

	Universidad Don Bosco Facultad de Ingeniería Escuela de Computación
Ciclo II - 2022	Guía de laboratorio N° 10
	Nombre de la práctica: JSON, localStorage y sessionStorage. Lugar de ejecución: Centro de computo Materia: Lenguaje Interpretado en el Cliente.

I. Objetivos.

- ✓ Adquirir dominio en la utilización de los principales métodos que ofrece JavaScript para trabajar con objetos JSON.
- ✓ Comprender las opciones para trabajar con almacenamiento local tanto con localStorage como con sessionStorage entendiendo sus diferencias.
- ✓ Utilizar las propiedades y métodos para el trabajo con los objetos localStorage y sessionStorage.

II. Introducción Teórica.

➤ JSON (Notación de objetos de JavaScript).

JSON es una manera más simple de representar objetos de JavaScript como cadenas, es una alternativa más simple a XML para pasar datos entre el cliente y el servidor. En JSON, cada objeto se representa como una lista de nombres de propiedad y valores contenidos entre llaves, en el siguiente formato:

{“NombrePropiedad”:valor1, “NombrePropiedad2”:valor2}

Los arreglos en json se representan en JSON mediante corchetes en el siguiente formato:

[valor1,valor2,valor3]

Cada valor puede ser una cadena, un número, una representación de JSON de un objeto, **true**, **false** o **null**. Podemos convertir cadenas JSON en objetos de javascript mediante la función `JSON.parse` de JavaScript. Las cadenas JSON son más fáciles de crear y analizar que XML, además de que requieren menos bytes. Por estas razones. JSON se utiliza con frecuencia para comunicarse en la interacción cliente-servidor.

```
//Definiendo la variable para manejar el objeto JSON
var objetoJSON;
//Asignando los valores para el objeto JSON
objetoJSON = {
    "foto" : "img/avatarjorge.jpg",
    "nombre" : "Jorge",
    "apellido" : "Pérez",
    "edad" : 32,
    "profesion" : "Ingeniero"
}

document.getElementById("content").innerHTML =
    "<div>\n<img src=\"\" + objetoJSON.foto + \"\" title=\"Avatar de Jorge\" />\n</div>\n" +
    "<ul>\n" +
    "<li>\nNombre: " + objetoJSON.nombre + "</li>\n" +
    "<li>\nApellido: " + objetoJSON.apellido + "</li>\n" +
    "<li>\nEdad: " + objetoJSON.edad + "</li>\n" +
    "<li>\nProfesión: " + objetoJSON.profesion + "</li>\n" +
    "</ul>\n";
```

Métodos para trabajar con JSON

Para trabajar más fácilmente con objetos JSON y un lenguaje de programación como JavaScript se utilizan dos métodos nativos que facilitan la utilización de estos objetos para distintos propósitos:

JSON.parse(): Este método analiza una cada de texto y la transforma en un objeto JSON a partir de pares **property:value**; contenidos en la cadena.

JSON.parse(cadena[, transformador]);

- ✓ La **cadena** son transformados de acuerdo a lo establecido en esta función antes de ser asignados como valores del objeto JSON.

- ✓ El **transformador** retorne *undefined* al intentar transformar el valor de una propiedad, dicha propiedad será eliminada del objeto.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title></title>
  <meta charset="utf-8" />
  <script type="text/javascript">
    //Ejemplo muy básico
    /* var objetoJSON = JSON.parse('{ "name": "Roberto Murcia" }');
    console.log(objetoJSON.name); */

