Unión Europea

Fondo Social Europeo
"El FSE invierte en tu futuro"

2º DAW

DESARROLLO WEB EN ENTORNO CLIENTE

UT 2: JavaScript. Manejo de la sintaxis.

Profesora: Irene Rodil Jiménez

0. Contenidos

1. Características de JavaScript

HISTORIA

- ***Tim Berners-Lee*** inventó la *World Wide Web* en 1989 y lanzó el primer navegador web del mundo *WorldWideWeb* en 1991. Posteriormente cambió el nombre del navegador a ***Nexus*** para no llevar a confusión.
- En los años 90 la navegación era lenta y comenzaron a surgir aplicaciones web cada vez más complejas → si el usuario no rellenaba correctamente un formulario, tenía que esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.
- ***Brendan Eich***, un programador que trabajaba en Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como *ScriptEase*) al navegador Netscape. En 10 días creo el primer lenguaje de scripting para la Web que primero se llamó ***Mocha*** y después ***LiveScript***.
- En 1995 con el lanzamiento de la versión 2 del navegador Netscape se incluyó el lenguaje de scripting pero con el nombre de ***JavaScript***. Este cambio se debió a la alianza de *Netscape* con *Sun Microsystems* (desarrollador de Java) por razones de marketing (Java era muy popular).

1. Características de JavaScript

HISTORIA

- En 1996 *Microsoft* tenía su lenguaje de scripting basado en el lenguaje *Visual Basic*: ***VBScript*** (*Visual Basic Script Edition*). Pero dado el éxito de JavaScript a finales de 1996 Microsoft lanzó también su propia versión de JavaScript, ***JScript*** (le cambiaron el nombre para evitar problemas legales), integrándolo en sus navegadores a partir de IE 3.
- Aunque las diferencias entre *JavaScript* y *JScript* no eran muy grandes, como cada navegador soportaba solo uno de ellos, obligó a los desarrolladores a programar dos veces la misma funcionalidad (una con cada lenguaje) e introducir en sus página web sentencias condicionales para distinguir en qué navegador se estaba ejecutando.
- Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje → en 1997 envió la especificación JavaScript 1.1 al organismo ***ECMA*** (*European Computer Manufacturers Association*) que estandarizó el lenguaje con el nombre de ***ECMAScript*** o ***ECMA-262***.
  - Versiones: *JavaScript*, *Jscript*, *TypeScript*
- En Junio de 1998 la organización internacional para la estandarización (***ISO***) adoptó el estándar ECMA-262 con el nombre ISO/IEC-16262.

1. Características de JavaScript

ECMAScript

Versión ECMAScript	Año
ES1	Junio 1997
ES2	Agosto 1998
ES3	Diciembre 1999
ES5	Diciembre 2009
ES 5.1	Junio 2011
ES6 / 2015 / Harmony JavaScript moderno	Junio 2015
ES7 / 2016	2016
ES8 / 2017	2017
ES9 / 2018	2018
ES10 / 2019	2019
ES11 / 2020	2020
ES12 / 2021	2021
ES13 / 2022	2022

1. Características de JavaScript

¿QUÉ ES JAVASCRIPT?

- Lenguaje de programación creado originalmente por ***Brendan Eich*** para el navegador Netscape.
- Es un lenguaje interpretado utilizado fundamentalmente para dotar de comportamiento dinámico a las páginas web. Para interactuar con una página web se provee al lenguaje de una implementación del ***Document Object Model*** (***DOM***).
- Es una marca registrada originariamente por la empresa *Sun Microsystems* (ahora *Oracle Corporation*).
- Se encuentra oficialmente bajo la organización de *Mozilla Foundation*, y periódicamente se añaden nuevas características del lenguaje.
- Actualmente es un dialecto del estándar ECMAScript.
- Desde 2012, cualquier navegador web actual incorpora un intérprete para código JavaScript.
- Se utiliza principalmente en su forma del lado del cliente (a través de un navegador) aunque también podemos utilizar JavaScript en el lado del servidor (utilizando node.js) que está teniendo auge en los últimos años.

1. Características de JavaScript

¿QUÉ ES JAVASCRIPT?

- **Características**:
  - Es interpretado → no hay que compilar los programas para ejecutarlos → se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. El navegador se encarga de interpretar y ejecutar el código.
  - Su sintaxis se asemeja a la de C++ y Java → se basa en el concepto de objeto, aunque no es un lenguaje orientado a objetos.
  - JavaScript está formado casi en su totalidad por objetos.
  - Los objetos utilizan herencia basada en prototipos. Los objetos heredan propiedades directamente de otros objetos sin necesidad de que sean instancias de una misma clase.
  - Es un lenguaje con tipado dinámico → permite que una variable pueda contener valores de distintos tipos en diferentes momentos de la ejecución. La ventaja es que se programa más rápido, y la desventaja es que muchos errores no los puede detectar el compilador y aparecen en tiempo de ejecución.
  - W3C (el consorcio responsable de la especificación del lenguaje HTML) lo ha elegido como estándar para HTML5.

1. Características de JavaScript

SEGURIDAD EN JAVASCRIPT

- Por seguridad, los scripts solo se pueden ejecutar dentro del navegador y con ciertas limitaciones:
  - No pueden comunicarse con recursos que no pertenezcan al mismo dominio desde el que se descargó el script.
  - No pueden cerrar ventanas que no hayan abierto esos mismos scripts.
  - No pueden acceder al sistema de ficheros, ni para leer ni para escribir.
  - No pueden acceder a las preferencias del navegador.
  - Si la ejecución de un script dura demasiado tiempo el navegador informa al usuario de que el script está consumiendo demasiados recursos y le da la posibilidad de detener su ejecución.
    - Esto podría ocurrir por un error de programación del script o por un script malicioso.

