Actividades unidad 3: Cookies, localStorage, Strings, Objetos y Funciones

Preguntas teóricas

1. Cookies y almacenamiento web:

a. Explica qué es una cookie y para qué sirve en el desarrollo web.

Una cookie es un pequeño archivo de texto que un sitio web almacena en el navegador del usuario. Sirve principalmente para recordar información sobre el usuario entre diferentes solicitudes o visitas.
Usos comunes:

- Gestión de sesiones: Mantener a un usuario logueado.
- Personalización: Recordar preferencias (tema oscuro, idioma).
- Rastreo (Tracking): Analizar el comportamiento del usuario para marketing o analíticas.

b. Diferencia entre expires y max-age.

Ambos atributos determinan la vida útil de una cookie, pero de forma distinta:

- expires: Define una fecha y hora específica en formato UTC en la que la cookie dejará de ser válida. Si la fecha es anterior a la actual, se borra.
- max-age: Define la duración de la cookie en segundos a partir del momento en que se crea. Es más moderno y tiene prioridad sobre expires si ambos están presentes.

c. ¿Qué tamaño máximo suele tener una cookie?

El tamaño máximo suele ser de 4 KB (4096 bytes) por cookie, incluyendo el nombre y el valor.

d. Explica por qué es necesario usar EncodeURIComponent() al guardar una cookie.

Las cookies se almacenan como una cadena de texto separada por puntos y coma (;). Si el valor que queremos guardar contiene caracteres especiales (como espacios, comas, puntos y coma o caracteres no ASCII), puede romper el formato de la cookie y causar errores de lectura. encodeURIComponent() convierte estos caracteres en un formato seguro (codificación URL) para su almacenamiento.

e. ¿Cómo se borra una cookie?

Para borrar una cookie, debes volver a definirla con el mismo nombre (y ruta/dominio si aplica) pero estableciendo su fecha de expiración en el pasado.

- Ejemplo con max-age: document.cookie = "usuario=; max-age=0";

- Ejemplo con expires: document.cookie = "usuario=; expires=Thu, 01 Jan 1970 00:00:00 UTC";

2. localStorage y sessionStorage:

a. Diferencias entre localStorage y sessionStorage.

localStorage almacena los datos indefinidamente (hasta que se borren manualmente o por código), incluso si se cierra el navegador. sessionStorage solo mantiene los datos mientras la pestaña o ventana del navegador esté abierta; al cerrarla, los datos se pierden.

Alcance: sessionStorage es específico de la pestaña, mientras que localStorage se comparte entre pestañas del mismo origen.

b. ¿Qué métodos tiene localStorage para guardar y recuperar información?

- localStorage.setItem('clave', 'valor'): Guarda un dato.
- localStorage.getItem('clave'): Recupera un dato.
- localStorage.removeItem('clave'): Elimina un dato específico.
- localStorage.clear(): Borra todos los datos almacenados.
- localStorage.key(índice): Devuelve el nombre de la clave en una posición numérica.

c. Explica un caso real donde sería adecuado usar sessionStorage.

Un formulario de varios pasos (por ejemplo, la compra de un billete de avión o una encuesta larga) donde no quieres que los datos persistan si el usuario decide abandonar la página o cierra la pestaña, pero sí necesitas que la información se mantenga si recarga la página accidentalmente durante el proceso.

3. Manipulación de strings

a. ¿Qué diferencia hay entre .substring() y .substr()?

- .substring(inicio, fin): Extrae caracteres desde el índice inicio hasta el índice fin (sin incluirlo). Si se intercambian los argumentos, funciona igual (el menor es el inicio).
- .substr(inicio, longitud): Extrae caracteres desde el índice inicio tomando la cantidad de caracteres indicada en longitud. Nota: .substr() se considera legacy (obsoleto) y se recomienda no usarlo en código nuevo.

b. ¿Qué hace .startsWith() y qué devuelve?

