DWEC
TEMA 3

MODELOS DE OBJECTOS PREDEFINIDOS EN JAVASCRIPT

P E G O
IES Enric Valor

GENERALITAT VALENCIANA
CONSELLERIA D'EDUCACIÓ, CULTURA I ESPORT

Unió Europea
Fons Social Europeu
El FSE inverteix en el teu futur

ÍNDICE

3.1 INTRODUCCIÓN

En la unidad anterior hemos realizado un repaso de la sintaxis del lenguaje JS. En esta unidad se va ha profundizar en conceptos como los objetos y funciones.
Otro concepto que se va ha trabajar es el almacenamiento local y las cookies, con estos objetos el navegador podrá almacenar información del usuario.
Además de lo comentado anteriormente, es importante que el programador sepa manejarse con las fechas y los tiempos: con el objeto Date.

3.2 MANEJO DE TIEMPOS EN JS

JS es la manera más útil de trabajar con el tiempo en un navegador, dado que no recarga el servidor y tiene el objeto Date. Además, permite actualizar el reloj mediante algunas funciones que se detallan en este apartado.
El objeto Date se creó en JavaScript para almacenar la fechas y horas. Una vez creado dicho objeto, es posible modificarlo y realizar los cálculos que el programador crea convenientes para modificar las fechas y las horas.
La creación de este objeto se podría hacer de la siguiente forma:

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

La variable miFecha contendrá la fecha y hora actual. Si se desea mostrar la fecha y hora en un campo de la página web:

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

3.2 MANEJO DE TIEMPOS EN JS

El problema que hay es que el aspecto que se muestra no es el que normalmente no interese mostrar, por tanto, hay que utilizar algunos métodos para mostrar la hora, los minutos y los segundos como queremos:

- getHours().
- getMinutes().
- getSeconds().

El siguiente ejemplo muestra la hora en formato convencional:

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

3.2 MANEJO DE TIEMPOS EN JS

Hay que tener en cuenta que estas funciones pueden no devolver **2 dígitos**, con lo cual habrá que modificar o mejorar el código anterior:

```
<div id="laHora"></div>
<script>
    var miFecha = new Date();
    var horas = miFecha.getHours();
    var minutos = miFecha.getMinutes();
    var segundos = miFecha.getSeconds();
    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;
</script>
```

3.2 MANEJO DE TIEMPOS EN JS

Las funciones setTimeout y setInterval

JS ofrece las siguientes funciones:

- *setTimeout (Función_a_llamar, milisegundos)*. Ejecutará la función *Funcíón_a_llamar* transcurrido el tiempo indicado en el segundo parámetro. Se debe de introducir la función sin parentesis.
- *setInterval (Función_a_llamar, milisegundos).* Función parecida a la anterior, pero, en este caso, se ejecutará la función Función_a_llamar de manera periódica según los milisegundos introducidos en el segundo parámetro.
- *clearInterval()*. Para la ejecución iniciada con setInterval(). Ver ejemplo de uso:
- *clearTimeout()*. Para la ejecución iniciada con setTimeout().

3.2 MANEJO DE TIEMPOS EN JS

Las funciones setTimeout y setInterval

Ejemplo:

```
<!DOCTYPE html>
<html>
<head>
<title>Reloj</title>
</head>
<body>
<div id="laHora"></div>
<script>
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);
}
</script>
</body>
</html>
```

3.2 MANEJO DE TIEMPOS EN JS

Las funciones setTimeout y setInterval

Actividad 3.2.1

Basándote en el código anterior, crea tu propio cronometro.

Actividad 3.2.2

Realiza un script que nos retorne la fecha con el siguiente formato y haz que se refresque cada segundo:

10:35:54 - 29/09/2020

3.3 COOKIES

Generalmente cuando se navega, supuestamente, es de forma anónima, pero, en muchos de los casos, esto no es necesariamente útil para el servidor.

Por ejemplo, en una página web de compras cuando tenemos el carrito con artículos y cerramos el navegador, si volvemos a entrar después de un tiempo, es recomendable que el carrito esté guardado y se pueda recuperar.

