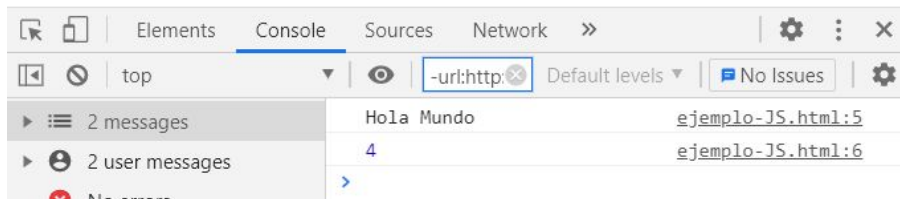


La consola de JavaScript

Para acceder a la consola Javascript del navegador pulsamos **CTRL+SHIFT+J**.

Un clásico ejemplo utilizado cuando se comienza a programar es crear un programa que muestre por pantalla un texto, generalmente el texto «*Hola Mundo*». O mostrar el resultado de alguna operación matemática. A continuación, el código JS para realizar ambas tareas, y la salida que podemos ver en la consola del navegador:

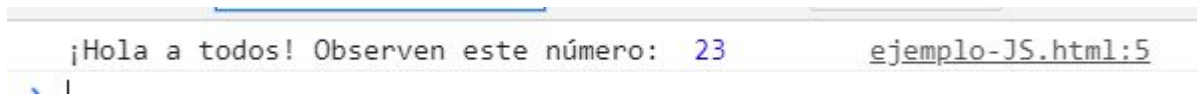
```
console.log("Hola Mundo");  
console.log(2 + 2);
```



La consola de JavaScript

Podemos mostrar texto, valores numéricos, etc. separados por comas:

```
console.log("¡Hola a todos! Observen este número: ", 5 + 18);
```



En esta consola podemos escribir **funciones** o **sentencias** de JavaScript que se ejecutan en la página que se encuentra en la pestaña actual del navegador. De esta forma podemos observar los resultados que nos devuelve en la consola al realizar diferentes acciones.

La consola de JavaScript

JS posee, además de **console.log**, varias instrucciones similares para interactuar con el desarrollador:

Función	Descripción
console.log()	Muestra la información proporcionada en la consola Javascript.
console.info()	Equivalente al anterior. Se utiliza para mensajes de información.
console.warn()	Muestra información de advertencia. Aparece en amarillo.
console.error()	Muestra información de error. Aparece en rojo.
console.clear()	Limpia la consola. Equivalente a pulsar <code>CTRL</code> + <code>L</code> o escribir <code>clear()</code> .

Texto

Info



Warning



Error

Incorporando un archivo externo

Podemos vincular al documento HTML un archivo con **extensión .js** usando la etiqueta `<script>`, haciendo referencia al nombre del archivo **JavaScript** con el atributo **src** (source):

```
<html>
  <head>
    <title>Título de la página</title>
    <script src="index.js"></script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

Los archivos .js se suelen incorporar en una carpeta llamada "js".

Comentarios

Los comentarios son utilizados por los programadores para anotaciones. No son tenidos en cuenta por el navegador.

Comentario de línea

```
// esto es un comentario de línea
```

Comentario de bloque

```
/*  
esto es un comentario de bloque (multilínea)  
*/
```

Son un buen recurso cuando queremos omitir la ejecución de ciertas instrucciones.

Variables | ¿Qué son?

Es un pequeño espacio en la memoria, donde se guarda un dato. Podemos imaginarlas como “cajitas” dentro de nuestro programa. Tienen tres características:

- **Nombre:** debe ser representativo de la información que contiene. Se utiliza para diferenciar unas de otras y hacer referencia a ellas.
- **Tipo de dato:** puede ser número, texto, valores booleanos, etc.
- **Contenido:** el valor concreto que posee el dato almacenado.

Se llaman **variables** porque pueden cambiar su valor a lo largo del programa. Un programa puede tener muchas variables, y cada una de ellas tendrá un nombre que la identifique, un valor y un tipo de dato.

Variables | ¿Cómo se declaran?

Una variable que se ha declarado con **var** pero a la que no se le asignó un valor se dice que está indefinida (no conocemos el tipo de dato):

```
var num3;
```

En este caso la variable está “vacía”, no está definido el valor que colocará en memoria. No se ha asociado ningún contenido a esa variable.

