetext.com/download>



The screenshot shows the 'Download' section of the Sublime Text website. It features the Sublime Text logo, navigation links (Download, Buy, Support, News, Forum), and a 'Download' heading. Below the heading, it states 'Sublime Text 4 is the current version of Sublime Text. For bleeding-edge releases, see the [dev builds](#).' A box lists the version as 'Build 4113' and provides links for macOS (10.9 or later), Windows (portable version), and Linux repos (x86-64 or ARM64 tarball). A note mentions that while the software is free, a license must be purchased for continued use. A 'Changelog' section for 'BUILD 4113' dated '14 July 2021' lists improvements in performance and file handling. On the right, there's a 'Sublime Merge' advertisement and a section for 'OTHER DOWNLOADS' including Dev Builds, Sublime Text 3, and Sublime Text 2.

Una vez editado el código, lo ejecutaremos en un navegador web (**Chrome**)

También existen entornos en los que se pueden hacer pruebas y ver directamente su resultado.

- ✓ Editor en línea para HTML5/CSS3/JavaScript: <http://liveweave.com/>
- ✓ JsFiddle: <https://jsfiddle.net/>
- ✓ JS Nin: <https://jsbin.com/?html,output>

Actividad: vamos a diseñar nuestro primer programa JS; por supuesto, ¡hola mundo Javascript!

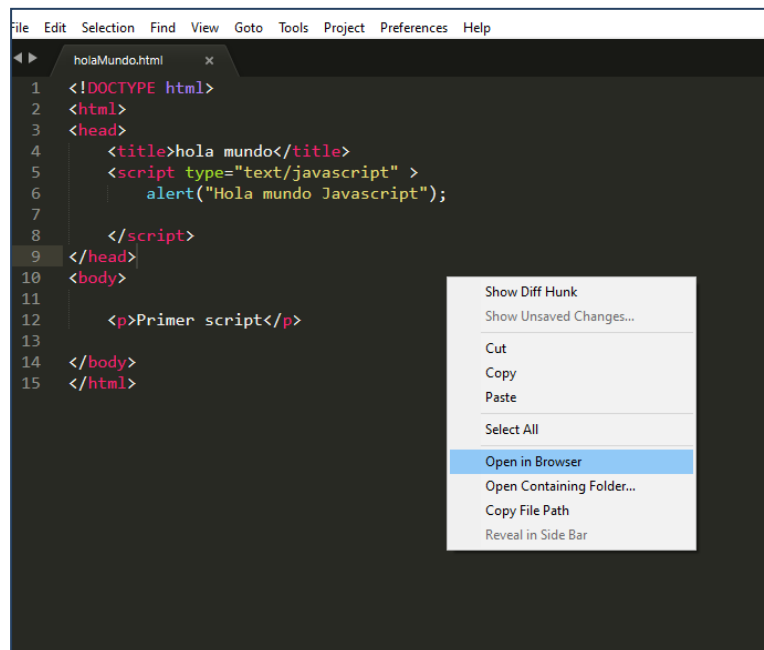
1º Abrimos el editor *Sublime Text* y editamos la página *HolaMundo.html* con el siguiente contenido:



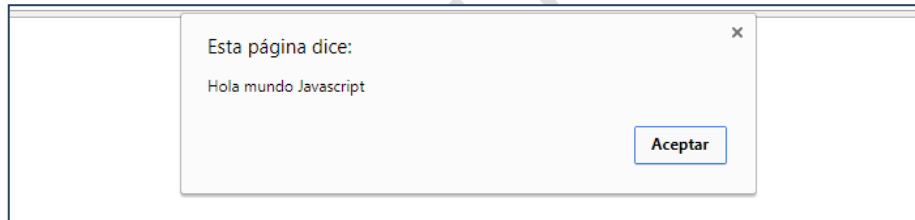
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Primer script</title>
5 <script type="text/javascript">
6     alert("Hola mundo Javascript");
7 </script>
8 </head>
9 <body>
10
11
12 <p>Primer script</p>
13
14
15 </body>
16 </html>
```

El código JS se incluye en la cabecera del documento HTML y muestra una alerta con el mensaje deseado

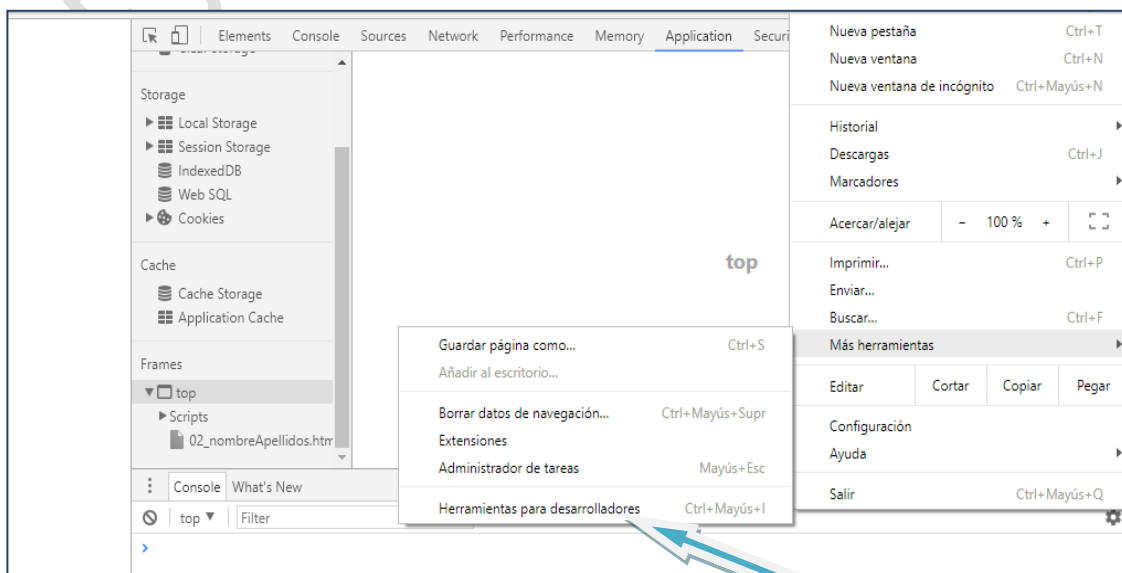
2º Ejecutamos el archivo en el navegador web; podemos ejecutarlo en modo local desde el propio entorno de desarrollo (botón derecho del ratón para ver el menú contextual):



El resultado de la ejecución es:

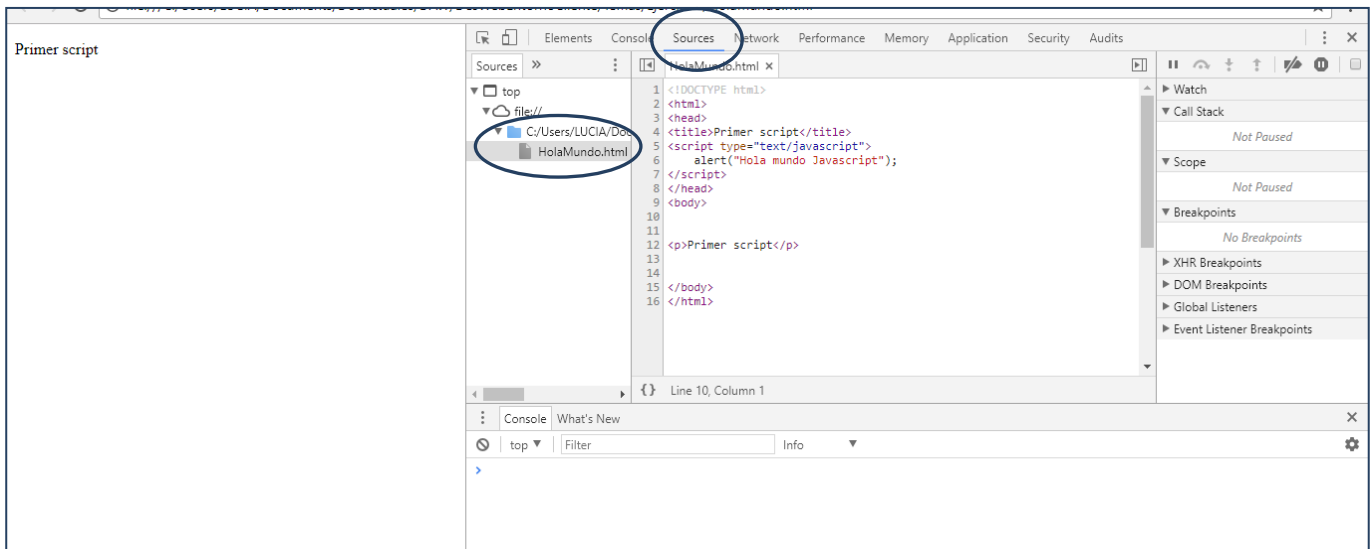


En Chrome, podemos hacer un **seguimiento de la ejecución** de nuestras páginas web con la opción "**herramientas para desarrolladores**" (atajo: **F12**). Resulta muy útil, entre otras cosas, para **depurar** nuestro trabajo.



En la ventana de la izquierda vemos la página en ejecución y en la ventana de la derecha encontramos las herramientas para desarrolladores.

Dentro de las herramientas, en la parte superior, pestaña "sources", podemos ver el código de nuestra página HTML y de los archivos .js, además de información sobre ámbitos y variables.



Actividad 1: modifica la actividad anterior para que el script se localice en un archivo independiente a la página HTML; el archivo se llamará *HolaMundo.js* y se localizará en la carpeta *js*

2. Variables, tipos de datos y operadores en JS

2.1. Variables

Los nombres de variables deben seguir unas reglas sintácticas en su construcción:

- deben contener caracteres alfanuméricos, el carácter subrayado (`_`) o el carácter `$` y no pueden contener caracteres especiales ni espacios en blanco
- tienen que comenzar por un carácter alfabético o el subrayado o bien `$`.

En Javascript no es obligatorio declarar las variables que se van a utilizar, pero sí es recomendable; además, existen diferencias en el uso entre variables declaradas y variables no declaradas. Para declarar variables en ES5 se utiliza la palabra reservada **var**.

Se puede declarar una o varias variables con la palabra reservada *var*, y se puede inicializar una variable en el momento de la declaración.

Ejemplos:

```
var operando1;  
var operando2;
```

Cuando se declara una variable y no se inicializa, por defecto toma el valor **undefined**.

```
var operando1 = 23;  
var operando2 = 33;
```

```
var operando1, operando2;
```

El **ámbito** de una variable depende del lugar dónde se declara:

- **ámbito global** (la página web): variables declaradas con la palabra reservada *var* fuera de las funciones o bien variables no declaradas.
- **ámbito local**: variables declaradas con la palabra reservada *var* dentro de una función.

Ejemplo:

```
<SCRIPT>  
  
var numero = 2 ; // numero es una variable de ámbito global  
  
function miFuncion (){  
  //variable global numero utilizada dentro de la función miFuncion()  
  
  numero = 19 ;  
}  
  
console.log(numero); //muestra valor 2 en la consola del navegador  
  
</SCRIPT>
```

Si una variable no se declara con *var*, Javascript considera que se trata de una variable global (ámbito de uso en toda la página web) independientemente de dónde se localice. Ejemplo:

```
<SCRIPT>  
  
function miFuncion (){  
  
  //numero es una variable de ámbito global porque no se ha declarado con la  
  //palabra reservada var  
  
  numero = 19 ;  
}  
  
//llamamos a la función  
miFuncion()  
console.log(numero) //imprime 19  
</SCRIPT>
```



