

**DWC**

**(Desarrollo Web en entorno cliente)**



**JavaScript**

**Tema 3**

**Objetos predefinidos**

## Índice

1.- Objetos predefinidos .....	1
2.- Window .....	2
3.- String.....	4
4.- Date .....	8
5.- Number y Math .....	11
6.- Almacenamiento local .....	14
6.1.- Cookies .....	15
6.2.- API de almacenamiento web.....	18
6.3.- IndexedDB .....	19

## 1.-Objetos predefinidos

Los objetos predefinidos en JavaScript son objetos que ya están disponibles en el lenguaje y no necesitan ser creados por el usuario. Estos objetos proporcionan funcionalidades básicas y comunes que son esenciales para el desarrollo web y otras aplicaciones JavaScript.

Algunos de los objetos predefinidos más importantes son:

### 1. Objeto Window:

- Representa la ventana del navegador y proporciona acceso a sus propiedades y métodos, como la URL de la página, el historial del navegador, el tamaño de la ventana, etc.

### 2. Objeto Document:

- Representa el documento HTML de la página web y proporciona acceso a sus elementos, atributos y métodos, como DOM (Document Object Model).

### 3. Objeto String:

- Se utiliza para representar cadenas de texto.
- Proporciona métodos para manipular cadenas de texto, como concatenar, dividir, buscar, reemplazar, convertir a minúsculas/mayúsculas, etc.

### 4. Objeto Date:

- Se utiliza para representar fechas y horas.
- Proporciona métodos para obtener y configurar información de fechas, como el año, el mes, el día, la hora, los minutos y los segundos.

### 5. Objetos Math:

- Proporciona constantes y funciones matemáticas, como PI, E, sen(), cos(), tan(), sqrt(), abs(), pow(), etc.

### 6. Objeto Array:

- Se utiliza para representar listas ordenadas de valores.
- Proporciona métodos para manipular arrays, como agregar, eliminar, insertar, buscar y ordenar elementos.

### 7. Objeto Object:

- Es el objeto base del que heredan todos los demás objetos en JavaScript.

- Proporciona propiedades y métodos para trabajar con objetos, como crear nuevos objetos, definir propiedades, acceder a propiedades y métodos, etc.

#### 8. Objeto Error:

- Se utiliza para representar errores en JavaScript.
- Proporciona propiedades para acceder a información sobre el error, como el mensaje de error, el nombre del archivo y el número de línea donde ocurrió el error.

#### 9. Objeto JSON:

- Se utiliza para analizar y generar JSON (JavaScript Object Notation), un formato de intercambio de datos ligero y legible por humanos.

#### 10. Objeto XMLHttpRequest:

- Se utiliza para realizar solicitudes HTTP y recuperar datos de servidores web.

## 2.- Window

El objeto window en JavaScript es un objeto global que representa la ventana del navegador y proporciona acceso a sus propiedades y métodos. Es uno de los objetos más importantes en JavaScript y se utiliza para interactuar con la ventana del navegador, obtener información sobre la página web actual, manipular elementos HTML y controlar el comportamiento del navegador.

#### Propiedades importantes del objeto window:

- **location:** Proporciona información sobre la URL actual de la página web, el historial del navegador y permite redirigir a otras páginas.

```
console.log(window.location.href); // Imprime la URL actual
window.location.href = "https://www.example.com"; // Redirige a otra página
```

- **document:** Representa el documento HTML de la página web y proporciona acceso a sus elementos, atributos y métodos (DOM - Document Object Model).

```
let titulo = document.getElementById("tituloPagina");
titulo.textContent = "Nuevo título de la página";
```

- **history:** Proporciona acceso al historial del navegador, permitiendo navegar entre las páginas visitadas anteriormente.

```
window.history.back(); // Navega a la página anterior
window.history.go(2); // Navega dos páginas hacia atrás
```

- **screen:** Proporciona información sobre las dimensiones de la pantalla, la profundidad de color y la resolución.

```
let anchoPantalla = window.screen.width;
console.log(`Ancho de la pantalla: ${anchoPantalla} pixeles`);
```

- **navigator:** Proporciona información sobre el navegador web que se está utilizando, como el nombre del navegador, la versión y el sistema operativo.

```
let navegador = window.navigator.userAgent;
console.log(`Navegador: ${navegador}`);
```

### Métodos importantes del objeto window:

Estos métodos son los que nos permiten interactuar de una forma sencilla con el usuario, más adelante veremos que la forma más adecuada serán los formularios.