Por lo tanto, las cookies sirven para recordar información del usuario. Son pequeños ficheros que se almacenarán en una ruta determinada del equipo y que contienen pares clave = valor como, por ejemplo:

```
usuario = Dimas
```

Cuando el navegador pide una página web, las cookies asociadas a dicha página web se envían en una petición y, de esa manera, los servidores tendrán *vigilado* al usuario.

https://www.php.net/manual/es/function.session-get-cookie-params.php

3.3 COOKIES

Ejemplos de uso 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. Utilizar las cookies para estos propósitos hace que los usuarios se sientan espiados.
- *Para mantener opciones de visualización o de aspecto para el usuario.* Las cookies pueden guardarse criterios del aspecto de la web que el usuario haya configurado.
- *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 las cookies.
- *Identificación o autentificación*. Las cookies tienen un periodo de validez, durante el cual se puede verificar cuando un usuario se autentifica en el sistema por primera vez.

3.3 COOKIES

- Crear cookie:

Para crear una cookie solo necesitamos asignar su valor al método document.cookie:

```
document.cookie=”usuario=Dimas”;
```

Si la cookie va ha viajar al servidor, en la cabecera HTTP del documento, debemos de evitar problemas con los caracteres especiales. Utilizando encodeURIComponent() y decodeURIComponent().

```
var usuarioCookie=”Dimas”
document.cookie=”usuario=”+ encodeURIComponent(usuarioCookie);
```

Las cookies tienen una fecha de caducidad y si se desea que perdure se ha de añadir una fecha de caducidad:

```
document.cookie="usuario=Dimas;expires=Sat, 6 Aug 2020 12:15:00 GMT";
```

3.3 COOKIES

- Leer cookie:

  Pare leer la cookie podemos utilizar el siguiente código:

```
<button type=”button” onclick=’alert(document.cookie)’>Ver las Cookies</
button>
```

  Para modificar una cookie, basta con crearla de nuevo.

- Eliminar cookie:

  Si lo que se quiere hacer es eliminar el contenido de la cookie deberemos de utilizar el siguiente comando:

```
document.cookie = "usuario= ; max-age=0;";
```

3.4 ALMACENAMIENTO LOCAL

Como se ha visto en el apartado 3.3, el almacenamiento en el lado del cliente solamente es posible a través de cookies. Pero, tras la aparición de HTML5, las cookies pasaron a un segundo plano para dar paso al almacenamiento local. EL almacenamiento local es un sistema más sencillo y eficiente que las antiguas cookies.
(ojo: El tamaño máximo del almacenamiento local es de 5 MB)

El objeto LocalStorage

Este objeto permite guardar y recuperar información en el navegador sin importar que sea otra sesión. Se utiliza con los métodos:

a) ***setItem(clave,valor)***. Se emplea para guardar información.
b) ***getItem(clave)***. Para recuperar la información de dicha clave.
c) ***removeItem(clave).*** Para eliminar los datos de la clave.

3.4 ALMACENAMIENTO LOCAL

Ver ejemplo:

```
<html lang="en">
<head>
</head>
<body>
    <script>
        localStorage.setItem("usuario","Dimas");
        alert ( localStorage.getItem("usuario"));
        localStorage.removeItem("usuario");
    </script>
</body>
</html>
```

Existe la posibilidad que solo se quiera guardar datos sobre la sesión, HTML5 ha creado el objeto ***sessionStorag***e que se utiliza de la misma manera que el localStorage.

