Desarrollo Web en Entorno Cliente

Tema 3

Utilización de los objetos predefinidos del lenguaje

Tema 3
Utilización de los objetos predefinidos del lenguaje

1. Manejo del tiempo en javascript
2. Cookies
3. Almacenamiento local
4. Objetos en javascript
5. Funciones en javascript

1.- Manejo del tiempo en JavaScript

- JavaScript es la manera mas útil y eficiente de trabajar con el tiempo en un navegador.
  - No recargamos de servicios al servidor
  - Tiene un objeto Date propio que utilizamos para definir contadores relojes, etc

FUNDAMENTAL

El BOM y el tiempo

El BOM (Browser Object Model), más concretamente el objeto window, proporciona los métodos setTimeout() y setInterval() que permiten manejar el tiempo dentro del navegador.

1.- Manejo del tiempo en JavaScript

- El objeto Date almacena fechas y horas.
  - Una vez creado lo podemos modificar y realizar operaciones o cálculos sobre él

```
var miFecha = new Date();
```

  - La variable **miFecha** contendrá fecha y hora actual

```
<div id="laHora"></div>
<script>
  var miFecha = new Date();
  var texto = document.getElementById('laHora');
  texto.innerHTML = miFecha;
</script>
```

    - Ejemplo para mostrar la fecha actual

1.- Manejo del tiempo en JavaScript

- Horas/minutos/segundos
  - Métodos especiales
    - getHours() : Método que extrae las horas del objeto Date actual
    - getMinutes() : Método que extrae los minutos del objeto Date actual
    - getSecons() : Método que extrae los segundos del objeto Date actúal

```
<div id="laHora"></div>
  <script>
    var miFecha = new Date();
    var texto = document.getElementById("laHora");
   texto.innerHTML = miFecha.getHours() + ":" + miFecha.getMinutes() + miFecha.get-
Seconds();
  </script>
```

1.- Manejo del tiempo en JavaScript

- Problema:
  - Las funciones getHours,Getmenutes y getSeconds no devuelven dos dígitos siempre
    - Ejemplo segundos 3 => getSecond devolverá 3 y no 03 para una correcta visualización

Ejercicio 1 :

Implementar el código js para la mostrar correctamente la hora con dos dígitos cada valor

Ejercicio 2 :

Implementar una función que determine la hora en formato 12 horas y no 24

1.- Manejo del tiempo en JavaScript

- Otros métodos importantes:
  - getDate() : devuelve el día del mes (1 y 31)
  - getFullYear() : devuelve el año
  - getMiliseconds() : devuelve los milisegundos (de 0 a 999)
  - getMonth() : devuelve el mes
    - **Ojo!!!!!! De 0 a 11**
  - getDay() : devuelv el día de la semana
    - enumerados de 0 a 6 (0 es domingo y el 6 sábado)
  - getDateString() convierte la fecha del objeto Date en una cadena de caracteres

1.- Manejo del tiempo en JavaScript

- Funciones SetTimeOut y SetInterval
  - setTimeout (nombreFuncion, milisegundos).
    - Ejecutará la función **nombreFuncion** transcurrido el tiempo indicado en el segundo parámetro.
    - El tiempo se expresa en milisegundos.
    - Es importante introducir en el primer parámetro solamente el nombre de la función sin paréntesis.
  - setInterval (nombreFuncion, milisegundos).
    - En este caso, se ejecutará la función **nombreFuncion** de manera periódica según los milisegundos introducidos en el segundo parámetro.
  - clearInterval().
    - Para la ejecución iniciada con setInterval().
  - clearTimeout().
    - Para la ejecución iniciada con setTimeout().

```
var elCrono = setInterval(crono, 1000);
clearInterval(elCrono);
```

```
var elTemporizador = setTimeout(laFuncion, 5000);
clearTimeout (elTemporizador);
```

1.- Manejo del tiempo en JavaScript

```
function crono() {
    var elCrono;
    var miFecha = new Date();
    var horas = miFecha.getHours();
    var minutos = miFecha.getMinutes();
    var segundos = miFecha.getSeconds()
    if (horas > 12) {
        ampm = "pm";
        horas -= 12;
    } else {
        ampm = "am";
    }
    if (horas < 10) { horas = '0' + horas; }
    if (minutos < 10) { minutos = '0' + minutos; }
    if (segundos < 10) { segundos = '0' + segundos; }
    var texto = document.getElementById("laHora");
    texto.innerHTML = horas + ":" + minutos + ":" + segundos + ampm;

}

window.onload = function(){
    elCrono = setInterval(crono,1000);
}
```