1. Características de JavaScript

FUNCIONALIDADES

- Lenguaje interpretado en el navegador: puede estar deshabilitado.
- No puede escribir ficheros en el servidor.
- Reacciona a la interacción del usuario.
- Controla múltiples ventanas, marcos, plugins, applets...
- Pre-procesa datos en el cliente.
- Modifica estilos y contenido de navegadores.
- Puede solicitar ficheros al servidor.

FUNCIONALIDADES

1. Características de JavaScript

HITOS EN EL DESARROLLO DE JAVASCRIPT

- **Inicios (1996-1997)**:
  - Automatizar la creación de código HTML. Ejm: generar un calendario mensual mediante bucles en lugar de tener que escribir manualmente el código.
  - Responder a eventos realizados por el usuario sobre elementos de formularios, enlaces (*links*), e imágenes de un modo rudimentario. Ejm: permitiendo realizar validación de formularios o botones *rollover* (botones que cambian su aspecto al colocar el puntero del ratón sobre ellos).
- **DOM (Document Object Model) (1997-2004):**
  - El desarrollo de la especificación del DOM (por parte del W3C) permitía manipular cualquier elemento de una página web mediante JavaScript, incluso una vez cargada y presentada al usuario.
  - Mejoro las posibilidades de interacción con el usuario, y empezaron los efectos de animación.
- **AJAX (Asynchronous JavaScript and XML) (1999-…):**
  - El término «Ajax» fue creado en 2005 pero la tecnología es de finales de los 90. Permitía por primera vez intercambiar datos con el servidor sin tener que recargar una página web.

1. Características de JavaScript

HITOS EN EL DESARROLLO DE JAVASCRIPT

- **Bibliotecas (2005-…)**:
  - Muy importantes en el desarrollo de la web 2.0.
  - Crean una capa de abstracción entre el programador y el intérprete JavaScript del navegador → permiten despreocuparse de los detalles de más bajo nivel (gestión de incompatibilidades entre navegadores, creación de interfaces de usuarios ricas (autocompletar, arrastrar y soltar, eventos táctiles…), y la aplicación de efectos visuales atractivos (menús desplegables, *light box* (superposición de imágenes sobre la página web), …)) y mejorar la productividad.
- **HTML5 (2008-…)**: W3C eligió JavaScript como estándar de HTML5 →
  - **Canvas o lienzo**: capacidad para dibujar y crear animaciones en una página web.
  - **Audio y vídeo**: sin recurrir a plugins externos.
  - **Arrastrar y soltar**.
  - **Aplicaciones offline** o en caché.
  - **Almacenamiento web**: almacenamiento de datos en el lado del cliente, similar a las Cookies, pero con más capacidad.
  - **Geolocalización**: a través de un disparador de eventos.
  - **Notificaciones**: enviar mensajes de escritorio que se muestran de una forma similar a los avisos de la bandeja de iconos de Windows.

1. Características de JavaScript

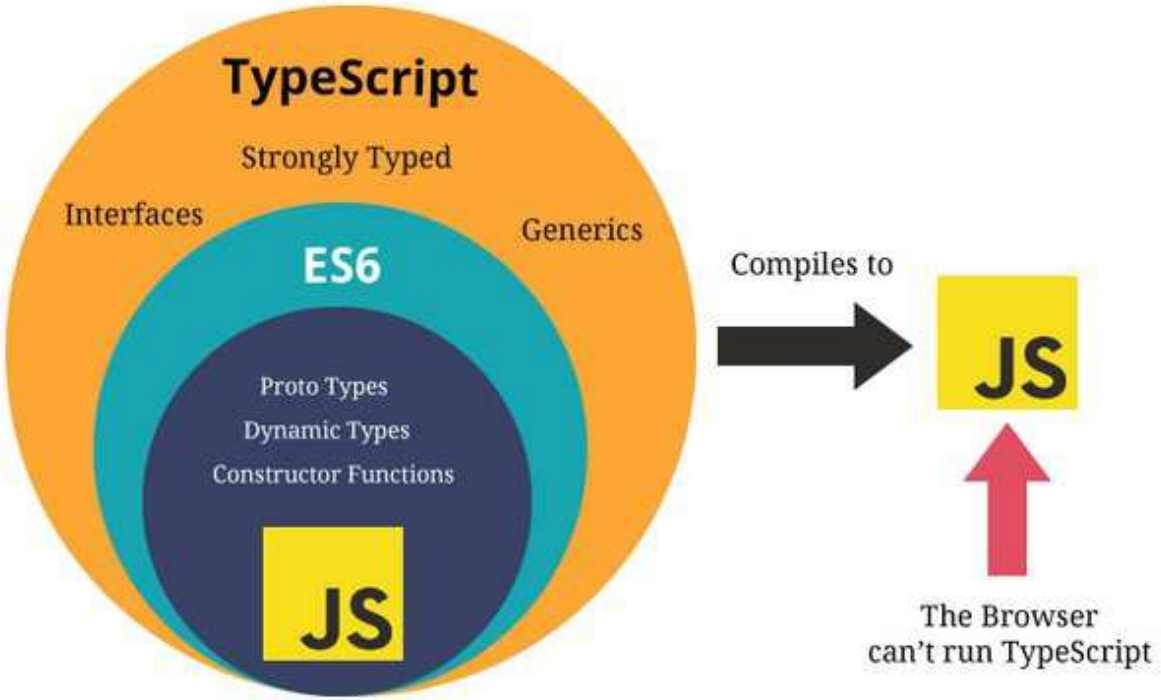
TECNOLOGÍA ALREDEDOR DE JAVASCRIPT

- 2002 **JSON**. Es un formato de texto ligero para el intercambio de datos. Una ventaja de JSON sobre XML es que es fácil utilizarlo en JavaScript con la función *eval*, aunque presenta problemas de seguridad.
- 2005 **Ajax** (Microsoft). Permite que el servidor y el navegador intercambien información de forma asíncrona.
- 2006 **jQuery**, simplificaba la utilización de *JavaScript*. Es una librería que permite agregar interactividad a un sitio web sin conocer JavaScript.
- 2008 Douglas Crockford escribe **The good parts** indicando que Javascript no era un lenguaje malo.
- 2009 **Node.js**. Permite ejecutar *JavaScript* en el servidor.
- 2010 **Angular**. Es un framework de *JavaScript* mantenido por *Google*. Extiende la sintaxis de HTML para darle más funcionalidad.
- 2013 **React**. Librería mantenida por Facebook, orientada a la visualización, no a la lógica. Principalmente en sitios web de alto tráfico.
- 2014 **Vue.js**. Es un framework de *JavaScript* que ayuda a relacionar la capa de presentación con la capa de negocio de forma sencilla y eficiente.