```
<SCRIPT>

function miFuncion (){
//en este caso numero es una variable local a miFuncion()
    var numero = 19 ;
}
//llamamos a la función
miFuncion()

console.log(numero) //error, la variable numero no está definida

</SCRIPT>
```

Se recomienda siempre declarar las variables, sin importar si están en una función o un ámbito global.

Por otra parte, cuando se trabaja en modo estricto (*strict mode*), asignar valor a una variable sin declarar lanzará un error.

En **ES5** no se permite declarar variables con un alcance más acotado, como las llaves de una condición *if*, un bucle, ...; como acabamos de ver, el ámbito de una variable es local a una función o global a la página.

En **ES6** se ha implementado una forma nueva de declarar variables; se trata de la construcción **let**.

Comparamos la ejecución de dos bloques de código que utilizan *var* y *let*:

```
<script>
for ( var i= 0 ; i< 10 ; i++){
    console .log(i);
}
console .log(i + 3 ); //muestra 13 en consola
</script>
```

```
<script>
for ( let i= 0 ; i< 10 ; i++){
    console .log(i);
}
console .log(i + 3 ); //Uncaught ReferenceError: i is not defined
</script>
```

Otra diferencia entre *let* y *var* es el mecanismo de doble pasada del intérprete de Javascript. Utilizando *var*, en la primera pasada leerá todas las variables definidas de esta forma y en la segunda al ejecutar el código ya conocerá esas variables y las podrá usar. Así pues, una

declaración *var* nos permite usar una variable que ha sido declarada más adelante en el código, siempre que su ámbito lo permita.

Sin embargo, se recomienda siempre **declarar variables al inicio de su ámbito**.

De forma análoga a otros lenguajes, en **ES6** se ha definido un tipo de variable que sólo puede asignarse en su declaración, y no puede ser modificada: **const**

Ejemplo:

```
const PI = 3.1416;  
PI = 3.14159; // ¡¡ERROR!!
```

El ámbito de una constante se limita al bloque dónde ha sido declarada (como ocurre con *let*)

Cuando se define como constante una referencia a un objeto, no se le puede asignar un nuevo valor a dicha constante, pero el objeto referenciado sí puede modificar su valor.

2.2. Tipos de datos

En Javascript existen datos primitivos y complejos.

Los tipos de datos **primitivos** son: *number*, *string*, *boolean*, *null*, *undefined*.

Los tipos **complejos** son *object* y *function*.

Vamos a trabajar, por el momento, con tipos primitivos.

Numéricos (*number*)

En JS sólo hay un tipo de dato numérico, que se puede escribir con o sin decimales, o bien en notación científica.

Ejemplos:

```
var x1 = 34.00;    // Written with decimals  
var x2 = 34;       // Written without decimals  
var y = 123e5;     // 12300000  
var z = 123e-5;    // 0.00123
```

Los números se pueden expresar en base decimal, octal o hexadecimal:

- Base decimal: cualquier número, por defecto, se expresa en base decimal
- Base octal: para escribir un número en octal basta con escribir dicho número precedido de un 0; por ejemplo 045.
 - ✓ En "modo estricto" esta notación no está permitida.
 - ✓ En ES6 la notación octal se escribe con el prefijo 0o: 0o45

- Base hexadecimal: para escribir un número en hexadecimal debemos escribirlo precedido de un cero y una equis; por ejemplo 0x3EF.

Cadenas de caracteres (string)

Son conjuntos de caracteres; los valores de tipo *string* tienen que ir encerrados entre comillas simples o dobles.

Ejemplos:

```
var carName = "Volvo XC60"; // Using double quotes
var carName = 'Volvo XC60'; // Using single quotes
```

Se pueden utilizar comillas dentro de un valor *string*, siempre y cuando no coincidan con las comillas que rodean la cadena. Ejemplos:

```
var answer = "He is called 'Johnny'"; // Single quotes inside double quotes
var answer = 'He is called "Johnny"'; // Double quotes inside single quotes
```

En JS, las cadenas de caracteres pueden incluir caracteres de escape:

Carácter de escape	Significado
\n	Salto de línea
\'	Comilla simple
\"	Comilla doble
\t	Tabulador
\r	Retorno de carro
\f	Avance de página
\b	Retroceder espacio
\\	Barra invertida

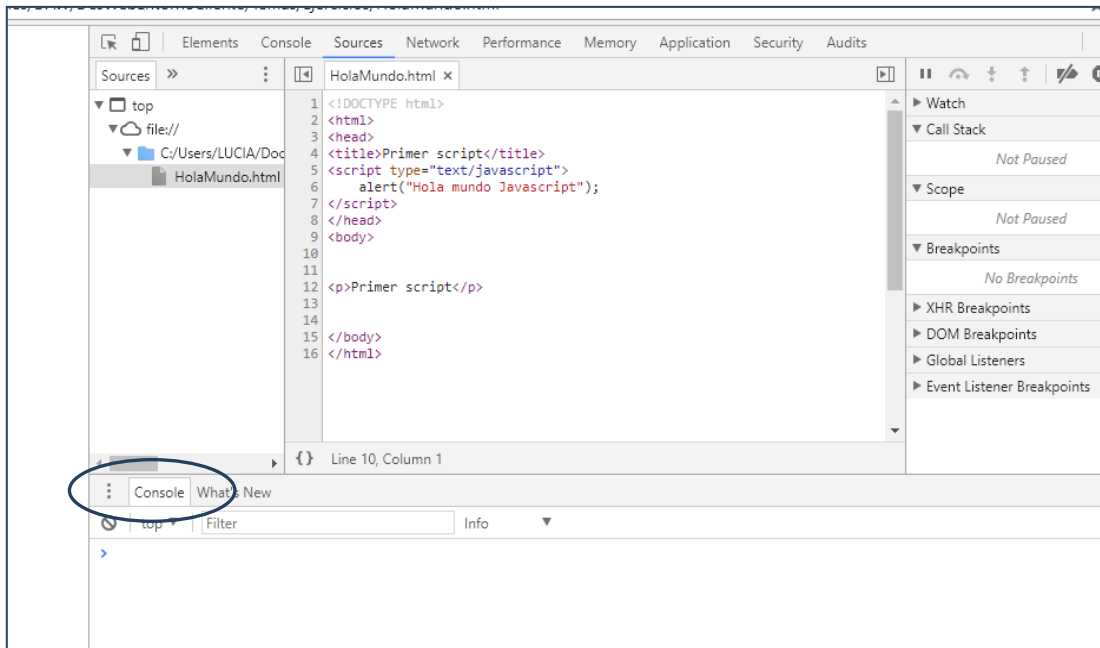
Actividad 2: diseña una página HTML que contenga un script que realice las siguientes tareas:

1º. Pedirá al usuario que introduzca su nombre. Para pedir un dato al usuario podemos utilizar la función *prompt()* que muestra, en un cuadro de diálogo, el mensaje que recibe por parámetro y devuelve el valor introducido por el usuario (en forma de *string*) o bien el valor *null* (si cancela la entrada)

- ✓ Ejemplo: *prompt("introduce tu nombre: ")*
- ✓ Recuerda guardar el valor que devuelve *prompt()* para no perderlo

2º. Mostraremos el nombre recibido en la consola del navegador con la función *console.log()*

Podemos ver la consola del navegador en las herramientas administrativas:



Boolean

Sólo pueden tomar dos valores: **true** o **false**.

undefined

En JS, una variable a la que no se le ha asignado ningún valor, es de tipo **undefined**.

Ejemplo:

```
var num; //num tiene valor indefinido y tipo undefined
```

null

Representa un valor vacío. Sin embargo, el tipo de dato **null** es un objeto.

Ejemplo:

```
var num = null; //num tiene valor null, pero el tipo es object
```

Tipos de datos dinámicos

JS trabaja con **tipos dinámicos**; esto significa que la misma variable puede contener diferentes tipos de datos. Ejemplo:

```
var x;           // Now x is undefined
var x = 5;       // Now x is a number
var x = "John";  // Now x is a string
```

En principio, trabajar con tipos de datos dinámicos puede parecer una ventaja, sobre todo para programadores inexpertos, ya que evita tener que estar pendiente de declaraciones y asignaciones de variables y datos. Pero a la larga puede ser fuente de errores ya que, dependiendo del tipo de dato que contenga una variable, ésta se comportará de un modo u otro y si no controlamos con exactitud los tipos utilizados podemos encontrarnos sumando un texto a un número. Javascript operará perfectamente, y devolverá un dato, pero en algunos casos puede que no sea lo que estábamos esperando. Por ejemplo:

```
var sumando1 = 23 ;  
var sumando2 = "33" ;  
var suma = sumando1 + sumando2 ;  
console.log(suma)
```

Este segmento de código mostrará en la consola el texto "2333", en lugar de la suma de los números 23 y 33.

2.3. Operadores y expresiones

Operadores aritméticos

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
++	Incremento
--	Decremento

Operadores de asignación

Operador	Ejemplo	Equivale a
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Operadores con string

Operador concatenación: + y +=

Ejemplos:

```
cadena1 = "hola"  
cadena2 = "mundo"  
cadenaConcatenada = cadena1 + cadena2 //cadenaConcatenada vale "holamundo"
```

Al concatenar (+) números y string, el resultado es de tipo string:

```
miNumero = 23  
miCadena1 = "pepe"  
miCadena2 = "456"
```

```

resultado1 = miNumero + miCadena1 //resultado1 vale "23pepe"
resultado2 = miNumero + miCadena2 //resultado2 vale "23456"
miCadena2 += miNumero //miCadena2 ahora vale "45623"

```

Operadores condicionales

Operator	Description
==	Igual que (sólo valor)
===	Igual valor e igual tipo
!=	Distinto (sólo valor)
!==	Distinto valor y distinto tipo
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Ejemplos: sea x=5

```

x==5           true
x=="5"        true
x===5         true
x==="5"       false

```

Las cadenas de caracteres son comparadas basándose en un orden lexicográfico estándar, usando valores Unicode.

La **comparación de tipos de datos distintos puede dar lugar a resultados inesperados**. Al comparar una cadena con un número, JavaScript convertirá la cadena en un número para hacer la comparación; una cadena vacía se convierte en 0; una cadena no numérica se convierte en *NaN*, que siempre es falsa.

Ejemplos:

Caso	Valor
2 < 12	True //comparación de number
2 < "12"	True //string se transforma a number para realizar comparación
2 < "John"	False //string se transforma a number para realizar comparación
2 > "John"	False //string se transforma a number para realizar comparación
2 == "John"	False //string se transforma a number para realizar comparación
"2" < "12"	false //comparación de string
"2" > "12"	true //comparación de string
"2" == "12"	false //comparación de string

Por tanto, para evitar resultados inesperados, **es conveniente trabajar con los tipos adecuados en cada momento**.

Operadores lógicos

Trabajan, normalmente, con valores booleanos.

Operador	Uso	Descripción
&&	expr1 && expr2	Devuelve <i>expr1</i> si puede ser convertido a <i>false</i> de lo contrario devuelve <i>expr2</i> . Por lo tanto, cuando se usa con valores booleanos, && devuelve <i>true</i> si ambos operandos son <i>true</i> , en caso contrario devuelve <i>false</i> .
	expr1 expr2	Devuelve <i>expr1</i> si puede ser convertido a <i>true</i> de lo contrario devuelve <i>expr2</i> . Por lo tanto, cuando se usa con valores booleanos, devuelve <i>true</i> si alguno de los operandos es <i>true</i> , o <i>false</i> si ambos son <i>false</i> .
!	!expr	Devuelve <i>false</i> si el operando puede ser convertido a <i>true</i> ; en caso contrario devuelve <i>true</i>

Ejemplos de expresiones que pueden ser convertidas a *false* son aquellas que pueden ser evaluadas como *null*, *0*, *NaN*, *undefined* o una cadena vacía.

El siguiente código muestra ejemplos del operador && (AND Lógico):

```
var a1 = true && true;    // t && t devuelve true
var a2 = true && false;   // t && f devuelve false
var a3 = false && true;    // f && t devuelve false
var a4 = false && (3 == 4); // f && f devuelve false
var a5 = "Cat" && "Dog";   // t && t devuelve "Dog"
var a6 = false && "Cat";   // f && t devuelve false
var a7 = "Cat" && false;   // t && f devuelve false
```

El siguiente código muestra ejemplos del operador || (OR Lógico).