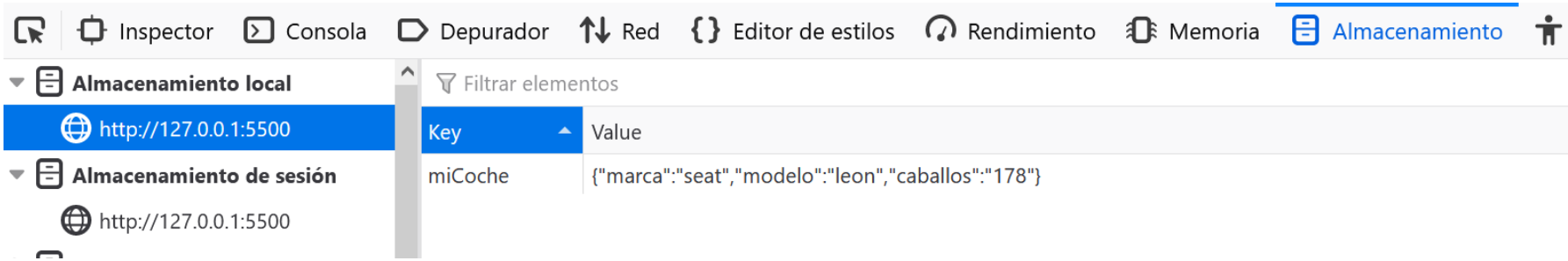
3.4 ALMACENAMIENTO LOCAL

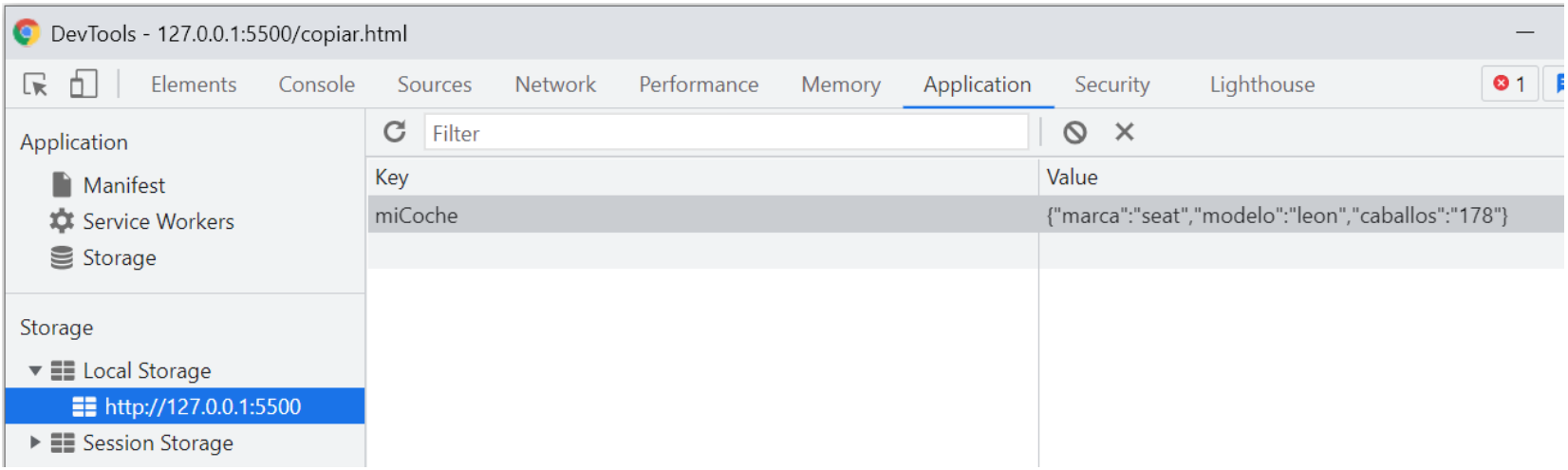
Hay que tener en cuenta que si vamos a guarda en el localStorage un objeto deberemos de utilizar las siguientes funciones predefinidas de JS para convertirlo en JSON i al contrario:
Ver ejemplo:

```
<html lang="es">
<head>
</head>
<body>
    <script>
        let coche ={
            marca: "seat",
            modelo: "leon",
            caballos: "178"
        }
        localStorage.setItem("miCoche",JSON.stringify(coche));
        console.log(JSON.parse(localStorage.getItem("miCoche")));
    </script>
</body>
</html>
```

3.4 ALMACENAMIENTO LOCAL

Las herramientas del programador del navegador nos van a ayudar a saber que hay guardo en nuestro localStorage.





3.5 OBJETOS EN JAVASCRIPT

En JavaScript, prácticamente todo es un objeto, las fechas, las expresiones regulares, los arrays, las funciones, etc.

En JS, pueden crearse objetos de dos formas como se puede ver a continuación:

Primera forma

```
var coche = {
    modelo:"Mercedes C320",
    color:"azul",
    kms:15000,
    combustible:"diésel"
    };
```

Segunda forma

```
var coche = new Object();
coche.modelo = "Mercedes C320";
coche.color = "azul";
coche.kms = 15000;
coche.combustible = "diésel";
```

3.5 OBJETOS EN JAVASCRIPT

Recorrer la información de un objeto

```
var usuario={nombre:"Felipe",apellido:"Ranas",edad:30,esAdmin:true};
for(campo in usuario){
    alert(campo);
    alert(usuario[campo]);
}
```

Acceso a los campos de un objeto

```
var usuario ={
    nombre: "Dinmas",
    apellidos:{ primer: "Perez", segundo: "Garcia"},
    edad: "38",
    esAdmin: true
}
// las dos líneas mostraran la misma información ( Perez)
alert(usuario.apellidos.primer);
alert(usuario["apellidos"]["primer"]);
```

3.5 OBJETOS EN JAVASCRIPT

Constructores de JavaScript

Hasta ahora no habíamos visto métodos de objetos en JS, el siguiente código muestra un constructor para la clase coche:

```
function coche(modelo, color, kms, combustible) {
    this.modelo = modelo;
    this.color = color;
    this.kms = kms;
    this.combustible = combustible;
    }
    var elmio = new coche("Mercedes E330", "negro", 120000, "diésel");
    var eltuyo = new coche("BMV 318", "blanco", 210000, "gasolina");
```