- **alert():** Muestra un cuadro de diálogo emergente con un mensaje y un botón de "Aceptar". Se utiliza para informar al usuario o mostrar mensajes de error simples.

```
window.alert("¡Hola, mundo!");
```

- **prompt():** Muestra un cuadro de diálogo emergente con un mensaje, un campo de entrada y los botones Ok y Cancelar. Se utiliza para obtener información del usuario, como un nombre o un valor.

Los resultados obtenidos son:

- Si pulsamos en Ok obtenemos el valor del campo
- Si pulsamos en Cancelar obtenemos **null**

```
let nombre = window.prompt("Dime un nombre:");
console.log("Nombre:", nombre);
```

- **confirm():** Muestra un cuadro de diálogo emergente con un mensaje y dos botones: "Aceptar" y "Cancelar". Se utiliza para confirmar una acción u obtener la decisión del usuario.

Los resultados obtenidos son:

- Si pulsamos en Aceptar obtenemos el valor true
- Si pulsamos en Cancelar obtenemos el valor false

```
let confirmar = window.confirm("¿Desea continuar?");
if (confirmar) {
  console.log("Usuario confirmó la acción");
} else {
  console.log("Usuario canceló la acción");
}
```

- **open():** Abre una nueva ventana del navegador con la URL especificada. Se utiliza para abrir enlaces en una nueva pestaña o ventana.

```
// Abre en una nueva pestaña
window.open("https://www.example.com", "_blank");
```

- **close():** Cierra la ventana del navegador actual. Se utiliza para cerrar la ventana actual o una ventana abierta con open y de la cual nos hemos guardado una referencia

### 3.- String

En JavaScript, los datos de tipo texto se almacenan como cadenas. No hay un tipo char para un solo carácter.

Los strings se representan mediante texto entrecomillado, existen 3 tipos de entrecomillado

- **Comillas dobles** o **comillas simples**: Permiten representar únicamente texto y en una sola línea.
- Los **backticks**, Además de representar las cadenas de texto, nos permiten incrustar cualquier expresión en la cadena, envolviéndola en `${...}` (interpolation de variables). También permiten utilizar un formato multilinea.

```
let nombre = "Juan";
let saludo = `Hola, ${nombre}`;
console.log(saludo); // Imprime "Hola, Juan"
```

Las comillas simples y dobles provienen de la antigüedad de la creación del lenguaje, cuando no se tuvo en cuenta la necesidad de cadenas multilínea. Los backticks aparecieron mucho después y, por lo tanto, son más versátiles.

### Creación de cadenas de texto:

Las cadenas de texto en JavaScript se pueden crear de dos maneras principales:

- **Literales de cadena:** Se utilizan comillas simples o dobles para delimitar la cadena de caracteres.

```
let saludo = "Hola, mundo!";  
let nombre = 'Juan';
```

- **Constructor String():** Se puede utilizar el constructor String() para crear una cadena a partir de cualquier valor.

```
let nombre = new String('Juan');
```

### Propiedades importantes del objeto String:

- **length:** Devuelve la longitud de la cadena.

```
let frase = "JavaScript es un lenguaje de programación";  
console.log(`Longitud de la frase: ${frase.length}`);
```

### Métodos importantes del objeto String

- **charAt(índice):** Devuelve el carácter en la posición especificada por el índice.

```
let texto = "ABCDEFGHijklmn";  
// Imprime "F" (carácter en la posición 5)  
console.log(texto.charAt(5));
```

- **charCodeAt(índice):** Devuelve el código Unicode del carácter en la posición especificada por el índice.

```
let letra = "ñ";  
// Imprime "165" (código Unicode de la letra "ñ")  
console.log(letra.charCodeAt(0));
```

- **indexOf(subcadena, inicio):** Busca la primera aparición de una subcadena dentro de la cadena, devolviendo la posición de inicio.

```
let frase = "Esta frase contiene la palabra programación";
let indice = frase.indexOf("programación");
if (indice !== -1) {
  console.log(`programación esta en la posición: ${indice}`);
} else {
  console.log("programación no se encuentra en la frase");
}
```