```
var o1 = true || true;    // t || t devuelve true
var o2 = false || true;   // f || t devuelve true
var o3 = true || false;   // t || f devuelve true
var o4 = false || (3 == 4); // f || f devuelve false
var o5 = "Cat" || "Dog";   // t || t devuelve "Cat"
var o6 = false || "Cat";   // f || t devuelve "Cat"
var o7 = "Cat" || false;   // t || f devuelve "Cat"
```

El siguiente código muestra ejemplos del operador ! (NOT Lógico).

```
var n1 = !true;    // !t devuelve false
var n2 = !false;   // !f devuelve true
var n3 = !"Cat";   // !t devuelve false
```

Operadores de tipos

Operador	Descripción
typeof	Devuelve una cadena de caracteres indicando el tipo del operando evaluado
instanceof	Devuelve <i>true</i> si el objeto especificado como primer operando es del tipo de objeto especificado como segundo operando.

Ejemplos de uso de *typeof*:

```
var forma = "redonda";
var largo = 1;
var hoy = new Date();
typeof forma;    // devuelve "string"
typeof largo;    // devuelve "number"
typeof hoy;      // devuelve "object"
typeof noExiste; // devuelve "undefined"
```

Sintaxis de uso de *instanceof*:

```
nombreObjeto instanceof tipoObjeto
```

Precedencia de los operadores

De **mayor a menor** precedencia:

. []
() new
! ~ - + ++ -- typeof void delete
* / %
+ -
<< >> >>>
< <= > >= in instanceof
== != === !==
&
^
&&
?:
= += -= *= /= %= <<= >>= >>>= &= ^= =
,

Actividad 3: Diseña un script dónde se declaren variables que almacenen tipos de datos distintos: number, string, boolean y undefined; a continuación, debe mostrar por consola del navegador, el tipo de cada uno de ellos. El script se localizará en una página HTML

EJERCICIOS PROPUESTOS:

Antes de continuar trabajando con este manual, realiza los **ejercicios** propuestos que encontrarás en el archivo **UT2_Ejercicios_JS_1.pdf** localizado en la subsección "actividades" de la sección "UT2"

3. Estructuras de control en JS

3.1. Estructuras condicionales

Una sentencia condicional es un conjunto de instrucciones que se ejecutan si una condición es verdadera. JavaScript soporta dos sentencias condicionales: **if...else** y **switch**

Estructura condicional if.. else

Sintaxis:

```
if (condición) {sentencia_1;...} else { sentencia_2;....}
```

La condición puede ser cualquier expresión que se evalúa a *true* o *false*. Recuerda que, además de *false*, en JS otros valores equivalen a falso: *undefined*, *null*, *0*, *NaN*, la cadena vacía ("")

Pueden existir estructuras *if... else* anidadas. Por ejemplo:

```
if (condición_1) {  
    sentencia_1;  
} else if (condición_2) {  
    sentencia_2;  
} else if (condición_n) {  
    sentencia_n;  
} else {  
    ultima_sentencia;  
}
```

Estructura condicional switch

Una sentencia *switch* permite a un programa evaluar una expresión e intentar igualar el valor de dicha expresión con un valor (*case*). Si se encuentra una coincidencia, el programa ejecuta la sentencia asociada. Sintaxis:

```
switch (expresión) {  
    case etiqueta_1:  
        sentencias_1
```

```
[break;]
case etiqueta_2:
  sentencias_2
  [break;]
...
default:
  sentencias_por_defecto
  [break;]
}
```

El programa primero busca una cláusula *case* con una etiqueta que coincida con el valor de la expresión *y*, entonces, transfiere el control a esa cláusula, ejecutando las instrucciones asociadas a ella. Si no se encuentran etiquetas coincidentes, el programa busca la cláusula opcional *default* y, si se encuentra, transfiere el control a esa cláusula, ejecutando las instrucciones asociadas. Si no se encuentra la cláusula *default*, el programa continúa su ejecución por la siguiente instrucción al final del *switch*. Por convención, la cláusula por defecto es la última cláusula, aunque no es necesario que sea así.

La instrucción opcional *break* asociada con cada cláusula *case* asegura que el programa finaliza la sentencia *switch* una vez que la instrucción asociada a la etiqueta coincidente se ejecuta y continúa la ejecución por las sentencias siguientes a *switch*. Si se omite la sentencia *break*, el programa continúa su ejecución por la siguiente instrucción que encuentre en secuencia hasta llegar a *break* o al final de *switch*.

Actividad 4: Diseña un script que genere dos números enteros al azar, comprendidos entre 1 y 100, y los muestre por consola, indicando cuál es el mayor, o bien si son iguales.

- ✓ El método *random()* del objeto *Math* devuelve un número aleatorio comprendido entre 0 y 1: *Math.random()*
- ✓ El método *trunc()* del objeto *Math* devuelve la parte entera del número *x*: *Math.trunc(x)*
 - El método *trunc()* se incluye en *Math* a partir de **ES6**
- ✓ Se puede mostrar información por consola con la instrucción *console.log(mensaje)*

Actividad 5: Diseña un script que genere un número al azar comprendido entre 1 y 12 y que muestre, en la consola del navegador, el número de días que corresponde a dicho mes. Utiliza una sentencia *switch* en la solución.

NOTA: Utiliza únicamente los contenidos de esta unidad de trabajo y la funcionalidad proporcionada hasta el momento para dar solución a los ejercicios y actividades propuestos.

3.2. Estructuras repetitivas (bucles)

Los bucles son estructuras de control que repiten la ejecución de un conjunto de instrucciones varias veces.

Bucles en Javascript: *while*, *do... while*, *for*, *for ... in*

Bucle *while*

Ejecuta el cuerpo del bucle mientras que la condición es cierta. Sintaxis:

```
while (condicion){  
  cuerpo del bucle  
}
```

Bucle *do...while*

Ejecuta el cuerpo del bucle mientras que la condición es cierta. Sintaxis:

```
do{  
  cuerpo del bucle  
}while (condicion);
```

Bucle *for*

Ejecuta el cuerpo del bucle mientras que la condición es cierta. Sintaxis:

```
for ([expresionInicial]; [condicion]; [expresionIncremento]){  
  cuerpo del bucle  
}
```

Funcionamiento:

1. La expresión de inicialización *expresionInicial*, si existe, se ejecuta. Esta expresión habitualmente inicializa uno o más contadores del bucle, pero la sintaxis permite una expresión con cualquier grado de complejidad. Esta expresión puede también declarar variables.
2. Se evalúa la expresión *condicion*. Si el valor de *condicion* es *true*, se ejecuta el cuerpo del bucle. Si el valor de *condicion* es *false*, el bucle *for* finaliza. Si la expresión *condicion* es omitida, la condición es asumida como verdadera.
3. Se ejecuta el cuerpo del bucle.
4. Se ejecuta la expresión *expresionIncremento*, si existe, y el control vuelve al paso 2.

Bucle *for... in*

Itera sobre las propiedades de un objeto. Sintaxis:

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

Bucle *for... of*

La sentencia **for...of** crea un bucle que itera a través de los elementos de los objetos iterables: *Array, String, Map, Set, DOM data structures*.

Se estudiará más adelante.

EJERCICIOS PROPUESTOS:

Realiza los **ejercicios** propuestos que encontrarás en el archivo **UT2_Ejercicios_JS_II.pdf** localizado en la subsección "actividades" de la sección "UT2"

4. Referencias bibliográficas

- <https://desarrolloweb.com/javascript/>
- <https://desarrolloweb.com/manuales/manual-javascript.html>
- <https://www.w3schools.com/js/default.asp>
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions and Operators](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators)
- [https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Control de flujo y manejo de errores](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Control_de_flujo_y_manejo_de_errores)
- <https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia>
- <https://www.sublimetext.com/download>