1. Características de JavaScript

TECNOLOGÍA ALREDEDOR DE JAVASCRIPT

- **TypeScript**
  - Desarrollado por Microsoft en 2012.
  - Es un superset de JavaScript (está construido por encima de Javascript).



2. Sintaxis básica

- La sintaxis de JavaScript es muy similar a la de Java o C++.
- La sintaxis especifica aspectos como: los caracteres que se deben utilizar para definir comentarios, la forma de los nombres de las variable o el modo de separar las diferentes instrucciones de código.

CASE SENSITIVE

- Distingue mayúsculas y minúsculas, a diferencia de HTML.
- No es lo mismo utilizar *alert()* que *Alert()*.
- **lowerCamelCase**: se suele utilizar este estilo de escritura:
  - Cuando una variable o función tiene más de una palabra se escriben con minúsculas la primera y las siguientes con la primera letra en mayúsculas.
  - Ejm: *ejemploDeLowerCamelCase*

TABULACIÓN Y SALTOS DE LÍNEA

- JavaScript ignora los espacios consecutivos y las tabulaciones.
- Emplear la tabulación y los saltos de línea mejora la presentación y la legibilidad del código.

2. Sintaxis básica

PUNTO Y COMA

- Es opcional. Algunas guías de estilo modernas recomiendan no usarlas.
- El ; separa expresiones → hace el mismo efecto que el salto de línea.
- Si queremos escribir dos expresiones en una misma línea su uso si es obligatorio.

COMENTARIOS

- Son líneas de código que no son interpretadas por el navegador.
- Su función principal es la de facilitar la lectura del código al programador.
- También se pueden utilizar para que no se ejecuten líneas de código cuando se está depurando el código.
- Dos formas de insertar comentarios:
  - Doble barra (//): permite comentar una sola línea de código
  - Signos /* al inicio del comentario y los signos */ al final del mismo. Esta técnica permite comentar una o varias líneas de código.

2. Sintaxis básica

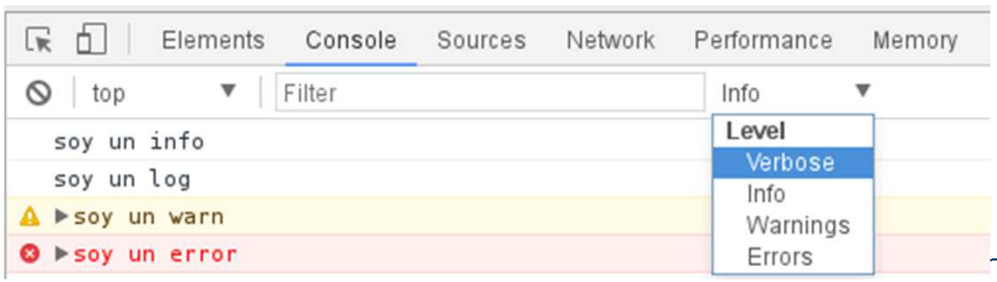
PALABRAS RESERVADAS

- Algunas palabras no se pueden utilizar para definir nombres de variables, funciones o etiquetas.
- Es aconsejable no utilizar tampoco las palabras reservadas para futuras versiones de JavaScript.
- En www.ecmascript.org se pueden consultar todas las palabras reservadas de JavaScript. **Ejm**: *break*, *case*, *class*, *catch*, *const*, *continue*, …

CONSOLE

- Permite escribir texto en la consola del navegador → solo será visible en modo desarrollador.
- Solo se debe utilizar en modo depuración de código.
- Hay distintos niveles que se puede filtrar para visualizar uno o varios.
- El más utilizado es **console.log**

```
1 console.info("soy un info");
2 console.log("soy un log");
3 console.warn("soy un warn");
4 console.error("soy un error");
```



2. Sintaxis básica

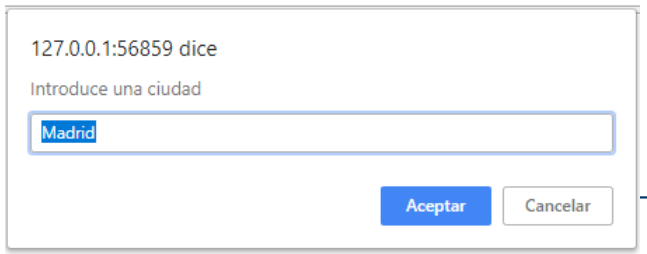
ALERT Y PROMPT

- Son dos funciones que muestran ventanas emergentes al usuario (*pop-ups*) → actualmente no son una buena práctica en JavaScript:
  - El uso de pop-ups se considera intrusivo, bloquean el acceso a otras partes de la página e incluso pueden estar bloqueadas por el usuario en su navegador y no mostrarse si no las acepta.
- Los vamos a utilizar únicamente en los primeros ejemplos para facilitar la interactividad, sin necesidad de complicar el código.
- **Alert**:
  - Muestra una ventana emergente con un texto y un botón ***Aceptar***.
  - **Sintaxis**: *alert("Texto a mostrar")*

```
<script>
   alert("Texto de alerta")
</script>
```