- **lastIndexOf(subcadena, inicio):** Busca la última aparición de una subcadena dentro de la cadena, devolviendo la posición de inicio. Similar a `indexOf` pero empezando por el final de la cadena
- **replace(subcadena, reemplazo):** Reemplaza todas las apariciones de una subcadena por una cadena de reemplazo.

```
let texto = "Hola a todos, soy un robot";
let textoReemplazado = texto.replace("robot", "humano");
// Imprime "Hola a todos, soy un humano"
console.log(textoReemplazado);
```

- **toUpperCase()** y **toLowerCase():** Convierten la cadena a mayúsculas y minúsculas
- **trim():** Elimina los espacios en blanco al inicio y al final de la cadena. También están disponibles `startTrim()` y `endTrim()`

```
let texto = "  Hola Mundo  ";
console.log(texto.trim()); // Imprime "Hola Mundo"
```

- **concat(cadena2, ...):** Concatena dos o más cadenas.

```
let nombre = "Juan";
let apellido = "Pérez";
let nombreCompleto = nombre.concat(" ", apellido);
console.log(nombreCompleto); // Imprime "Juan Pérez"
```

- **slice(inicio, fin):** Extrae una subcadena desde una posición inicial hasta una posición final especificada.

```
let frase = "JavaScript es un lenguaje de programación";
// Extrae la subcadena "es un lenguaje"
let subcadena = frase.slice(10, 25);
console.log(subcadena);
```



- **split()**: se utiliza para dividir una cadena de texto en una lista de subcadenas, basándose en un separador especificado. Es uno de los métodos más comunes para trabajar con cadenas y tiene diversas aplicaciones en el desarrollo web.

### Sintaxis

`cadena.split([separador], [limite])`

### Parámetros:

- **separador** (opcional): Cadena que se utiliza como separador para dividir la cadena original. Si no se especifica, se utiliza una coma (,) como separador por defecto.
- **límite** (opcional): Número entero que indica el límite máximo de subcadenas a devolver. Si no se especifica, se devuelven todas las subcadenas posibles.

El método `split()` devuelve un array que contiene las subcadenas resultantes de la división. Cada elemento del array es una subcadena separada por el separador especificado.

#### 1. Dividir una cadena por espacios:

```
let frase = "Esta frase se divide por espacios";
let palabras = frase.split(" ");
console.log(palabras); // Imprime un array con las palabras:
["Esta", "frase", "se", "divide", "por", "espacios"]
```

#### 2. Dividir una cadena por comas:

```
let lista = "rojo,verde,azul,amarillo";
let colores = lista.split(",");
console.log(colores); // Imprime un array con los colores:
["rojo", "verde", "azul", "amarillo"]
```

#### 3. Dividir una cadena con un límite especificado:

```
let cadenaLarga = "Lorem ipsum dolor sit amet, consectetur adipiscing elit.
let partes = cadenaLarga.split(" ", 5); // Límite de 5 subcadenas
```

```
console.log(partes); // Imprime las primeras 5 palabras de la cadena
```

#### 4. Obtener caracteres individuales:

```
let texto = "Hola, mundo!";  
let caracteres = texto.split("");  
console.log(caracteres); // Imprime un array con cada caracter
```

Estos son los métodos más habituales, para conocer todos los métodos del objeto string:

[https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global\\_Objects/String](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String)

## 4.- Date

El objeto Date en JavaScript representa un punto específico en el tiempo. Se utiliza para trabajar con fechas y horas, permitiendo obtener la fecha actual, crear fechas futuras o pasadas,

### Creación de objetos Date:

Existen diferentes formas de crear un objeto Date:

#### 1. Sin argumentos:

- Crea un objeto Date con la fecha y hora actual según el reloj del sistema.

```
let fechaActual = new Date();  
console.log(fechaActual);
```

#### 2. Con argumentos numéricos:

- Permite crear un objeto Date con una fecha y hora específicas especificando valores numéricos para el año, mes, día, hora, minutos, segundos y milisegundos.

```
let fechaEspecifica = new Date(2024, 5, 12, 10, 30, 15, 500);  
console.log(fechaEspecifica);
```

#### 3. Con cadena de fecha:

- Permite crear un objeto Date a partir de una cadena de texto que represente una fecha y hora en un formato específico.

```
let fechaCadena = "2024-06-12T10:30:15.500+02:00";  
let fechaObjeto = new Date(fechaCadena);  
console.log(fechaObjeto);
```

## Métodos get del objeto Date

El objeto Date en JavaScript proporciona diversos métodos get para acceder a diferentes componentes de una fecha y hora específica. Estos métodos permiten obtener información como el año, mes, día, hora, minutos, segundos y milisegundos de la fecha representada por el objeto Date.