    //Ejemplo con función transformadora
    var textoJSON = '{ "usb" : 20, "mouse" : 30, "mousepad": 10, "keyboard" : 12.5 }',
    //Esta es la función de transformación
    precioConIVA = function(producto, precio){
      if(producto === "mousepad"){
        // No lo considero en el nuevo objeto
        return undefined;
      }
      // Última confirmación
      if(producto === ""){
        // Retornamos el nuevo objeto
        return precio;
      }
      // Manipulamos a todos los demás valores
      return (precio + (precio * 0.13));
    },
    objetoJSON = JSON.parse(textoJSON, precioConIVA);
    console.log(objetoJSON); // logs Object {usb: 23.6, mouse: 35.4, keyboard: 12}
  </script>
</head>
<body>
</body>
</html>
```

JSON.stringify(): Este método recibe un objeto JavaScript y devuelve su JSON equivalente y puede tomar hasta tres parámetros.

JSON.stringify(objeto[, reemplazador[, separador]]);

El método **JSON.stringify()** recibe un **objeto** JavaScript como primer argumento y devuelve su **JSON equivalente**. El primer argumento es el único obligatorio y en la práctica es un objeto o matriz (arreglo), aunque también puede ser una cadena, booleano, número o valor nulo.

El parámetro opcional **reemplazador**, es una función que modifica la forma en que los objetos y los arreglos se encadenan. El parámetro opcional **separador**, es una cadena o número usado para insertar caracteres o espacios en blanco en la cadena de texto JSON, permitiendo una mejor legibilidad.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <title></title>
  <meta charset="utf-8" />
  <script type="text/javascript">
    //Ejemplo básico del método stringify()
    //var objetoJavaScript = {"clave" : "valor"},
    //textoJSON = JSON.stringify(objetoJavaScript);
    //console.log(typeof textoJSON); // Imprimirá "string"
    //console.log(textoJSON); // Imprimirá {"clave":"valor"}
    //Uso con función reemplazador como filtro
    //var objetoJavaScript = {"hora": 9, "dia": 31, "mes": 7, "año": 2014},
    //textoJSON = JSON.stringify(objetoJavaScript , ["año", "mes", "dia"]);
    //console.log(textoJSON); // Imprimirá '{"año" : 2014, "mes": 7}'
    //Uso con función reemplazador para quitar la clave "hora"
    //de la cadena JSON resultante y añadir con el argumento
    //separador 4 espacios
    var filtro = function (clave, valor) {
      if(clave === "hora") {
        return undefined;
      }
      return valor;
    }
    var objetoJavaScript = {"hora": 10, "dia": 15, "mes": 10, "año": 2014},
    textoJson = JSON.stringify(objetoJavaScript , filtro, 4);
    console.log(textoJson);
  </script>
</head>
<body>
</body>
</html>

```

localStorage.

A partir de HTML5 hay dos nuevos mecanismos para almacenar pares claves/valor que ayudan a eliminar algunos problemas con las cookies. Las aplicaciones Web pueden usar la propiedad localStorage del objeto Windows para almacenar hasta varios megabytes de datos de cadena de pares clave/valor en la computadora del usuario y pueden acceder a esos datos entre sesiones de navegación y pestañas del navegador. A diferencia de las cookies, los datos en el objeto localStorage no se envían al servidor Web con cada solicitud. El dominio de cada sitio Web tiene un objeto localStorage separado: todas las páginas de un dominio dado comparten un objeto localStorage. Por lo general se reservan 5MB para cada objeto localStorage, pero un navegador Web puede preguntar al usuario si debe asignar más espacio cuando este llene.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<script>
function clickCounter() {
  if(typeof(Storage) !== "undefined") {

    if (localStorage.clickcount) {
      //Ingresando datos en el Store
      localStorage.clickcount = Number(localStorage.clickcount)+1;
    } else {
      localStorage.clickcount = 1;
    }

    //Mostrar datos almacenados en el Store
    document.getElementById("resultado").innerHTML = "Ha hecho clic en el botón" + localStorage.clickcount + " veces.";
  } else {
    document.getElementById("resultado").innerHTML = "Lo siento, su navegador no soporta Storage...";
  }
}
</script>
</head>
<body>
<p><button onclick="clickCounter()" type="button">Clickame!</button></p>
<div id="resultado"></div>
<p>Haga clic en el botón para ver el incremento del contador.</p>
<p>Cierre la pestaña del navegador (o ventana), y vuelva a intentarlo, y el contador seguirá contando (no se restablece).</p>
</body>
</html>

```

sessionStorage.

Las aplicaciones Web necesitan acceso a los datos solo para una sesión de navegación y que deben mantener esos datos separado entre múltiples pestañas pueden usar la propiedad sessionStorage del objeto Window. Hay un objeto sessionStorage separado por cada sesión de navegación, incluyendo pestañas separadas que accedan al mismo sitio Web.