3.5 OBJETOS EN JAVASCRIPT

Constructores de JavaScript

Un objeto se caracteriza por tener un estado (atributos) y un comportamiento (métodos). A continuación se muestran algunos métodos para la clase coche creada anteriormente.

```
function coche(modelo, color, kms, combustible) {
    this.modelo = modelo;
    this.color = color;
    this.kms = kms;
    this.combustible = combustible;
    this.setmodelo = function (nuevomodelo) {
        this.modelo = nuevomodelo;
    }
    this.getmodelo = function () {
        return this.modelo;
    }
}
```

3.6 FUNCIONES EN JAVASCRIPT

Como se ha dicho anteriormente, las funciones en JS son objetos y en el tema 2, se ha explicado el funcionamiento de las funciones con el ejemplo

```
function suma(a, b) {
    return a + b;
}
```

Así pues esta función se puede crear mediante un constructor llamado *Function()* de la siguiente ,manera:

```
var suma = new Function("a", "b", "return a + b");
```

La llamada a la función seria la misma en los dos casos

```
var c = suma(2,2);
```

3.6 FUNCIONES EN JAVASCRIPT

La recursividad en JavaScript

La recursividad es una función o algoritmo que recursivo que genera la solución realizando llamadas a sí mismo.

En JS es posible utilizar la recursividad para solucionar problemas. Como por ejemplo el siguiente código:

```
<!DOCTYPE html>
<html>
    <head><title>Ejemplo de recursividad</title></head>
    <body>
        <script>
            function factorial(num){
                if(num == 0){
                    return 1;
                }else{
                    return (num * factorial(num -1));
                }
            }
            var numero = factorial(10);
            document.write(numero);
        </script>
    </body>
</html>
```

https://www.sangakoo.com/es/temas/el-factorial-de-un-numero

3.6 FUNCIONES EN JAVASCRIPT

Los parámetros de las funciones

En JS, es posible llamar a una función con memos parámetros de los previstos, No dará error en la ejecución peró hay que prever que hará la función.

```
function suma(a, b) {
    if (b === undefined){ //en el caso que no se introduzca segundo parámetro
    return a + a;
    }
    return a + b;
}
// llamada normal
var c = suma(2,2);
// llamada con un parámetro solo
var c = suma(2);
```

3.6 FUNCIONES EN JAVASCRIPT

Funciones y métodos en objetos de tipo array

Los arrays son uno de los objetos que tiene más método (función) para su tratamiento. Los métodos más comunes son:

- *Concat()*. Concatena y devuelve una copia de los arrays concatenados.
- *Fill().* Rellena un array con un valor suministrado.
- *Filter()*. Devuelve un array con los elementos que pasen el test suministrado por parámetro, ejemplo:

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];

const result = words.filter(word => word.length > 6);

console.log(result);
// expected output: Array ["exuberant", "destruction", "present"]
```

3.6 FUNCIONES EN JAVASCRIPT

Funciones y métodos en objetos de tipo array.

- *Find().* Devuelve el valor del primer elemento que pase el test suministrado por parámetro.

```
const array1 = [5, 12, 8, 130, 44];

const found = array1.find(element => element > 10);

console.log(found);
// expected output: 12
```

- forEach(). Realiza una llamada a una función por cada uno de los elementos de un array. Ejemplo:

```
const array1 = ['a', 'b', 'c'];

array1.forEach(element => console.log(element));

// expected output: "a"
// expected output: "b"
// expected output: "c"
```

3.6 FUNCIONES EN JAVASCRIPT

Funciones y métodos en objetos de tipo array.

- *includes().* Comprueba si en el array está un elemento.
- *indexOf().* Busca en el array un elemento y devuelve su posición.
- *isArray().* Comprueba si un objeto determinado es un array.
- *lastIndexOf().* Devuelve la posición de un element de un array. Pero comenzadon desde atrás.
- *Pop()*. Elimina un elemento de la última posición.
- *Push()*. Añade un elemento nuevo al array en su última posición.
- *Shift().*Elimina el primer elemento del array i devuelve ese elemento para ser tratado.
- *slice().* Selecciona parte del array.
- *sort().* Ordena los elementos de array.
- *splice()*. Metodo para añadir o eliminar elementos del array.
- *toString().* Convierte un array a un string.
- *vauleOf().* Devuelve los valores de un array. Para hacer copia de un array.

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach?v=example

3.6 FUNCIONES EN JAVASCRIPT

Funciones y métodos en objetos y variables string.

- *Saber la longitud de un string.*

```
var web = "myfpschool.com";
var longitud = web.length; //longitud valdrá 14
```

- Conocer la posición de una cadena dentro de otra.