Los métodos get más comunes del objeto Date son:

### 1. getFullYear():

Obtiene y devuelve el año de la fecha como un número entero de cuatro dígitos.

```
let fecha = new Date();
let año = fecha.getFullYear();
console.log(`Año actual: ${año}`); // Ejemplo: 2024
```

### 2. getMonth():

Obtiene y devuelve el mes de la fecha como un número entero de 0 a 11, donde 0 representa enero y 11 diciembre.

```
let fecha = new Date();
let mes = fecha.getMonth();
console.log(`Mes actual: ${mes + 1}`); // Ejemplo: 6 (junio)
```

### 3. getDate():

Obtiene y devuelve el día del mes de la fecha como un número entero de 1 a 31.

```
let fecha = new Date();
let dia = fecha.getDate();
console.log(`Día actual: ${dia}`); // Ejemplo: 12
```

### 4. getHours():

Obtiene y devuelve la hora de la fecha como un número entero de 0 a 23.

JavaScript

```
let fecha = new Date();
let hora = fecha.getHours();
console.log(`Hora actual: ${hora}`); // Ejemplo: 13
```

### 5. getMinutes():

Obtiene y devuelve los minutos de la fecha como un número entero de 0 a 59.

```
let fecha = new Date();
let minutos = fecha.getMinutes();
console.log(`Minutos actuales: ${minutos}`); // Ejemplo: 26
```

## 6. getSeconds():

Obtiene y devuelve los segundos de la fecha como un número entero de 0 a 59.

```
let fecha = new Date();  
let segundos = fecha.getSeconds();  
console.log(`Segundos actuales: ${segundos}`); // Ejemplo: 12
```

## 7. getMilliseconds():

Obtiene y devuelve los milisegundos de la fecha como un número entero de 0 a 999.

```
let fecha = new Date();  
let milisegundos = fecha.getMilliseconds();  
console.log(`Milisegundos actuales: ${milisegundos}`);
```

## Métodos set del objeto Date

El objeto Date en JavaScript proporciona diversos métodos set para modificar los componentes de una fecha y hora específica. Estos métodos permiten establecer valores como el año, mes, día, hora, minutos, segundos y milisegundos de la fecha representada por el objeto Date.

Los métodos set más comunes del objeto Date son:

### 1. setFullYear(año):

Establece el año de la fecha en el valor entero especificado.

```
let fecha = new Date();  
fecha.setFullYear(2023); // Ejemplo: establecer el año a 2023  
console.log(fecha.getFullYear()); // Imprime: 2023
```

### 2. setMonth(mes):

Establece el mes de la fecha en el valor entero especificado (de 0 a 11, donde 0 representa enero y 11 diciembre).

```
let fecha = new Date();  
fecha.setMonth(4); // Ejemplo: establecer el mes a mayo (mes 4)  
console.log(fecha.getMonth() + 1); // Imprime: 5 (mayo)
```

### 3. setDate(día):

Establece el día del mes de la fecha en el valor entero especificado (de 1 a 31).

```
let fecha = new Date();  
fecha.setDate(10); // Ejemplo: establecer el día a 10  
console.log(fecha.getDate()); // Imprime: 10
```

#### 4. setHours(hora):

Establece la hora de la fecha en el valor entero especificado (de 0 a 23).

```
let fecha = new Date();  
fecha.setHours(18); // Ejemplo: establecer la hora a 18  
console.log(fecha.getHours()); // Imprime: 18
```

#### 5. setMinutes(minutos):

Establece los minutos de la fecha en el valor entero especificado (de 0 a 59).

```
let fecha = new Date();  
fecha.setMinutes(30); // Ejemplo: establecer los minutos a 30  
console.log(fecha.getMinutes()); // Imprime: 30
```

#### 6. setSeconds(segundos):

Establece los segundos de la fecha en el valor entero especificado (de 0 a 59).

```
let fecha = new Date();  
fecha.setSeconds(15); // Ejemplo: establecer los segundos a 15  
console.log(fecha.getSeconds()); // Imprime: 15
```