- **Prompt**:
  - Pide al usuario que introduzca un texto. Si el usuario *Cancela* devuelve ***null***.
  - **Sintaxis**: *prompt("Texto descriptivo","Valor por defecto")*
    El segundo parámetro es opcional

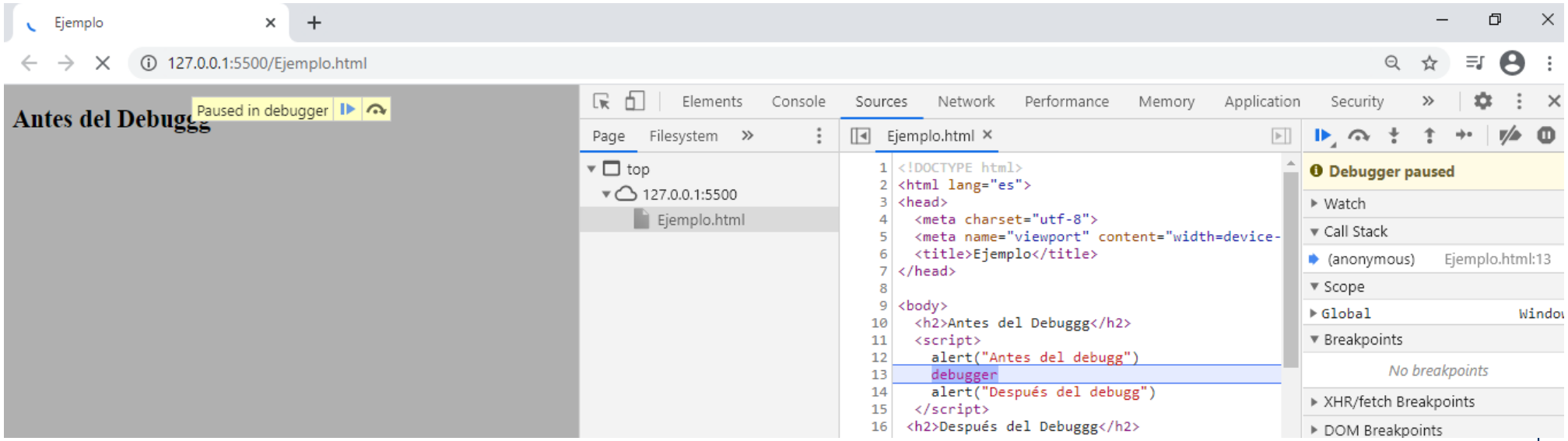
```
<script>
   let ciudad = prompt("Introduce una ciudad","Madrid")
</script>
```



2. Sintaxis básica

DEBUGGER

- Los navegadores web utilizan la palabra clave sentencia **debugger;** como punto de interrupción para iniciar el depurado de un código JavaScript.
- **Importante**: hay que tener la herramienta de depuración abierta (*F12*)
  - Se pueden poner expresiones de inspección, nuevos puntos de ruptura e ir paso a paso en la ejecución del código JavaScript.



2. Sintaxis básica

DOCUMENT.WRITE

- Con este método podemos mostrar un texto en el HTML en vez de en una ventana emergente.
- No es muy óptimo ya que al usar la etiqueta <script> para incluir el código se bloquea la construcción del DOM.
- **Nota**: si lo usamos cuando la página ya está cargada → borraremos todo el contenido HTML dejando solo el resultado de *document.write()*:
  - Ejemplo: si cargamos la página y se utiliza *document.write()* en un botón → mostrará el contenido introducido en *document.write()* reemplazando todo el texto que ya tuviéramos.

```
<button type="button" onclick="document.write('BORRADO')">Pulsa si tienes valor!</button>
```

- La alternativa, más utilizada, es manipular el contenido HTML de una página web utilizando *DOM* (*Modelo de Objetos del Documento*).

```
<body>
  <p>MI PAGINA</p>
  <p id="resultado"></p>
  <button type="button" onclick="document.getElementById('resultado').innerHTML='HE ESCRITO EN EL HTML'">
  Pulsa para escribir en el HTML</button>
</body>
```

3. Variables

- Las variables son la base de cualquier lenguaje de programación.
- Una variable es un contenedor para un dato que puede variar. Este dato se almacena en memoria y se puede utilizar todas las veces que se necesite.
- Se pueden definir como zonas de la memoria de un ordenador que se identifican con un nombre y en las cuales se almacenan ciertos datos.
- Deben tener un nombre descriptivo para facilitar la lectura del código del programa.
- Las variables pueden tener dos tipos de alcance (*scope*):
  - **Local**: cuando la variable se declara dentro de un bloque → solo se puede acceder a la variable en esta parte del código.
  - **Global**: cuando se declara fuera de un bloque. Se puede acceder a este tipo de variables desde cualquier parte del código, ya sea dentro o fuera del bloque. Las variables globales no son una buena práctica ya que tenemos la posibilidad de declarar dos variables globales en distintas partes del código con el mismo nombre sin notarlo.
- Por otro lado, tenemos las **constantes** que no pueden variar su valor. Se les asigna un valor que no puede ser modificado.

3. Variables

- **Consideraciones**:
  - **No requiere declaración del tipo** (es de "tipado débil"), e incluso permite que una variable almacene contenido de distintos tipos en distintos momentos.
  - JavaScript decide el tipo de la variable **por inferencia**.
  - El nombre de la variable no puede empezar por un número: obligatoriamente ha de empezar con una letra, un signo dólar o un guion bajo.
    - Nombrar las variables utilizando la notación camelCase
    - Nombrar las contantes en mayúsculas.
  - **Distingue entre mayúsculas y minúsculas**.
  - Permite hacer uso de una variable **sin que haya sido declarada**. Cuando JavaScript se encuentra una variable no declarada, crea automáticamente una variable global y permite su uso.
  - Una variable se inicializa cuando se establece su contenido o valor por primera vez.
  - Una variable se puede declarar e inicializar al mismo tiempo.
  - Se recomienda declarar las variables en la parte superior del ámbito en el que se van a utilizar.