2.- Cookies

- Las cookies sirven para recordar información del usuario.
  - Son pequeños ficheros que se almacenarán en una ruta determinada en el equipo
  - Almacenan pares clave = valor como, por ejemplo:

    usuario = Dimas
  - Cuando el navegador pide una página web, las cookies asociadas a dicha página web se envían también en la petición
    - De esta manera, los servidores tendrán vigilado al usuario
- Resumen :
  - Las cookies guardan información de las visitas del usuario
  - Ejemplos:
    - Mantener iniciada tu sesión
    - Recordar tus preferencias del sitio
    - Ofrecerte contenido relevante según tu zona

2.- Cookies

- Características de las cookies
  - **Monitorizar la actividad de los usuarios.**
    - Esta es una de las opciones más controvertida, pues consiste en monitorizar patrones de actividad, gustos, navegación, etc.
    - Muchas empresas utilizan esta técnica. Utilizar las cookies para estos propósitos hace que los usuarios se sientan espiados.
  - **Para mantener opciones de visualización o de aspecto para el usuario.**
    - Cuando un usuario quiere ver unas cosas en alguna página web y otras no, pueden utilizarse cookies para este propósito.
  - **Almacenar variables en el lado del cliente.**
    - Es conocido que, después de terminar la sesión, los datos desaparecen para el servidor, salvo que se utilicen cookies.
  - **Identificación o autenticación.**
    - Las cookies tienen un periodo de tiempo de validez, durante el cual se puede verificar cuándo un usuario se autentica en el sistema por primera vez.

2.- Cookies

- ¿Cómo crear una cookie?
  - Bastaría con una simple línea : ***document.cookies="usuario=Monica"***
    - El valor de la cookie debe estar perfectamente codificado ya que se envía en la cabecera HTTP
    - Para evitar problemas deberí**amos programar una correcta codificación**

      var usuarioCookies="monica";

      document.cookies="usuario"+encodeURIComponent(usuarioCookies);
    - **encodeURIComponent**() y **decodeURIComponent**() codifican y decodifican en HTML los caracteres especiales como ,/?:08=+$ +.
- Las cookies tienen una **fecha de caducidad**
  - Por tanto, las anteriores cookies se eliminarán al cerrarse el navegador.
  - Si se desea que perduren se ha de añadir una fecha de caducidad:

```
<button type="button" onclick= "document.cookies='usuario=Monica; expires=Dom, 28 Jul 2024 23:59:00 GMT'">
        Boton crear cookies
</button>
```

2.- Cookies

- Proceso para crear cookies

1. **Establecer el nombre y valor de la cookie**
   - Nombre : es la etiqueta que identifica la información
   - Valor : es la información que deseamos guardar (clave-valor)
     - Puede ser una cadena de texto , un numero etc
2. **Asignar la cookie al navegador**
   - Utilziamos el objeto **document.cookies** para asignar la cookie al navegador
     - Document.cookies=<<nombre-valor>>;

2.- Cookies

- Proceso para crear cookies

3. **Configurar otros atributos**
   - expires:
     - Define la fecha de expiración de la cookie. Después de esa fecha, la cookie se elimina automáticamente.
   - path:
     - Especifica la ruta del sitio web donde la cookie es válida.
   - domain: Indica el dominio en el que la cookie es válida.
   - secure: Si se establece en true, la cookie solo se enviará a través de una conexión segura (HTTPS).

2.- Cookies

- Proceso para crear cookies

Configurar otros atributos

- domain:
  - Indica el dominio en el que la cookie es válida.
- secure:
  - Si se establece en true, la cookie solo se enviará a través de una conexión segura (HTTPS).

```
document.cookie = "user=miCookie; expires=Thu, 18 Dec 2025 12:00:00 UTC; path=/;";
```

```
<body>
    <script>
        document.cookie = "user=Iker; expires=Thu, 18 Dec 2025 12:00:00 UTC; path=/;";
    </script>
</body>
```

2.- Cookies

- Acceso a los datos después de crear cookies
  - Accedemos mediante el objeto **document.cookies**
  - Devuelve todas las cookies almacenadas en el sitio web como una única cadena
  - Para obtener los datos específicos necesitamos técnicas de manipulación de cadenas

```
<script>
        //CREAR COOKIE
        document.cookie = "user=Iker; expires=Thu, 18 Dec 2025 12:00:00 UTC; path=/;";

        //ACCEDER A LA COOKIE Y MOSTRAR
        function getCookieValue(nombre) {
            var cookies = document.cookie.split(";"); // Divide la cadena de cookies en un array
            for (var i = 0; i < cookies.length; i++) {
                var cookie = cookies[i].trim(); // Elimina los espacios en blanco al principio y al final
                if (cookie.startsWith(nombre + "=")) {
                    return cookie.substring(nombre.length + 1); // Retorna el valor de la cookie
                }
            }
            return null; // Si no se encuentra la cookie, retorna null
        }

        var nombreCookie = getCookieValue("user");
        console.log(nombreCookie); // Muestra el valor en la consola
</script>
```