```
<!DOCTYPE html>
<html>
<head>
<script>
function clickCounter() {
  if(typeof(Storage) !== "undefined") {
    if (sessionStorage.clickcount) {
      //ingresando sesión
      sessionStorage.clickcount = Number(sessionStorage.clickcount)+1;
    } else {
      sessionStorage.clickcount = 1;
    }
    //Mostrar datos almacenados en el Sesión
    document.getElementById("resultado").innerHTML = "Ha hecho clic en el botón " + sessionStorage.clickcount + " veces.";
  } else {
    document.getElementById("resultado").innerHTML = "Lo siento, su navegador no soporta Storage...";
  }
}
</script>
</head>
<body>
<p><button onclick="clickCounter()" type="button">Clickame!</button></p>
<div id="resultado"></div>
<p>Haga clic en el botón para ver el incremento del contador.</p>
<p>Cierre la pestaña del navegador (o ventana), y vuelva a intentarlo, y el contador se pone a cero.</p>
</body>
</html>
```

Otros métodos y propiedades

Propiedad length.

La propiedad length permite saber el número de datos (pares "clave/valor") que tenemos almacenados. su sintaxis es:

num = localStorage.length;

La variable num nos dice el número de datos que tenemos guardados.

Borrar datos.

Los datos mientras no les digamos otra cosa se almacenan de forma permanente y sin fecha de caducidad, por lo que si queremos borrar algún dato debemos hacerlo de forma manual, utilizando el método **removeItem()** de la siguiente manera:

localStorage.removeItem("clave");

En el ejemplo anterior para borrar el dato haremos lo siguiente:

localStorage.removeItem("nombre")

Debemos pasar como argumento la clave del elemento que queremos borrar. Si la "clave" que pasamos en el argumento no existe el método simplemente no hará nada.

Método key().

El método key() nos permite saber las claves que tenemos guardadas. Las claves se guardan en un array, por lo que como argumento pondremos el número de elemento de la clave.

clave = localStorage.key(0);

Este ejemplo nos devuelve la clave del primer elemento guardado.

Aunque este método no es muy útil para buscar claves, sí que puede ser útil para ver todos los datos (pares "clave/valor") que hay guardados. Para ello utilizaremos un bucle que recorra todos los datos. La propiedad **length** nos dirá cuántos elementos hay guardados y el método **key()** nos muestra los datos. El código ser parecido a éste:

```
num=localStorage.length; //número de datos guardados
info=document.getElementById("lista"); //elemento donde sacar los datos en pantalla
info.innerHTML="<ul>"; //principio de lista
for (i=0;i<num;i++) { //bucle para recorrer todos los datos
    clave=localStorage.key(i); //obtener la clave del dato
    valor=localStorage.getItem(clave); //obtener el valor del dato
    info.innerHTML+="<li>"+clave+" : "+valor+"</li>"; //sacar dato en pantalla
}
info.innerHTML+="</ul>"; //Fin de la lista
```

En este ejemplo hemos sacado todos los datos guardados en pantalla dentro de una lista.

Siguiendo este mismo ejemplo no nos sería difícil hacer un bucle que borrara todos los datos que tenemos almacenados. Para ello una vez obtenida la clave se le aplicaría el método **removeItem(clave)**.

Guardar datos que no son cadenas.

El sistema de almacenamiento de **localStorage** sólo puede guardar cadenas de texto, por lo que tenemos el problema de querer guardar otro tipo de datos como números, fechas, arrays, etc.

En este caso el navegador convertirá este dato a cadena de texto, por lo que cuando lo recojamos, debemos convertirlo al tipo de dato original para poder usarlo como tal.

Uso actual de localStorage.

LocalStorage se ha convertido en una herramienta esencial en numerosas aplicaciones web, si bien su uso se convierte en esencial en aplicaciones web móviles encapsuladas con Apache Córdova (PhoneGap), donde muchas de ellas no necesitan grandes necesidades de almacenamiento, pero sí hacer persistentes ciertas preferencias de usuario, o cachear alguna información para uso en modo offline. Gracias a **localStorage** no es necesario recurrir a plugins extras cuando las necesidades de almacenamiento no son excesivas.

Eventos para trabajar con localStorage.

También podemos detectar modificaciones en **localStorage** ya que cada vez que ejecutamos funciones como **setItem**, **removeItem** o **clear**, se dispara el evento 'storage', pudiendo establecer un listener sobre este:

```
<script>
function reportStorage(evt) {
    alert("El `Storage` fue modificado desde: " + evt.url);
    alert("Se ha actualizado: " + evt.key + 'r' + "Antes: " + evt.oldValue + 'r' +
    "Nuevo: " + evt.newValue + 'r' + "Hora: " + (new Date(evt.timeStamp)).toLocaleString('es-ES'));
}
window.onload = function () {
    window.addEventListener('storage', reportStorage, false);
    paste();
}
function alias() {
    var valor = document.getElementById("C").value;
    localStorage['aValue'] = valor;
    alert('Alias: ' + valor);
}
function nombre() {
    // Tomamos la referencia del nombre a copiar
    var valor = document.getElementById("B").value;
    localStorage['sValue'] = valor;
    alert('Nombre: ' + valor);
}
function paste() {
    // Recuperamos el valor del nombre y el alias
    var localValue = localStorage['sValue'];
    var localValueAlias = localStorage['aValue'];
    // Pegamos el valor del nombre y el alias
    document.getElementById("B").value = localValue;
    document.getElementById("C").value = localValueAlias;
    alert('RECUPERADOS');
}
</script>
```

La función que asociemos al evento recibirá un parámetro de tipo `StorageEvent` donde se nos indicará la key modificada, el valor antiguo, el nuevo y la url que propició el cambio.

Depuración.

Según el navegador que utilicemos podremos consultar visualmente el contenido almacenado en `localStorage`. Algunos navegadores incorporan estos visualizadores en sus herramientas para desarrolladores, como es el caso de Chrome y Safari (Recursos -> Almacenamiento local). Otros como Firefox de momento necesitan hacerlo a través de una extensión.

III. Material y Equipo.

Para la realización de la guía de práctica se requerirá lo siguiente:

No.	Requerimiento	Cantidad
1	Guía de práctica #8: JSON, <code>localStorage</code> y <code>sessionStorage</code>	1
2	Computadora con editor HTML instalado y navegadores	1

IV. Procedimiento.

Ejercicio #1: El siguiente ejemplo muestra cómo se puede utilizar un arreglo de objetos JSON para almacenar datos personales de una lista de personas.

Guión 1: `listanombres.html`

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Datos personales con JSON</title>
  <meta charset="utf-8" />
  <link rel="stylesheet" href="css/grid.css" />
</head>
<body>
<div id="wrapper">
<header>
  <h1>Datos personales con JSON</h1>
</header>
<section>
<article id="test">
<div id="info"></div>
</article>
</section>
<script src="js/names.js"></script>
</div>
</body>
</html>
```