3. Variables

- **Declaración de variables**:
  - Mediante la palabra clave ***var*** seguida por el nombre que se quiera dar a la variable. Es posible declarar más de una variable en una sola línea.

```
var miVariable
```

```
var miVariable1, miVariable2
```

    - Es local al ámbito donde se declara.
    - Si no declaramos y asignamos o llamamos directamente → **variable global**.
- Con ES6 se han introducido dos nuevas formas de declarar variables:
  - ***let*** para variables y ***const*** para constantes.
  - El alcance es local al bloque en el que se declara, con *var* solo a la función.
  - Si existe una colisión de nombres entre dos variables recibiremos un mensaje de error. En cambio con *var* se reescribían los valores de las variables → es más recomendable usar *let* y *const*.
  - Si se tiene una función dentro de otra, la función que está por fuera no tendrá acceso a las variables que se encuentran en la función de adentro, sin embargo, la función de adentro sí tendrá acceso a las variables que se encuentran declaradas en la función de afuera.
- **Asignación de un dato a una variable**: con el operador **=**

3. Variables

- **Ejemplos**:

```
let variable1  = "Está fuera del bloque";
if (true) {
  let variable1 = 'Está dentro del bloque'
  console.log(variable1) //Está dentro del bloque
}
console.log(variable1) //Está fuera del bloque
```

```
Console
top | Filter | Default levels | Group similar
Está dentro del bloque  ejemplo.js:4
Está fuera del bloque  ejemplo.js:6
```

```
var variable1  = "Está fuera del bloque"
if (true) {
  var variable1 = 'Está dentro del bloque'
  console.log(variable1) //Está dentro del bloque
}
console.log(variable1) //Está dentro del bloque
```

```
Console
top | Filter | Default levels | Group similar
Está dentro del bloque  ejemplo.js:4
Está dentro del bloque  ejemplo.js:6
```

```
let variable1  = "Está fuera del bloque"
if (true) {
  let variable1 = 'Está dentro del bloque'
  console.log(variable1) //Está dentro del bloque
}
let variable1  = "Misma variable declarada dos veces" //Error
console.log(variable1) //Está fuera del bloque
```

```
Console
top | Filter | Default levels | Group similar
Uncaught SyntaxError: Identifier 'variable1' has already been declared  ejemplo.js:6
```

```
function afuera() {
  let variableFuera = 'Esta variable está declarada fuera'

  function adentro() {
    let variableDentro = 'Esta variable está declarada dentro'
    console.log(variableFuera) //variable declarada fuera
  }
  adentro()
  console.log(variableDentro) //Error
}
afuera()
```

```
Console
top | Filter | Default levels | Group similar
Esta variable está declarada fuera  ejemplo.js:6
Uncaught ReferenceError: variableDentro is not defined  ejemplo.js:11
    at afuera (ejemplo.js:11)
    at ejemplo.js:14
```

4. Tipos de datos

- Especifican qué tipo de valor se guardará en una determinada variable.
- Los **tres tipos de datos primitivos** de JavaScript son:
  - **Números (*Number*):**
    - Existe sólo un tipo de dato numérico en notación decimal, el llamado ***double*** en los lenguajes Java o C++ → coma flotante de 64 bits (IEEE 754).
    - 0.1 + 0.2 == 0.3 se evalúa como false (0.1 no tiene una representación exacta en IEEE 754).
    - Comprendidos entre ***Number.MAX_VALUE*** y ***Number.MIN_VALUE***.
    - También permite representar números *hexadecimales* (prefijo **0x**), números *octales* (prefijo **0o**) y números *binarios* (prefijo **0b**) .
    - Y *notación científica* (**e**). **Ejm**: $10e2=10*10^2=1000$
    - ***NaN*** e ***Infinity***
  - **Cadenas de texto (*String*)**:
    - Tipo ***string*** que puede representar letras, dígitos, signos de puntuación o cualquier otro carácter de Unicode.
    - Se debe definir entre comillas dobles o comillas simples.
  - **Valores booleanos (*Boolean*)**:
    - Sólo admite dos valores: ***true*** o ***false***
    - Es muy útil a la hora de evaluar expresiones lógicas o verificar condiciones.

4. Tipos de datos

- Son también **datos primitivos**:
  - **undefined**
    - Se asigna implícitamente a todas las variables que han sido declaradas pero a las que no se ha asignado un dato.
    - También se devuelve este valor cuando se usa una propiedad de objeto que no existe.
    - Además de un tipo de datos, existe una variable global con ese nombre y también es un valor como tal. No es una palabra reservada del lenguaje.
  - **null**
    - No se asigna implícitamente como *undefined* → se asigna al declarar una variable que vamos a referir a un objeto → indica que es un objeto vacío.
    - Identifica a los punteros a objetos que aún no tienen un objeto asignado.
    - Se puede utilizar para borrar el contenido de una variable, sin eliminar la variable.
    - Si en la ventana *prompt* se pulsa el botón *Cancelar* devuelve el valor *null*.
    - Es un literal definido en la especificación del lenguaje → una palabra reservada del lenguaje.
    - No dispone de un tipo específico nos dice que es de tipo *Objeto*, cuando en realidad no lo es.