2.- Cookies

- IMPORTANTE!!!!
- El navegador en versión escritorio no permite el uso y almacenaje de cookies.
- Necesitaréis subir vuestra solución a un hosting mediante FTP parar poder practicar con las cookies

3.- Almacenamiento Local

- Con HTML5, el uso de cookies paso a un segundo lugar dando origen al **almacenamiento local.**
  - El almacenamiento en el lado del cliente solo era posible a través de cookies
- El almacenamiento local es un sistema más sencillo y eficiente que las cookies.
  - Los navegadores son capaces de almacenar hasta 5MB de información, de esta manera conseguimos no sobrecargar al servidor enviando y recibiendo datos
  - IDEA: dentro de un mismo dominio y un protocolo, la información es compartida por todas y cada una de las paginas

3.- Almacenamiento Local

Características del localStorage

- **Persistencia:**
  - Los datos se mantienen almacenados hasta que se eliminen manualmente
- **Capacidad:**
  - Aunque el limite puede variar entre navegadores, generalmente puedes almacenar hasta 5MB de datos.
- **Acceso entre pestañas:**
  - Los datos almacenados en **localSorage** están disponibles para todas las pestañas y ventanas que comparten el mismo origen.

3.- Almacenamiento Local

Objeto localstorage

- Permite guardar y recuperar información en el navegador sin importar que sea otra sesión.
- Su utilización es similar a las cookies:
  - **setItem(clave,valor):** Se emplea para guardar información y tiene dos parámetros que son la clave y el valor
  - **getItem(clave):** Para recuperar información, habrá que especificar la clave para poder recuperar el valor de esta

```
<script>
    localStorage.setItem("usuario", "Monica");
    alert(localStorage.getItem("usuario"));
</script>
```

  - **removeItem(clave):** Para eliminar un dato del almacenamiento local
  - **clear()** elimina todos los ítems alacenados

3.- Almacenamiento Local

- Objeto **sessionStorage**
  - Objeto creado por HTML5 para guardar datos durante una sesión y no de forma permanente.
  - Se utiliza de la misma manera que localStorage
    - Pero con la particularidad de que los datos solo se almacenan durante la sesión
- Diferencias:
  - **localStorage:** Los datos persisten indefinidamente hasta que se eliminen explícitamente
    - Los datos están disponibles incluso después de cerrar el navegador
  - **sessionStorage:** los datos se eliminan cuando la sesión del navegador finaliza
    - Cuando se cierra la pestaña o la ventana del navegador.

3.- Almacenamiento Local

- Ejemplo : guardar preferencias del usuario
  - Almacenamos el tema de la pagina (oscuro o claro

```
// Guardar la preferencia del tema
localStorage.setItem('tema', 'oscuro');

// Leer la preferencia del tema
var tema = localStorage.getItem('tema');
if (tema === 'oscuro') {
    document.body.classList.add('oscuro');
}
```

3.- Almacenamiento Local

- Beneficios:
  - **Mejora la experiencia del usuario**
    - Los usuarios no tienen porque reingresar sus preferencias o datos cada vez que visitan el sitio web
    - Esto mejora la experiencia de usuario
  - **Eficiencia y rendimiento**
    - Guardar datos localmente reduce la necesidad de realizar solicitudes repetidas al servidor
    - Conseguimos una mejora de rendimiento de la aplicación al reducir la carga en le servidor
  - **Facilidad de uso**
    - La API de localStorage es muy sencilla y fácil de usar
    - Es accesible para cualquier tipo de desarrollador
  - **Seguridad**
    - Los datos almacenados en localStorage solo son accesibles desde el mismo sitio.
    - Es recomendable encriptar datos sensibles

3.- Almacenamiento Local

- Buenas practicas y particularidades
  - **Limite de almacenamiento**
    - Tener en cuenta que el almacenamiento máximo en el lado del cliente es de 5MB
    - Almacenar datos relativamente pequeños
  - **Seguridad de los datos**
    - No se debe almacenar información sensible como contraseñas o datos personales sin aplicar métodos de encriptado
  - **Manejo de errores**
    - Manejar posibles errores como exceder el limite de almacenamiento, para asegurarnos que la aplicación maneja estas situaciones de manera "elegante"

```
try {
    localStorage.setItem('miClave', 'miValor');
} catch (e) {
    if (e.code === 22) {
        console.error('Límite de almacenamiento excedido');
    }
}
```