Guión 2: `names.js`

```
var nombres = {
  "Personas": [
    { "imagen" : "img/juan-robles.jpg",
      "nombre" : "Juan",
      "apellido" : "Robles",
      "edad" : 29,
      "profesion" : "Licenciado"},
    { "imagen" : "img/lilian-rosales.jpg",
      "nombre" : "Lilian",
      "apellido" : "Rosales",
      "edad" : 27,
      "profesion" : "Bachiller"},
    { "imagen" : "img/gustavo-gonzalez.jpg",
      "nombre" : "Gustavo",
      "apellido" : "Gonzalez",
      "edad" : 28,
      "profesion" : "Arquitecto"},
    { "imagen" : "img/genesis-deras.jpg",
      "nombre" : "Génesis",
      "apellido" : "Deras",
      "edad" : 31,
      "profesion" : "Mercadóloga"},
  ]
}
```

```

        {"imagen" : "img/reina-benitez.jpg",
         "nombre" : "Reina",
         "apellido" : "Benitez",
         "edad" : 25,
         "profesion" : "Secretaria"},
        {"imagen" : "img/hugo-perez.jpg",
         "nombre" : "Hugo",
         "apellido" : "Pérez",
         "edad" : 34,
         "profesion" : "Diseñador"}
    ]
};

//Obteniendo el elemento contenedor donde se cargarán
//todos los datos del objeto JSON
var div = document.getElementById("info");
div.innerHTML = volcarDatos(nombres.Personas);

function volcarDatos(datos){
    var total = datos.length;
    data = "<ul class='grid'>\n";
    for(var i=0; i<total; i++){
        data += "<li class='box'>\n";
        data += "<div class='box-shadow'></div>\n";
        data += "<div class='box-gradient'>\n";
        data += "<img src='\" + datos[i].imagen + \"' alt='Avatar de \" + datos[i].nombre
+ \" \" + datos[i].apellido + \"' class='avatar' />\n";
        data += "<h4>\nNombre: \" + datos[i].nombre + \" \" + datos[i].apellido +
\"</h4>\n";
        data += "<p>\nEdad: \" + datos[i].edad + \"<br />\n";
        data += "Profesión: \" + datos[i].profesion + \"</p>\n";
        data += "</div>\n";
        data += "</li>\n";
    }
    data += "</ul>\n";
    return data;
}

```

Resultado al visualizarlo en el navegador de su preferencia:



Ejercicio #2: En el siguiente ejemplo se muestra cómo utilizar el almacenamiento local para capturar datos ingresados desde un formulario, de modo que una vez presionado el botón guardar, aunque se actualice la página desde el navegador, e incluso, cuando se cierre la ventana o pestaña, el dato guardado se conservará, obteniéndolo desde la memoria local.

Guión 1: almacenamientolocal.html

```

<!DOCTYPE html>
<html lang="es">
<head>
    <title>Disponibilidad de localStorage</title>
    <meta charset="utf-8" />
</head>
<body>
<section>
<article>
    <span id="webStorageStoredText"></span>
    <br />
    <label for="webStorageInput">Ingrese el dato:</label>
    <input type="text" id="webStorageInput" name="webStorageInput" /><br />
    <input type="button" id="webStorageSaveBtn" value="Guardar" />

```



```

        <input type="button" id="clearWebStorage" value="Borrar" />
        <span id="notCompatibleMsg" style="color:#CC0000; display:none;">
            Desafortunadamente, tu navegador no tiene soporte para almacenamiento
        local.</span>
        <span id="isCompatibleMsg" style="color:#009900; display:none;">
            ¡Felicidades, tu navegador soporta almacenamiento local!</span>
    </article>
</section>
<script src="js/savelocalstorage.js"></script>
</body>
</html>