#### 7. setMilliseconds(milisegundos):

Establece los milisegundos de la fecha en el valor entero especificado (de 0 a 999).

```
let fecha = new Date();  
fecha.setMilliseconds(500); // Ejemplo: establecer los milisegundos a 500  
console.log(fecha.getMilliseconds()); // Imprime: 500
```

Estos son los métodos más habituales, para conocer todos los métodos del objeto Date:

[https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global\\_Objects/Date](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Date)

## 5.- Number y Math

Para trabajar con números y funciones matemáticas disponemos de dos objetos: Number y Math.

### Number

El objeto Number en JavaScript representa valores numéricos, tanto enteros como decimales. Proporciona propiedades y métodos para trabajar con números, incluyendo:

**Propiedades:**

- **MAX\_VALUE:** El valor numérico más grande posible (representación de punto flotante).
- **MIN\_VALUE:** El valor numérico más pequeño posible (representación de punto flotante).
- **NEGATIVE\_INFINITY:** Representa el infinito negativo (-Infinity).
- **POSITIVE\_INFINITY:** Representa el infinito positivo (Infinity).
- **NaN:** Representa un valor no numérico ("Not a Number").

**Métodos principales:**

- **toString():** Convierte un número a una cadena de texto.
- **toFixed(n):** Convierte un número a una cadena de texto con un número específico de decimales.
- **toExponential(n):** Convierte un número a una cadena de texto en notación exponencial.
- **valueOf():** Devuelve el valor primitivo del número (el número en sí).
- **isFinite(number):** Comprueba si un número es finito (no NaN ni infinito).
- **isNaN(number):** Comprueba si un valor es NaN ("Not a Number").
- **isInteger(number):** Comprueba si un número es un entero.
- **parse(string):** Convierte una cadena de texto a un número.

**Ejemplos:**

```
// Convertir un número a una cadena de texto con 2 decimales
const numero = 123.4567;
const texto = numero.toFixed(2); // "123.46"

// Comprobar si un número es finito
const esFinito = Number.isFinite(numero); // true

// Comprobar si un valor es NaN
const esNaN = Number.isNaN('Hola'); // true

// Convertir una cadena de texto a un número
const cadena = '100';
const numeroConvertido = Number(cadena); // 100

// Crear un objeto Number con el constructor
const nuevoNumero = new Number(200);
console.log(nuevoNumero); // Number {valueOf: 200}
```

## Math

El objeto Math en JavaScript es un objeto global que proporciona constantes y funciones matemáticas. Es una herramienta esencial para trabajar con números en aplicaciones web.

### Propiedades:

- **Constantes matemáticas:**
  - $\pi$  (Pi): Representa el valor de pi (aproximadamente 3.14159).
  - e (Base de los logaritmos naturales): Aproximadamente 2.71828.
  - LN2 (Logaritmo natural de 2): Aproximadamente 0.693147.
  - LN10 (Logaritmo natural de 10): Aproximadamente 2.302585.
  - SQRT2 (Raíz cuadrada de 2): Aproximadamente 1.414214.
  - SQRT1\_2 (Raíz cuadrada de 1/2): Aproximadamente 0.707107.
  - E (Número de Euler): Aproximadamente 2.718282.
  - Infinity (Infinito positivo): Representa el valor infinito.
  - NEGATIVE\_INFINITY (Infinito negativo): Representa el valor infinito negativo.
  - NaN ("Not a Number"): Representa un valor no numérico.
- **Constantes de redondeo:**
  - ROUND\_DOWN (1): Redondea hacia abajo.
  - ROUND\_UP (2): Redondea hacia arriba.
  - ROUND\_CEIL (3): Redondea hacia arriba (equivalente a Math.ceil()).
  - ROUND\_FLOOR (4): Redondea hacia abajo (equivalente a Math.floor()).
  - ROUND\_HALF\_UP (5): Redondea hacia el valor más cercano, con desempate hacia arriba.
  - ROUND\_HALF\_DOWN (6): Redondea hacia el valor más cercano, con desempate hacia abajo.

### Funciones:

- **Funciones aritméticas básicas:**
  - abs(x): Valor absoluto de x.
  - pow(x, y): Eleva x a la potencia y.
  - sqrt(x): Raíz cuadrada de x.
  - floor(x): Redondea x hacia abajo al entero más cercano.