4. Tipos de datos

- **Array**:
  - Son un tipo especial de objeto que almacena un inventario de contenidos ordenados que pueden ser de cualquier tipo de variable (texto, números, booleanas, otras listas y objetos).
  - No se pueden sumar, restar o multiplicar. Para hacer operaciones entre contenidos de listas hay que llamar a los contenidos específicos de cada una de la lista con las que queramos operar.
  - El primer elemento de la lista es *0* y el último *lista.length-1*.

```
1  let frutas = ["Manzana", "Pera", "Plátano", "Sandía", "Melón"];
2
3  console.log(frutas)                    //devuelve ["Manzana", "Pera", "Plátano", "Sandía", "Melón"]
4  console.log(frutas.length)             //devuelve 5
5  console.log(frutas[1])                 //devuelve Pera
6  console.log(frutas[frutas.length-1])   //devuelve Melón
```

```
▼ (5) ["Manzana", "Pera", "Plátano", "Sandía", "Melón"]
    0: "Manzana"
    1: "Pera"
    2: "Plátano"
    3: "Sandía"
    4: "Melón"
    length: 5
  ▶ __proto__: Array(0)
```

4. Tipos de datos

- **Objetos**:
  - Es un tipo de datos compuesto → representa una colección de valores primitivos o de valores compuestos por otros objetos.
  - Pueden ser vistos como una colección de propiedades
  - JavaScript posee una serie de objetos predefinidos, denominados objetos del núcleo, como ***Number***, ***String***, ***Boolean*** → cuando en una expresión interviene un *number*, *string* o *boolean* se crea automáticamente para él un objeto de tipo *Number*, *String* o *Boolean*, respectivamente, de modo que podemos acceder a sus propiedades.
  - **Ejm**:

```
let nombre = "Irene"; console.log(nombre.length)
5
```

  - También se pueden crear nuevos objetos:
    - Las propiedades del objetos van encerradas en llaves {} y se separan con comas.

```
let profesor = {
  nombre: "Irene",
  apellido1: "Rodil",
  apellido2: "Jiménez",
  aficiones: ["html","css","JavaScript"]
}

console.log(profesor.nombre)                                  //devuelve Irene
console.log(profesor.apellido1 + " " + profesor.apellido2)    //devuelve Rodil Jiménez
console.log(profesor.aficiones[0])                            //devoelve html
console.log(profesor.aficiones[profesor.aficiones.length-1])  //devuelve JavaScript
```

4. Tipos de datos

- Secuencias de escape para cadenas de caracteres:

Secuencia de escape	Resultado
\\	Barra invertida
\'	Comilla simple
\"	Comillas dobles
\n	Salto de línea
\t	Tabulación horizontal
\v	Tabulación vertical
\f	Salto de página
\r	Retorno de carro
\b	Retroceso

5. Operadores

- JavaScript utiliza principalmente **cinco tipo de operadores**:
  - **Aritméticos.**
  - **Lógicos.**
  - **De asignación.**
  - **De comparación.**
  - **Condicionales.**
- Una expresión es cualquier unidad de código que se resuelve en un valor. Las expresiones más comunes son las expresiones matemáticas y las lógicas.
- Los elementos utilizados en una expresión y unidos a través de un operador se definen como operandos.
- **Operador binario**:
  - Ejm: 5 + 3

```
operando1 operador operando2
```

- **Operador unario**:
  - Ejm: x++ o ++x

```
operando operador
```

```
operador operando
```

5. Operadores

1. OPERADORES ARITMÉTICOS

- Permiten realizar cálculos elementales entre variables numéricas.

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

NaN (*Not a Number*): *alert*(15*"aa")

NaN es el único valor de JavaScript que no es igual a sí mismo: NaN==NaN → false

Método: Number.isNaN(15*"aa")

Infinity (infinito): *alert*(3/0)

Propiedad: *Number.POSITIVE_INFINITY*

Propiedad: *Number.NEGATIVE_INFINITY*

Método: *Number.isFinite*(3/0)

- Incremento y decremento es en una unidad y se puede utilizar el operador antes o después de la variable, aunque el efecto es diferente.

```
1  let num1 = 1
2  num1++
3  console.log('num1: '+num1) //num1: 2
```

```
1  let num1 = 1
2  let num2 = ++num1
3  console.log('num1: '+num1) //num1: 2
4  console.log('num2: '+num2) //num2: 2
```

```
1  let num1 = 1
2  let num2 = num1++
3  console.log('num1: '+num1) //num1: 2
4  console.log('num2: '+num2) //num2: 1
```

5. Operadores

2. OPERADORES LÓGICOS

- Combinan diferentes expresiones lógicas con el fin de evaluar si el resultado de dicha combinación es verdadero o falso.
- Generalmente, se utilizan cuando tenemos que tomar decisiones dentro de un programa.

Operador	Nombre
&&	Y
||	O
!	No

- && y || son de tipo cortocircuito (o perezoso) → si el operando de la izquierda es suficiente para conocer el resultado de la operación → no se evalúa el de la derecha. Por el contrario, si el operando de la izquierda no es suficiente para conocer el resultado de la operación se devuelve el de la derecha.
  - false && "casa" → false
  - true && "casa" → casa

  false || "casa" → casa
  true || "casa" → true

5. Operadores

3. OPERADORES DE ASIGNACIÓN

- El principal operador para realizar la asignación es el igual (=).
- **Otros operadores de asignación**:
  - Permiten obtener métodos abreviados para evitar escribir dos veces la variable que se encuentra a la izquierda del operador (al igual que los operadores de incremento y decremento) → ahorran tiempo y disminuyen la cantidad de código escrito.

Operador	Nombre
+=	Suma y asigna
-=	Resta y asigna
*=	Multiplica y asigna
/=	Divide y asigna
%=	Módulo y asigna

```
1  let deudas = 1300
2  deudas -=300 // deudas = deudas - 300
3  console.log('Deudas: '+deudas) //Deudas: 1000
```

5. Operadores

4. OPERADORES DE COMPARACIÓN

- Permiten comparar todo tipo de variables y devuelve un valor booleano.
- La comparación se puede realizar solo sobre números y cadenas → si utilizamos otro tipo de datos, estos se convertirán automáticamente para intentar que la comparación se pueda llevar a cabo.