```

Guión 2: savelocalstorage.js

```

//Verificar si el navegador utilizado posee soporte para usar localStorage.
//Si no lo es advertir al usuario enviando una advertencia de no compatibilidad.
if(typeof(localStorage) == "undefined") {
    document.getElementById("notCompatibleMsg").style.display = 'block';
}
else {
    document.getElementById("isCompatibleMsg").style.display = 'block';
}

//Obtener todos los elementos del documento que se requieren
//utilizando el modelo del DOM Nivel 2.
var storedTextContainer = document.getElementById("webStorageStoredText");
//Acceder al dato almacenado previamente con localStorage.
var storedText = localStorage.getItem('webStorageTestInput');
var inputBox = document.getElementById("webStorageInput");
var saveBtn = document.getElementById("webStorageSaveBtn");
var clearBtn = document.getElementById("clearWebStorage");

//Verificar si el dato almacenado en el objeto localStorage es nulo.
if(storedText != null){
    //Si estamos aquí significa que existe un dato almacenado en localStorage.
    //Colocamos el datos almacenado en el elemento de la página con id
    storedTextContainer.
    storedTextContainer.innerHTML = "<strong>Dato cargado desde almacenamiento
    local:</strong> " + storedText;
    //Volver a cargar el dato almacenado en el campo de texto
    inputBox.value = storedText;
}

//Usamos el manejador de evento onclick sobre el botón Guardar para almacenar
//el dato ingresado hasta el momento en el objeto localStorage.
saveBtn.onclick = function(){
    //Before we save this data, let's escape it so you "hackers" out there don't try
    anything funny!
    var valueToSave =inputBox.value.replace(/<\|/?[^\>]+>/gi, '');

    //This is the way we store things in localStorage
    localStorage.setItem('webStorageTestInput',valueToSave);

    //Let the user know the text was saved.
    storedTextContainer.innerHTML = "Se ha almacenado correctamente '" + valueToSave +
    "' Actualice la página para que observe que el dato almacenado es mostrado en el campo
    de formulario.";
}

//Borramos todo el contenido (clave y valor) almacenado en el objeto localStorage
clearBtn.onclick = function(){
    //Verificamos que exista dato almacenado en localStorage
    if(storedText != null){
        localStorage.clear();
        //Limpiar la caja de texto
        inputBox.value = "";
        storedTextContainer.innerHTML = "Almacenamiento local liberado.";
    }
}

```


Resultado en el navegador:

Se ha almacenado correctamente 'Estamos probando e.' Actualice la página para que observe que el dato almacenado es mostrado en el campo de formulario.
Ingrese el dato: Estamos probando e

¡Felicidades, tu navegador soporta almacenamiento local!

Dato cargado desde almacenamiento local: Umbr
Ingrese el dato: Umbr

¡Felicidades, tu navegador soporta almacenamiento local!

Almacenamiento local liberado.
Ingrese el dato:

¡Felicidades, tu navegador soporta almacenamiento local!

Ejercicio #3: El siguiente ejemplo explora la disponibilidad de las tres formas de usar localStorage, usando la API de JavaScript, notación de matrices y como propiedades de objeto.

Guión 1: formasdistintas.html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Formas de utilizar localStorage</title>
  <meta charset="utf-8" />
</head>
<body>
<section>
<article>
<form name="form">
  <select name="methods">
    <option value="api" selected="selected">API localStorage</option>
    <option value="array">Notación de matrices</option>
    <option value="object">Propiedades de objeto</option>
  </select><br />
  <input type="button" name="seleccionar" id="seleccionar"
value="Seleccionar" />
</form>
</article>
</section>
<div id="content"></div>
<script src="js/formstorage.js"></script>
</body>
</html>
```

Guión 2: formstorage.js

```
var valor;

function init(){
  var seleccion = document.getElementById("seleccionar");
  if(seleccion.addEventListener){
    seleccion.addEventListener("click", mostrar, false);
  }
  else if(seleccion.attachEvent){
    seleccion.attachEvent("onclick", mostrar);
  }
}

function mostrar(){
  //Determinando la opción seleccionada
  var opcion = document.form.methods.options[document.form.methods.selectedIndex].value;
  switch(opcion){
    case "api":
      // Utilizando la Api de Local Storage, podemos guardar y luego recuperar un
      valor = localStorage.getItem('bienvenida');
      localStorage.setItem('bienvenida', 'Bienvenidos a localStorage con API de
      JavaScript');
      break;
    case "array":
      // Utilizar notación de matrices o arreglos
      localStorage['bienvenida'] = 'Bienvenidos, también podemos usar notación de
      matrices';
```