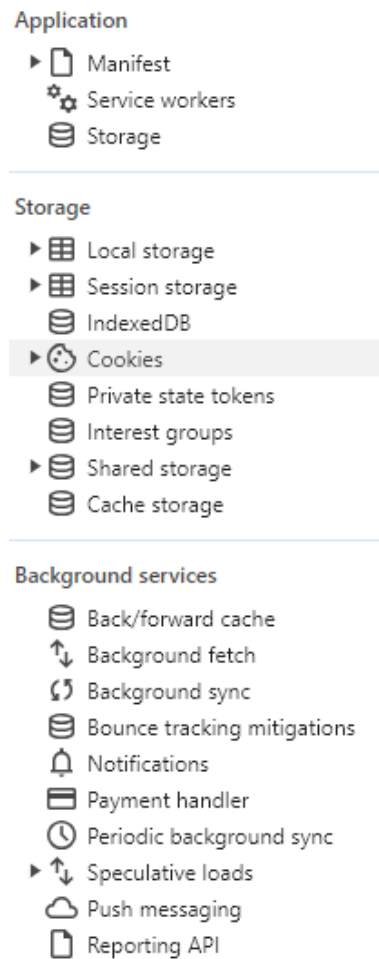
- `ceil(x)`: Redondea x hacia arriba al entero más cercano.
- `round(x)`: Redondea x al entero más cercano.
- `random()`: Genera un número aleatorio entre 0 y 1.
- **Funciones trigonométricas:**
  - `sin(x)`: Seno de x (en radianes).
  - `cos(x)`: Coseno de x (en radianes).
  - `tan(x)`: Tangente de x (en radianes).
  - `asin(x)`: Arcoseno de x (en radianes, entre  $-\pi/2$  y  $\pi/2$ ).
  - `acos(x)`: Arcocoseno de x (en radianes, entre 0 y  $\pi$ ).
  - `atan(x)`: Arcotangente de x (en radianes, entre  $-\pi/2$  y  $\pi/2$ ).
  - `atan2(y, x)`: Arcotangente de y/x (en radianes, entre  $-\pi$  y  $\pi$ ).
- **Funciones de logaritmos:**
  - `log(x)`: Logaritmo en base 10 de x.
  - `log10(x)`: Logaritmo en base 10 de x.
  - `log2(x)`: Logaritmo en base 2 de x.
  - `exp(x)`: Exponencial de x ( $e^x$ ).
- **Funciones de redondeo:**
  - `round(x)`: Redondea x al entero más cercano.
  - `floor(x)`: Redondea x hacia abajo al entero más cercano.
  - `ceil(x)`: Redondea x hacia arriba al entero más cercano.
- **Otras funciones:**
  - `max(x, y, ...)`: Devuelve el valor máximo entre los argumentos.
  - `min(x, y, ...)`: Devuelve el valor mínimo entre los argumentos.

## 6.- Almacenamiento local

En Javascript disponemos de varias alternativas para poder trabajar con almacenamiento de datos interno y en modo local. Tenemos **la pestaña Aplicación** en la consola del navegador para poder comprobar estas herramientas.

La pestaña Aplicación en la consola del navegador (herramientas de desarrollo) proporciona información y herramientas para depurar y analizar el comportamiento de una aplicación web.





## 6.1.- Cookies

Una cookie HTTP, cookie web o cookie de navegador es una pequeña pieza de datos que un servidor envía a el navegador web del usuario. El navegador guarda estos datos y los envía de regreso junto con la nueva petición al mismo servidor. Las cookies se usan generalmente para decirle al servidor que dos peticiones tienen su origen en el mismo navegador web lo que permite, por ejemplo, mantener la sesión de un usuario abierta. Las cookies permiten recordar la información de estado en vista a que el protocolo HTTP es un protocolo sin estado.