Operador	Nombre
<	Menor que
<=	Menor o igual que
==	Igual
>	Mayor que
>=	Mayor o igual que
!=	Diferente
===	Estrictamente igual
!==	Estrictamente diferente

Estrictamente igual verifica que el operador a la izquierda del operador es igual y del mismo tipo de datos que el operador de la derecha

```
let A = ''  // empty string
let B = 0   // zero
let C = '0' // zero string

A == B // true - ok...
B == C // true - so far so good...
A == C // **FALSE** - Plot twist!
```

5. Operadores

5. OPERADORES CONDICIONALES

- Permite indicar al navegador que ejecute una acción en concreto después de evaluar una expresión.
- Consta de tres partes:
  - Expresión a evaluar
  - **?** Acción a realizar si la expresión es verdadera
  - **:** Acción a realizar si la expresión es falsa

```
let dividendo = prompt("Introduce el dividendo: ")
let divisor = prompt("Introduce el divisor: ")
let resultado
divisor!=0 ? resultado = dividendo/divisor : alert("No es posible la división por cero")
alert("El resultado es " + resultado) // si se divide por cero da como resultado undefined
```

```
let dividendo = prompt("Introduce el dividendo: ")
let divisor = prompt("Introduce el divisor: ")
let resultado
divisor!=0 ? resultado = dividendo/divisor : alert("No es posible la división por cero");resultado = 0
alert("El resultado es " + resultado)
```

5. Operadores

OTROS OPERADORES

Operador	Nombre
typeof	Tipo de (calcula el tipo de dato) Devuelve una cadena de texto
+	Concatenación de cadenas
( )	Agrupación de operaciones

PRECEDENCIA DE LOS OPERADORES

- De mayor a menor:

```
()
++ -- !
* / %
+ -
< > <= >=
== !=
&&
||
= += -= *= /= %=
```

5. Operadores

CONVERSIÓN DE TIPOS

- En JavaScript si hay **conversión automática** de tipos:
  - **true** se convierte en 1 y **false** en 0 antes de intervenir en una comparación.
  - Al comparar una cadena con un número, la cadena intentará convertirse en un número expresado en decimal; si no es posible se convertirá en ***NaN***. Si es una cadena vacía se interpreta como 0.
  - Cuando uno de los operandos del operador **+** es una cadena, el otro se convertirá también en una cadena, esto no pasa con otros operadores.
  - +"cadena": lo intenta convertir a número, si no puede devuelve *NaN*. +"" → 0
  - **Ejm**:
    - "1" == true → true (1 == 1)
    - "false" == false → false ("false" == 0 → NaN == 0)
    - "2"+2 → "22"
  - ***undefined***:
    - Se comporta como un false cuando se utiliza en un contexto booleano.
    - Se convierte en NaN cuando se usa en una operación aritmética.
  - ***null***:
    - Se comporta como un false cuando se utiliza en un contexto booleano.
    - Se convierte en 0 cuando se usa en una operación aritmética.

5. Operadores

CONVERSIÓN DE TIPOS

- Se recomienda evitar mezclar tipos de datos en las operaciones, hay que utilizar funciones de forzado de tipo (*casting*):
  - ***Number***(*dato*): es un conversor a tipo numérico (tanto entero como decimal)
    - Cualquier cadena que pueda ser interpretada como un número ('12.45', '0x3f', '314e-2') se convierte en ese número.
    - Una cadena vacía se convierte en 0.
    - Cuando encuentra caracteres no convertibles a número → devuelve *NaN*.
  - ***parseInt***(*string[,base]*): extrae un número entero de una cadena de texto
    - Analiza la cadena carácter a carácter empezando por la izquierda.
    - Cuando se encuentra el primer carácter no numérico se ignora el resto de la cadena.
    - Si el primer carácter encontrado no es convertible a número, el resultado será *NaN*. La cadena vacía también da como resultado *NaN*.
  - ***parseFloat***(*string*): extrae un número (que puede ser decimal) de una cadena.
    - Funciona igual que *parseInt* pero además de números permite un signo, un punto decimal y un exponente.
  - ***Boolean***(*dato*),
  - ***String***(*dato*)

5. Operadores

CONVERSIÓN DE TIPOS

```
Number(3.55)
3.55
parseInt(3.55)
3
parseFloat(3.55)
3.55

Number(3/2)
1.5
parseInt(3/2)
1
parseFloat(3/2)
1.5
```

```
Number("")
0
parseInt("")
NaN

Number("1aaaa1")
NaN
parseInt("1aaaa1")
1

Number(".33")
0.33
parseInt(".33")
NaN
parseFloat(".33")
0.33
```

```
Number(null)
0
parseInt(null)
NaN

Number(undefined)
NaN
parseInt(undefined)
NaN

Number(Infinity)
Infinity
parseInt(Infinity)
NaN

Number(NaN)
NaN
parseInt(NaN)
NaN
Number("NaN")
NaN
parseInt("NaN")
NaN
```

```
Number(true)
1
parseInt(true)
NaN

Number(false)
0
parseInt(false)
NaN
```

```
Number(0xA)
10
Number("0xA")
10
parseInt(0xA)
10
parseInt("0xA")
10
parseFloat(0xA)
10
parseFloat("0xA")
0

Number(0b11)
3
parseInt(0b11)
3
parseFloat(0b11)
3

parseFloat("0b11")
0
```

5. Operadores

DIFERENCIAS ENTRE NULL Y UNDEFINED

```
var miVariable;

console.log(miVariable); // undefined
console.log(typeof miVariable); // "undefined"
```

```
var variable = null;
console.log(variable); // null
console.log(typeof variable); // advertencia! muestra "object"
```

```
var x;

x == null              // true
x == undefined         // true
x === null             // false
x === undefined        // true

var y = null;

y == null              // true
y == undefined         // true
y === null             // true
y === undefined        // false
```