```
var num4 = 5;
```

Las sentencias en JS finalizan con “;”. La imagen anterior corresponde a la declaración de la variable “**num4**” con un valor numérico entero de “**5**”.

Variables | ¿Cómo se nombran?

Los nombres de las variables (o identificadores) permiten distinguir una de otras. Para asignar los nombres de las variables debemos seguir ciertas reglas:

Un identificador de JavaScript debe comenzar con una letra, un guión bajo (`_`) o un signo de dólar (`$`). Los siguientes caracteres también pueden ser dígitos (`0 - 9`). JavaScript distingue entre mayúsculas y minúsculas (es *case-sensitive*).

Se recomienda usar la escritura camelCase en el nombre de variables que tienen más de una palabra.

Variables

Podemos cambiar el valor de una variable durante el flujo del programa:

```
var IVA= 21;  
IVA= 10.5;  
console.log(IVA);
```

El `=` es el operador de asignación, y permite asignar un valor a una variable. Ese valor puede ser el resultado de una operación aritmética, que se evalúa y luego se asigna su resultado a la variable:

```
var resultado= (1 + 3) * 2;
```

Luego de ejecutar esa línea, la variable “resultado” contiene el valor “8”.

Constantes

El concepto de **constante** es similar al de **variable**, con la salvedad de que la información que contiene es siempre la misma (no puede variar durante el flujo del programa). Declaramos las constantes utilizando **const**. Su sintaxis es:

```
const PI= 3.141592;  
const IVA= 21;
```

Si intentamos modificar el valor de una constante, obtenemos un error:

```
const IVA= 21;  
IVA= 10.5;  
console.log(IVA);
```

```
✖ Uncaught TypeError: Assignment to constant variable.  
  at ejemplo-JS.html:9
```

Tipos de datos

Las variables de JavaScript pueden contener distintos tipos de datos: numérico, cadena de caracteres, lógicos, indefinido, null, objetos y más. El tipo de dato es la **naturaleza del contenido** de la variable o constante. JavaScript tiene **tipado dinámico**, es decir que la misma variable se puede utilizar para contener diferentes tipos de datos:

```
var x;      // ahora x es indefinido (no tiene un valor definido)
x = 5;      // ahora es numérico (5)
x = "Juan"; // ahora es una cadena de caracteres o string ("Juan")
```

JavaScript *deduce* cuál es el tipo de dato de la variable. El tipo de dato asociado a esa variable lo determina el dato que se almacena en ella. Y si luego se le asigna un valor de otro tipo, el tipo de la variable cambia.

Tipos de datos

Los tipos de datos en JavaScript son los siguientes:

Tipos de
datos
primitivos

Tipo de dato	Descripción	Ejemplo básico
NUMBER number	Valor numérico (enteros, decimales, etc...)	42
STRING string	Valor de texto (cadenas de texto, caracteres, etc...)	'MZ'
BOOLEAN boolean	Valor booleano (valores verdadero o falso)	true
UNDEFINED undefined	Valor sin definir (variable sin inicializar)	undefined
FUNCTION function	Función (función guardada en una variable)	function() {}
OBJECT object	Objeto (estructura más compleja)	{}

Tipos de datos

El último estándar **ECMAScript** define nueve tipos de datos:

- Seis tipos de datos primitivos [+info](#)
 - Undefined [+info](#)
 - Boolean [+info](#)
 - Number [+info](#)
 - String [+info](#)
 - BigInt [+info](#)
 - Symbol [+info](#)
- Null (tipo primitivo especial) [+info](#)
- Object [+info](#)
- Function [+info](#)

Identificar el tipo de dato de una variable

Para determinar qué tipo de dato tiene una variable utilizamos **typeof()**, que devuelve el tipo de dato primitivo asociado a una variable:

```
var s = "Hola, me llamo Juan"; // s, de string  
var n = 28; // n, de número  
var b = true; // b, de booleano  
var u; // u, de undefined
```