Como funcionan con el protocolo HTTP deberemos de abrir nuestra página Web desde un servidor web

Las cookies se almacenan como un diccionario guardando una clave y un valor, si hay más de una cookie van separadas por un punto y coma. También es recomendable añadirles un tiempo de vida y un path desde donde podrían ser alcanzadas

Las cookies se utilizan principalmente con tres propósitos:

- **Gestión de Sesiones:** Inicios de sesión, carritos de compras, puntajes de juegos o cualquier otra cosa que el servidor deba recordar
- **Personalización:** Preferencias de usuario, temas y otras configuraciones
- **Rastreo:** Guardar y analizar el comportamiento del usuario

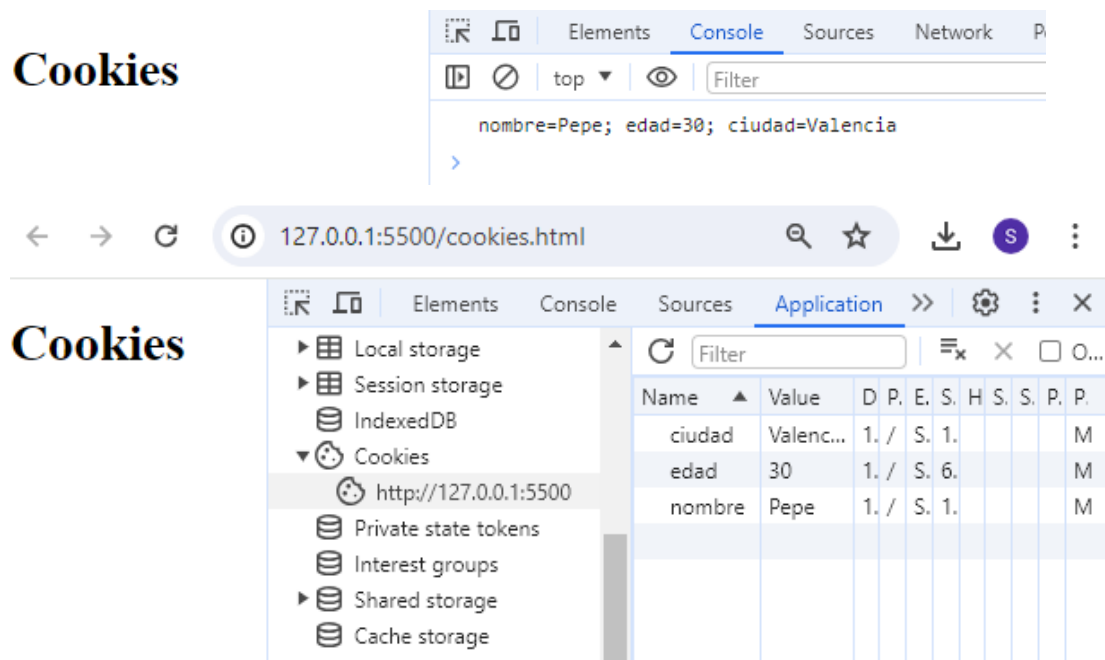
### Creación de cookies

Se pueden acceder y crear nuevas cookies usando la propiedad `document.cookie`.

```
document.cookie = "nombre=Pepe";
document.cookie = "edad=30";
document.cookie = "ciudad=Valencia";

alert("Cookies guardadas correctamente.");
console.log(document.cookie)
```

## Cookies



The screenshot shows a web browser interface. The top part displays the 'Console' tab with the output of the JavaScript code: `nombre=Pepe; edad=30; ciudad=Valencia`. Below this, the 'Application' tab is selected, showing a tree view of storage areas. Under 'Cookies', the cookies for the URL `http://127.0.0.1:5500` are listed in a table.

Name	Value	D	P	E	S	H	S	S	P	P
ciudad	Valenc...	1.	/	S.	1.					M
edad	30	1.	/	S.	6.					M
nombre	Pepe	1.	/	S.	1.					M

Cuando trabajamos hay dos propiedades importantes para configurar

- **path:** Usualmente, debemos configurarlo en la raíz: `path=/'` para hacer la cookie accesible a todas las páginas del sitio web

```
document.cookie = "ciudad=Valencia; path=/'
```

- **expires, max-age:** de forma predeterminada, si una cookie no tiene una de estas opciones, desaparece cuando el navegador se cierra. Tales cookies se denominan “**cookies de sesión**”. Para que las cookies sobrevivan al cierre del navegador, podemos usar las opciones `expires` o `max-age`.