6. Estructuras de control de flujo

- Con las sentencias condicionales se puede gestionar la toma de decisiones y el posterior resultado por parte del navegador.
- Dichas sentencias evalúan condiciones y ejecutan ciertas instrucciones en base al resultado de la condición.
- Las sentencias condicionales en JavaScript son:
  - `if`
  - `switch`
  - `while`
  - `for`

6. Estructuras de control de flujo

SENTENCIA IF

- **Sintaxis**:

```
if (expresión_1){
  instrucciones_1
}else if (expresión_2){
  instrucciones_2
}else{
  instrucciones_3
}
```

```
1  let numero = prompt("Introduce un número: ")
2  if (numero % 2 == 0){
3    alert("El número es divisible entre 2")
4  }else if (numero % 3 == 0){
5    alert("El número es divisible entre 3")
6  }else if (numero % 5 == 0){
7    alert("El número es divisible entre 5")
8  }else if (numero % 7 == 0){
9    alert("El número es divisible entre 7")
10 }else{
11   alert ("El número no es divisible entre ningún entero menor que 10")
12 }
```

- **Nota**: El uso de las llaves {} no es obligatorio en el caso en que debamos ejecutar una sola instrucción, de lo contrario, sí es obligatorio su uso.

```
1  let edad = prompt("Introduce una edad: ")
2  if (edad % 2 == 0)
3    alert("Tu edad es un número par")
4  else
5    alert ("Tu edad es un número impar")
```

6. Estructuras de control de flujo

SENTENCIA SWITCH

- En el caso en el que debamos utilizar demasiadas sentencias del tipo *else-if*, es preferible utilizar la sentencia ***switch***, ya que la lectura y el mantenimiento de una sentencia compleja de varios *if-else* se puede convertir en una tarea difícil → el uso de *switch* es más elegante.
- Esta sentencia consta de una expresión a evaluar y una serie de posibles valores de dicha expresión, llamados **casos**, en los que se encuentran las instrucciones a ejecutar cuando coincidan el valor de la expresión y el valor de cada caso.
- **Break**: Cada caso termina con la palabra clave ***break***.
  - Detiene la ejecución del switch en el momento en que uno de los casos resulte verdadero → no se pierde tiempo en evaluar el resto de casos.
- **Default**: las instrucciones que se encuentran dentro de la sentencia opcional ***default***, se ejecutarán solo cuando ninguno de los valores de cada caso coincida con el valor de la expresión.

6. Estructuras de control de flujo

SENTENCIA SWITCH

- **Sintaxis**:

```
switch (expresión){
  case valor1:
    instrucciones a ejecutar si expresión = valor1
  break;
  case valor2:
    instrucciones a ejecutar si expresión = valor2
  break;
  case valor3:
    instrucciones a ejecutar si expresión = valor3
  break;
  default:
    instrucciones a ejecutar si expresión es diferente a
    los valores anteriores
}
```

6. Estructuras de control de flujo

SENTENCIA SWITCH

- Ejemplo:

```
let edad = prompt("Introduce una edad: ")
switch (edad) {
  case "18":
  case "17":
  case "16":
    alert("Juvenil");
    break;
  case "15":
  case "14":
    alert("Cadete");
    break;
  case "13":
  case "12":
    alert("Infantil");
    break;
  case "11":
  case "10":
    alert("Alevín");
    break;
  case "8":
  case "9":
    alert("Benjamin");
    break;
  default:
    alert("Pre-Benjamin");
}
```

6. Estructuras de control de flujo

SENTENCIA SWITCH

- **Ejemplo**: más elegante que un if … else if

```
1  let numero = prompt("Introduce un numero: ")
2  switch (true) {
3    case (numero % 2 == 0):
4      alert("El número es divisible entre 2");
5      break;
6    case (numero % 3 == 0):
7      alert("El número es divisible entre 3");
8      break;
9    case (numero % 5 == 0):
10     alert("El número es divisible entre 5");
11     break;
12   case (numero % 7 == 0):
13     alert("El número es divisible entre 7");
14     break;
15   default:
16     alert("El número no es divisible entre ningún entero menor que 10");
17 }
```

6. Estructuras de control de flujo

BUCLE WHILE

- Permite que el navegador ejecute una o más instrucciones continuamente hasta que una cierta condición deje de ser verdadera.
- Se utiliza cuando no disponemos de la información que nos indique el número de veces que debemos repetir la iteración del bucle.
- Consta de **tres partes**:
  - Palabra clave ***while***.
  - Expresión condicional
  - Instrucciones que se ejecutan en el caso en que la condición sea verdadera.
- **continue** y **break**: abandonar iteración y el bucle respectivamente.
- **Sintaxis**:

```
while (expresión){
   instrucciones
}
```

```
1   let cont = 0
2 ▼ while (cont<10) {
3       cont++
4       alert(cont)
5   }
```

6. Estructuras de control de flujo

BUCLE WHILE

- **Bucle do-while**:
  - Es una variación del bucle *while*.
  - Asegura que la ejecución del bucle se lleve a cabo al menos una vez, ya que solo al final se evalúa si se debe seguir ejecutando o no.
  - **Sintaxis**:

```
do{
   instrucciones
} while (expresión)
```

```
1   let cont = 0
2   do {
3      cont++
4      alert(cont)
5   }while (cont<10)
```

```
do {
   valor = prompt("Introduce un valor")
} while (valor!=null)
```

6. Estructuras de control de flujo

BUCLE FOR

- Permite que el navegado ejecute las instrucciones que se encuentren dentro del bucle hasta que la expresión condicional devuelva el valor *false*.
- Consta de **cinco partes**:
  - Palabra clave ***for***
  - Valor inicial de la variable de control
  - Condición basada en dicha variable
  - Incremento o decremento de la variable
  - Instrucciones a ejecutar
- **Sintaxis**:

```
for(valor_inicial_variable; expresión_condicional;incremento_o_decremento_de_la_variable){
    instrucciones
}
```

```
1  let i
2  for (i=1;i<=10;i++) {
3    alert(i)
4  }
```