```

        valor = localStorage['bienvenida'];
        break;
    case "object":
        // Y claro, también es válido utilizar notación de propiedades de objetos */
        localStorage.bienvenida = 'Bienvenidos, igual podemos usarlo como propiedades
de objeto';
        valor = localStorage.bienvenida;
        break;
    default:
        alert("Esta opción no existe");
        break;
    }
    var contenido = document.getElementById("content");
    contenido.innerHTML = "<p>\nUsando : <strong>" + opcion + "</strong> - " + valor +
"\n\t</p>\n";
}

if(window.addEventListener){
    window.addEventListener("load", init, false);
}
else if(window.attachEvent){
    window.attachEvent("onload", init);
}

```

Ejemplo #4: El siguiente ejemplo ilustra las distintas formas de utilización del objeto localStorage para almacenar un dato simple, un objeto o un arreglo de datos. Se utiliza un formulario para ingresar dos datos como nombre y edad y luego se guardan con los botones Añadir, un dato como cadena, con el botón Guardar dato complejo se almacena como un objeto y con el botón Guardar arreglo se almacenan ambos datos como arreglo. En este último caso, puede almacenar muchos pares de datos nombre y edad. Adicionalmente, puede remover los datos almacenados con Añadir con el botón Remover o Limpiar todo el contenido del localStorage con el botón Limpiar. También se pueden restaurar el objeto o el arreglo guardado con los botones Restaurar dato complejo y Restaurar arreglo, respectivamente.

Guión 1: teststorage.html

```

<!DOCTYPE html>
<html>
<head>
    <title>Prueba de almacenamiento local</title>
    <meta charset="utf-8" />
    <link rel="stylesheet" href="css/base.css" />
    <script src="js/appform.js"></script>
</head>
<body>
<section>
<article>
<form name="registryform">
<fieldset id="myform">
<legend>Datos personales</legend>
<ul class="formset">
    <li>
        <label for="name">Nombre:</label>
        <input type="text" name="name" id="name" placeholder="(Su nombre)" />
    </li>
    <li>
        <label for="age">Edad:</label>
        <input type="number" name="age" id="age" placeholder="(Su edad)" />    </li>
    </li>
    <li>
        <input type="button" id="add" value="Añadir"
onclick="Javascript:save()">
        <input type="button" id="get" value="Obtener"
onclick="Javascript:obtain()">
        <input type="button" id="del" value="Remover"
onclick="Javascript:remove()">
        <input type="button" id="clear" value="Limpiar"
onclick="Javascript:clearStorage()">
    </li>
    <li>
        <input type="button" id="saveC" value="Guardar dato complejo"
onclick="Javascript:saveComplexData()">

```

```

        <input type="button" id="restoreC" value="Restaurar dato complejo"
onclick="Javascript:restoreComplexData()">
        <input type="button" id="sArray" value="Guardar arreglo"
onclick="Javascript:saveArrayData()">
        <input type="button" id="rArray" value="Restaurar arreglo"
onclick="Javascript:restoreArrayData()">
    </li>
</ul>
</fieldset>
</form>
<div id="state"></div>
</article>
</section>
</body>
</html>

```

Guion 2: appform.js

```

var myArrayObject = [];
var divState;

function init() {
    console.log("Carga de la página finalizada.");
    if(typeof(Storage) == "undefined") {
        alert("El navegador no tiene soporte para HTML5 y almacenamiento local. Se
recomienda actualizarlo.");
    }
    else {
        console.log("El navegador soporta tanto localStorage como sessionStorage.");
        divState = document.getElementById("state");
    }
    divState = document.getElementById("state");
    if(typeof(localStorage) == "undefined"){
        divState.style.display = 'none';
    }
    else{
        divState.style.display = 'block';
    }
}