**La fecha de expiración** define el momento en que el navegador la borra automáticamente, el formato es `expires=Tue, 19 Jan 2038 03:14:07 GMT`

La fecha debe estar exactamente en ese formato, en el huso horario GMT. Podemos obtenerlo con `date.toUTCString`. Por ejemplo, podemos configurar que la cookie expire en un día:

```
// +1 día desde ahora
let date = new Date(Date.now() + 86400e3);
date = date.toUTCString();
document.cookie = "ciuda=Valencia" + date;
```

Si establecemos `expires` en una fecha en el pasado, la cookie es eliminada.

- **max-age** es una alternativa a `expires`, especifica la expiración de la cookie en segundos desde el momento actual el formato es `max-age=3600`

```
// la cookie morirá en +1 hora a partir de ahora
document.cookie = "nombre=Pepe; max-age=3600";
alert();
// borra la cookie (la hacemos expirar ya)
document.cookie = "nombre=Pepe; max-age=0";
```

Si la configuramos a cero o un valor negativo, la cookie es eliminada:

Las cookies se usaron una vez para el almacenamiento general del lado del cliente. Si bien esto era legítimo cuando eran la única forma de almacenar datos en el cliente, hoy en día se recomienda preferir las API de almacenamiento

modernas. Las cookies se envían con cada solicitud, por lo que pueden empeorar el rendimiento (especialmente para las conexiones de datos móviles).

Las APIs modernas para el almacenamiento del cliente son la [Web storage API](#) (`localStorage` y `sessionStorage`) e [IndexedDB](#). La más utilizada es `localStorage`

## 6.2.- API de almacenamiento web

La **API de almacenamiento web** proporciona los mecanismos mediante los cuales el navegador puede almacenar información de tipo clave/valor, de una forma mucho más intuitiva que utilizando cookies.

Los dos mecanismos en el almacenamiento web son los siguientes:

- **sessionStorage** mantiene un área de almacenamiento separada para cada origen que está disponible mientras dure la sesión de la página (mientras el navegador esté abierto, incluyendo recargas de página y restablecimientos).
- **localStorage** hace lo mismo, pero persiste incluso cuando el navegador se cierre y se reabra.

### LocalStorage

El almacenamiento local (`localStorage`) es una API web que permite a las aplicaciones web almacenar datos en el navegador del usuario de forma persistente. Esto significa que los datos almacenados no se pierden cuando se cierra la pestaña o ventana del navegador, y están disponibles para la misma aplicación en futuras sesiones.

El almacenamiento local utiliza pares clave-valor para almacenar datos. La clave es una cadena de texto que identifica el dato, y el valor puede ser cualquier tipo de dato que pueda ser convertido en una cadena JSON (por ejemplo, strings, números, booleanos, arrays, objetos).

**Los métodos más importantes del almacenamiento local son:**

- **setItem(clave, valor):** Almacena un dato en el almacenamiento local con la clave especificada y el valor correspondiente.

```
localStorage.setItem("nombre", "Juan Pérez");
localStorage.setItem("edad", 30);
```

- **getItem(clave):** Recupera el valor almacenado en el almacenamiento local con la clave especificada. Si la clave no existe, devuelve null.

```
let nombre = localStorage.getItem("nombre");
console.log(nombre); // Imprime: "Juan Pérez"

let edad = localStorage.getItem("edad");
console.log(edad); // Imprime: 30
```

- **removeItem(clave):** Elimina el dato almacenado en el almacenamiento local con la clave especificada.

```
localStorage.removeItem("nombre");
console.log(localStorage.getItem("nombre")); // Imprime: null

localStorage.removeItem("edad");
console.log(localStorage.getItem("edad")); // Imprime: null
```

- **clear():** Elimina todos los datos almacenados en el almacenamiento local.

```
localStorage.clear();
console.log(localStorage.getItem("nombre")); // Imprime: null
console.log(localStorage.getItem("edad")); // Imprime:
null
```

### 6.3.- IndexedDB

IndexedDB es una API de bajo nivel que ofrece almacenamiento en el cliente de cantidades significativas de datos estructurados, incluyendo archivos y blobs. Para búsquedas de alto rendimiento en esos datos usa índices. Mientras localStorage es útil para el almacenamiento de pequeñas cantidades de datos, no es útil para almacenar grandes cantidades de datos estructurados. IndexedDB proporciona una solución.

Para este tipo de almacenamiento podemos recurrir a peticiones al servidor que almacenara los datos en una API Rest