Este método comprueba si una cadena de texto comienza con los caracteres de otra cadena especificada. Devuelve un valor booleano ,true si empieza con esa cadena, false si no.

c. ¿Qué diferencia existe entre .replace() y .replaceAll()?

•.replace(): Solo reemplaza la primera aparición de la cadena buscada (a menos que se use una expresión regular con la bandera global /g).

•.replaceAll(): Reemplaza todas las apariciones de la cadena buscada en el texto original.

4. Objetos en JavaScript

a. Explica dos formas de crear objetos en JavaScript.

1.Literal de objeto: Es la forma más común.

- const coche = { marca: "Toyota", modelo: "Corolla" };

2.Constructor new Object():

- const coche = new Object(); coche.marca = "Toyota";

b. ¿Qué hace el operador in en un bucle for (campo in objeto)?

El bucle for...in itera sobre todas las propiedades enumerables (las claves) de un objeto. En cada iteración, la variable definida toma el nombre de una propiedad (ej: "nombre", "edad").

c. ¿Qué diferencia hay entre acceder a una propiedad con usuario.nombre y usuario['nombre']?

•Notación de punto (usuario.nombre): Es más limpia y común, pero requiere que se conozca el nombre de la propiedad al escribir el código y que este sea un identificador válido (sin espacios, no empieza por números).

•Notación de corchetes (usuario['nombre']): Permite usar variables para acceder a propiedades dinámicamente (ej: usuario[variable]) y permite acceder a claves que tienen espacios o caracteres especiales.

5. Funciones en JavaScript

a. ¿Qué es una función anónima?

Es una función que no tiene nombre. Generalmente se asignan a una variable o se pasan como argumento (callback) a otra función.
Ejemplo: const saludo = function() { console.log("Hola"); };

b. ¿Qué es una IIFE? Pon un ejemplo.

Significa Immediately Invoked Function Expression (Expresión de función ejecutada inmediatamente). Es una función que se define y se ejecuta al mismo tiempo. Sirve para crear un ámbito local y no contaminar el global. Ejemplo:

```
(function() {
    let secreto = "Dato privado";
    console.log("Me ejecuto ahora mismo");
})();
```

c. ¿Qué ventajas ofrecen las funciones flecha?

1.Sintaxis más corta: Permiten escribir funciones en menos líneas (incluso omitir return y llaves si es una sola línea).

2.this léxico: No crean su propio contexto para this, sino que heredan el this del ámbito donde fueron definidas. Esto es muy útil en callbacks y eventos.

6. Temporizadores

a. ¿Qué diferencia hay entre setTimeout() y setInterval()?

•setTimeout(funcion, tiempo): Ejecuta la función una única vez después de que pase el tiempo indicado (en milisegundos).

•setInterval(funcion, tiempo): Ejecuta la función de manera repetitiva e indefinida cada vez que pasa el intervalo de tiempo indicado.

b. ¿Qué hace clearTimeout()? ¿Y clearInterval()?

Ambos detienen la ejecución de un temporizador pendiente. Para usarlos, necesitas el ID que devuelve la función al crearse.

•clearTimeout(id): Evita que se ejecute la función programada con setTimeout.

•clearInterval(id): Detiene el bucle de repeticiones iniciado con setInterval.

7. Arrays y colecciones

a. ¿Para qué sirve .filter() en un array?

El método .filter() crea un nuevo array con todos los elementos que cumplan una condición implementada por una función proporcionada. Si el elemento devuelve true en la función, se incluye; si es false, se descarta. No modifica el array original.

b. **¿Qué diferencia hay entre Map y Array?**

•Array: Es una colección ordenada de elementos indexados numéricamente (0, 1, 2...). Se usa para listas de datos.

•Map: Es una colección de pares clave-valor. A diferencia de los objetos, las claves en un Map pueden ser de cualquier tipo (incluso objetos o funciones), mantiene el orden de inserción y tiene métodos específicos para obtener el tamaño (.size).

c. ¿Qué devuelve map.get(key) si la clave no existe?

Devuelve undefined.