```
console.log(typeof s);  
console.log(typeof n);  
console.log(typeof b);  
console.log(typeof u);
```

string
number
boolean
undefined

Las variables numéricas

En JavaScript, los **números** constituyen un tipo de datos básico (primitivo). Para crear una variable numérica basta con escribirlas. No obstante, dado que en Javascript “todo es un objeto”, también podemos declararlas como si fuesen un objeto:

Constructor	Descripción
<small>NUMBER</small> <code>new Number(n)</code>	Crea un objeto numérico a partir del número <code>n</code> pasado por parámetro.
<small>NUMBER</small> <code>n</code>	Simplemente, el número en cuestión. Notación preferida.

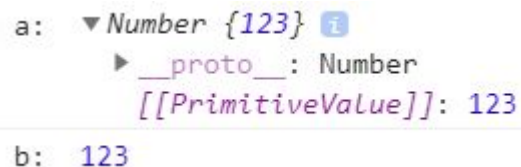
```
// Declarados como literales  
const n1 = 4;  
var n2 = 15.8;
```

```
// Declarados como objetos  
const n1 = new Number(4);  
var n2 = new Number(15.8);
```

Objeto Number

Number es el objeto primitivo que permite representar y manipular valores numéricos. El constructor *Number* contiene constantes y métodos para trabajar con números. Valores de otro tipo pueden ser convertidos a números usando la función **Number()**. Su sintaxis es:

```
var a = new Number('123'); // a es igual a 123
var b = Number('123'); // b es igual a 123
console.log("a: ", a);
console.log("b: ", b);
```



The screenshot shows a web browser's developer console. The variable 'a' is expanded, showing its prototype chain: `__proto__: Number` and `[[PrimitiveValue]]: 123`. The variable 'b' is shown with the value `123`.

```
a: ▼ Number {123} ⓘ
  __proto__: Number
  [[PrimitiveValue]]: 123
b: 123
```

Creamos el objeto **a** mediante el constructor y guardamos en **b** el valor de la cadena '123' en forma de número. Mostramos en consola ambos elementos.

Comprobaciones numéricas

Varias funciones de JS permiten conocer la naturaleza de una variable numérica (número finito, número entero, número seguro o si no es representable como un número). Devuelven *true* o *false* (un valor booleano). Las podemos ver en la siguiente tabla:

Método	Descripción
BOOLEAN Number.isFinite(<i>n</i>)	Comprueba si <i>n</i> es un número finito.
BOOLEAN Number.isInteger(<i>n</i>)	Comprueba si <i>n</i> es un número entero.
BOOLEAN Number.isSafeInteger(<i>n</i>)	Comprueba si <i>n</i> es un número seguro.
BOOLEAN Number.isNaN(<i>n</i>)	Comprueba si <i>n</i> no es un número.

Comprobaciones numéricas

Veamos dos ejemplos para cada una de estas funciones:

```
// ¿Número finito?  
Number.isFinite(42); // true  
Number.isFinite(Infinity); // false, es infinito  
// ¿Número entero?  
Number.isInteger(5); // true  
Number.isInteger(4.6); // false, es decimal  
// ¿Número seguro?  
Number.isSafeInteger(1e15); // true  
Number.isSafeInteger(1e16); // false, es un valor  
no seguro  
// ¿No es un número?  
Number.isNaN(NaN); // true  
Number.isNaN(5); // false, es un número
```

Numero finito (42):	true
Numero finito (infinito):	false
Numero entero (5):	true
Numero entero (4.6):	false
Numero seguro (1e15):	true
Numero seguro (1e16):	false
Not a Number (NaN):	true
Not a Number (5):	false

Conversión numérica

Es posible convertir cadenas de texto en números, para posteriormente realizar operaciones con ellos. Las funciones de parseo numérico, **parseInt()** y **parseFloat()**, permiten realizar esto:

Método	Descripción
<small>NUMBER</small> Number.parseInt(S)	Convierte una cadena de texto S en un número entero.
<small>NUMBER</small> Number.parseInt(S, radix)	Idem al anterior, pero desde una base radix .
<small>NUMBER</small> Number.parseFloat(S)	Convierte una cadena de texto S en un número decimal.
<small>NUMBER</small> Number.parseFloat(S, radix)	Idem al anterior, pero desde una base radix .

Conversión numérica

Veamos un ejemplo con **parseInt()**. Recibe como parámetro un texto que queremos convertir a número:

```
Number.parseInt("42"); // 42
Number.parseInt("42€"); // 42
Number.parseInt("Núm. 42"); // NaN
Number.parseInt("A"); // NaN
```

```
parseInt (42) 42
```

```
parseInt (42$) 42
```

```
parseInt (Num. 42) NaN
```

```
parseInt (A) NaN
```

parseInt() funciona con variables de texto que contienen números o que comienzan por números. Sin embargo, si la variable de texto comienza por un valor que no es numérico, **parseInt()** devuelve un **NaN** (*Not a Number*).

Conversión numérica

Si utilizamos **parseInt()** con dos parámetros, donde el primero es el texto con el número y el segundo es la base numérica del número, se realiza la conversión de tipo respetando la base elegida:

```
Number.parseInt("11101", 2); // 29 en binario  
Number.parseInt("31", 8); // 25 en octal  
Number.parseInt("FF", 16); // 255 en hexadecimal
```

```
parseInt (11101, 2 (binario)) 29  
parseInt (31, 8 (octal)) 25  
parseInt (FF, 16 (hexadecimal)) 255
```

Esta modalidad de **parseInt()** se utiliza para pasar a base decimal un número que se encuentra en otra base (binario, octal, hexadecimal, etc.) **parseFloat()** funciona exactamente igual, pero en lugar de operar con números enteros opera con números en coma flotante.

Operadores aritméticos y de asignación

El **operador de asignación (=)** le otorga un valor a una variable y se coloca entre la variable y el valor a asignar.

```
var x = 10;
```

Los operadores aritméticos que vemos a la derecha se utilizan para realizar operaciones aritméticas en números:

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
**	Exponenciación
/	División
%	Módulo: resto de dividir
++	Incremento
--	Decremento

Operadores de cadena y números

Los operadores `+` y `+=` también se pueden utilizar para agregar (concatenar) cadenas. En este contexto, **el operador `+`** se denomina **operador de concatenación**.

```
var txt1 = "Juan";  
var txt2 = "Pablo";  
var txt3 = txt1 + " " + txt2;  
console.log(txt3);
```

Juan Pablo

```
var txt4 = "Bienvenidos ";  
txt4 += "a Javascript";  
console.log(txt4);
```

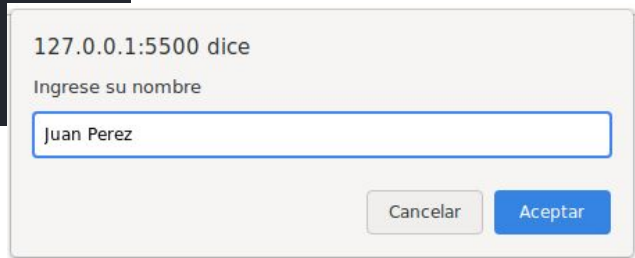
Bienvenidos a Javascript

Agregar dos números devolverá la suma, pero agregar un número y una cadena devolverá una cadena.

La función prompt()

La función **prompt** es un **método** del objeto **Window**. Se utiliza para solicitarle al usuario que ingrese datos por medio del teclado. Recibe dos parámetros: el mensaje que se muestra en la ventana y el valor inicial del área de texto. Su sintaxis es: `variable = prompt(mensaje, valor inicial)`

```
<script>
  var nombre = prompt ("Ingrese su nombre", "")
  document.write( "Hola " + nombre)
</script>
```



Uso de document.write()

document.write() nos permite escribir directamente dentro del propio documento HTML.

```
<html>
  <head>
    <title>Título de la página</title>
    <script>
      document.write("Hola mundo (HTML)");
    </script>
  </head>
  <body>
  </body>
</html>
```



Material extra

Artículos de interés

Documentación extra:

- [¿Qué es JavaScript?](#)
- [¿Qué es EcmaScript?](#)
- [¿Debo usar “;” en Javascript?](#)
- [Tipos de datos en JavaScript](#)
- [Variables en JavaScript](#)
- El [objeto Number](#) en JavaScript
- Métodos del objeto Math en [Developer Mozilla](#), [W3Schools](#) y en [LenguajeJS](#)

Video:

- [Introducción a JavaScript](#)