function save() {
    var myName = document.getElementById("name");
    var age = document.getElementById("age");
    var msg;
    //Verificar si se puede utilizar localStorage en el navegador
    try {
        localStorage.setItem("name", myName.value);
        localStorage.setItem("age", age.value);
        myName.value = "";
        age.value = "";
        msg = "Datos guardados en el localStorage.";
        console.log(msg);
        //Mostrar al usuario mensaje de estado
        divState.innerHTML = "<p>" + msg + "</p>";
    }
    catch (e) {
        //Verificar si el límite de almacenamiento no se ha sobrepasado
        if (e >= QUOTA_EXCEEDED_ERR) {
            console.log("Error: Límite para almacenamiento local se ha alcanzado.");
        }
        else {
            console.log("Error: Guardando en el almacenamiento local.");
        }
    }
}

function obtain() {
    var msg = "Obteniendo el dato " + localStorage.key(0) + " y " + localStorage.key(1)
+ " desde el localStorage.";
    var myName = document.getElementById("name");
    var age = document.getElementById("age");
    console.log(msg);
    divState.innerHTML = "<p>" + msg + "</p>";
    myName.value = localStorage.getItem("name");

```

```

    age.value = localStorage.getItem("age");
}

function remove() {
    console.log("Removiendo dato del localStorage.");
    localStorage.removeItem("name");
    localStorage.removeItem("age");
}

function clearStorage() {
    divState.innerHTML = "";
    console.log("Borrando todo el contenido de localStorage.");
    localStorage.clear();
}

function saveComplexData() {
    console.log("Guardando datos complejos a localStorage.");
    var myName = document.getElementById("name");
    var age = document.getElementById("age");
    var personObject = new Object();
    personObject.name = myName.value;
    personObject.age = age.value;
    localStorage.setItem("person", JSON.stringify(personObject));
}

function restoreComplexData() {
    console.log("Restaurando datos complejos desde localStorage.");
    var myName = document.getElementById("name");
    var age = document.getElementById("age");
    var personObject = JSON.parse(localStorage.getItem("person"));
    myName.value = personObject.name;
    age.value = personObject.age;
    //Mostrar al usuario en el elemento DIV el contenido del objeto
    divState.innerHTML = "<p>Nombre: " + personObject.name + ", Edad: " + personObject.age
+ "</p>";
}

function saveArrayData() {
    var myName = document.getElementById("name");
    var myAge = document.getElementById("age");

    //Creando el arreglo con los datos
    var personObject1 = new Object();
    personObject1.name = myName.value;
    personObject1.age = parseInt(myAge.value);
    myArrayObject.push(personObject1);

    console.log("Guardando arreglo de datos en localStorage.");
    localStorage.setItem("persons", JSON.stringify(myArrayObject));
}

function restoreArrayData() {
    divState.innerHTML = "";
    console.log("Restaurando arreglo de datos desde localStorage.");
    var myArrayObject = JSON.parse(localStorage.getItem("persons"));
    for(var i=0; i<myArrayObject.length; i++){
        var personObject = myArrayObject[i];
        console.log("Nombre: " + personObject.name, "Edad: " + personObject.age);
        divState.innerHTML += "<p>Nombre: " + personObject.name + ", Edad: " +
personObject.age + "</p>";
    }
}

if(window.addEventListener){
    window.addEventListener("load", init, false);
}
else if(window.attachEvent){
    window.attachEvent("onload", init);
}

```

Resultado:

Datos personales

Nombre:
Pedro Cruz

Edad:
42

Añadir

Obtener

Remover

Limpiar

Guardar dato complejo

Restaurar dato complejo

Guardar arreglo

Restaurar arreglo

Obteniendo el dato age y bienvenida desde el localStorage.

Datos personales

Nombre:
Lidia Sambrano

Edad:
27

Añadir

Obtener

Remover

Limpiar

Guardar dato complejo

Restaurar dato complejo

Guardar arreglo

Restaurar arreglo

Nombre: Pedro Cruz, Edad: 42
Nombre: René Orantes, Edad: 34
Nombre: Lidia Sambrano, Edad: 27