```
var web = "La web myfpschool es una de las mejores en tecnología";
var posicion = web.lastIndexOf("myfpschool"); //posición valdrá 7
```

- search(string1). Busca en un string un determinado valor (o expresión regular) y devuelve la posición donde la encontró.
- slice(inicio, fin). Extrae parte de un string y devuelve el nuevo string.
- substr(inicio,fin). Extrae caracteres de un string comenzando en la posición específica en el primer parámetro con una longitud expresada en el segundo parámetro.
- Replace(string1,string2). Reemplaza en un string una cadena de caracteres (o expresión regular) por la cadena expresada en el segundo parámetro.
- toUpperCase(). Convierte la cadena a mayúsculas
- toLowerCase(). Convierte la cadena a minúsculas.
- concat (string1, string2). Concatena dos o mas cadenas devolviendo el string resultante.

MANUAL DE REFERENCIA :
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

3.6 FUNCIONES EN JAVASCRIPT

Funciones y métodos en objetos y variables string. Ejemplos:

```
var stringSearch = "Hola Mundo";
console.log(stringSearch.search("Mun"));
// mostrará por consola 5

console.log(stringSearch.slice(2, stringSearch.length));
// mostrará por consola la Mundo

console.log(stringSearch.substring(1, 4));
// mostrará por consola ola
console.log(stringSearch.substr(1, 4));
// mostrará por consola  ola

console.log(stringSearch.replace("a", "X"));
// mostrará por consola HolX Mundo

console.log(stringSearch.toUpperCase());
// mostrará por consola HOLA MUNDO
```

3.6 FUNCIONES EN JAVASCRIPT

Funciones y métodos en objetos y variables string. Ejemplos:

```
console.log(stringSearch.toLowerCase());
// mostrará por consola hola mundo

var cad = "mmm";
console.log(stringSearch.concat(cad));
// mostrará por consola Hola Mundommm

console.log(stringSearch.charAt(5));
// mostrará por consola M

console.log(stringSearch.split(" "));
// Creará el Array [ "Hola", "Mundo" ]. Se ha utilizado como separador
el espacio

console.log(stringSearch.repeat(2));
// mostrará por consola Hola MundoHola Mundo
```

3.6 FUNCIONES EN JAVASCRIPT

Funciones globales del lenguaje JS de números:

Las siguientes funciones son útiles cuando se validan formularios o campos especiales. Es conocido que **HTML5** ya provee al programador de ciertas validaciones nativas, pero, en ocasiones, se necesita una validación más sofisticada.

1-. isFinite(). Evalúa si un dato determinado es valor finito.

2-. isNaN(). Evalúa si un dato determinado no es numérico.

```
isNaN(456) //devuelve false porque es un número
isNaN(0) //devuelve false
isNaN(-3.14) //devuelve false
isNaN(9+4) //devuelve false porque al evaluar la expresión el resultado
es numérico
isNaN('666') //devuelve false porque al evaluar el contenido el resultado
es numérico
isNaN('') //devuelve false
isNaN(true) //devuelve false
isNaN(8/0) //devuelve false
isNaN(0/0) //devuelve true
isNaN('Sintesis') //devuelve true al ser un string
isNaN('2005/12/12') //devuelve true al ser una fecha
isNaN(undefined) //devuelve true
isNaN('NaN') //devuelve true
isNaN(NaN) //devuelve true
```

3.6 FUNCIONES EN JAVASCRIPT

Funciones globales del lenguaje JS de números:

3-. Number(). Convierte un dato en un número.

```
Number(true) // devuelve 1
Number(false) // devuelve 0
Number("666") // devuelve 666
```

2-. parseFloat(). Analiza un string y devuelve un número en coma flotante.

```
parseFloat("10") //devuelve 10
parseFloat("3.14") // devuelve 3.14
parseFloat("   100   ") // devuelve 100
parseFloat("80 monos") // devuelve 80
```

3-. parseInt(). Analiza un string y devuelve un número entero.

```
parseInt(«10») //devuelve 10
parseInt(«3.14») // devuelve 3
parseInt(«   100   «) // devuelve 100
parseInt(«80 monos») // devuelve 80
```

4-. String(). Convierte un dato en un string.