# Práctica de JavaScript

## Operaciones con arreglos de usuarios

Objetivo: Emplear el conjunto de instrucciones de JavaScript para simular operaciones de altas, bajas, cambios y consultas en un archivo JSON.

## Descripción de la aplicación

Se desea tener una interfaz web que permita realizar diferentes operaciones sobre un arreglo de objetos de usuarios. Estas funciones se mandarán a llamar desde la consola del navegador. Las operaciones a realizar son:

1. Lectura de un JSON, paseo de datos
2. Mostrar arreglo de objetos en HTML
3. Insertar datos de un nuevo usuario
4. Editar datos de un usuario
5. Borrar un usuario
6. Ordenamiento de usuarios
7. Búsqueda de usuarios por diferentes criterios
8. Guardar en el JSON

## Creación estructura principal

En tu editor crea un archivo index.html que tenga la estructura básica y Bootstrap. (recuerda el snippet **b4-$**). A este archivo añádele un Título y un div vacío con el **id="info"**
Crea 2 archivos de JavaScript users.js y almacenamiento.js (añádeles al inicio “use strict”)
En tu html importa los scripts de javascript en cualquier lugar (en head o en body)

## Creación y lectura de un JSON

Copia y pega este arreglo de objetos de tipo usuarios, puedes agregar más si lo deseas.

```
[{
    "nombre":"Juan",
    "apellidos": "Perez",
    "email":"perez@gmail.com",
    "password":"hoola",
    "fecha":"2000-10-28",
    "sexo":"H",
    "uid":10001,
    "image":"https://randomuser.me/api/portraits/men/0.jpg",
}]
```

En tu archivo de almacenamiento.js guárdalo en una variable como string. let url = "http://practica2-iteso.mybluemix.net/users"
Copia el código del siguiente enlace: https://pastebin.com/1nHVcjMR
En tu consola verifica que funciona (**Intentalo!**), solo debes poner el nombre de la función y la url, en la consola se debe imprimir el arreglo con un elemento (un usuario).

# Como agregar contenido al HTML

Recuerda que puedes agregar contenido al HTML asignando valores a la propiedad de *document.getElementById('info').innerHTML = código_html*

# La carga de los datos es asíncrona

Esta tal vez sea **la parte más difícil de la práctica**, la dejamos al inicio para que consultes al profesor en caso de que lo requieras.
En users.js crea un arreglo vacío llamado usuarios.
Crea una clase que se llame User con las propiedades del objecto que se muestra arriba de tipo Usuario.
Queremos en una función llamada **initData()** se mande llamar la función de **loadJSON** y que una vez que se reciben los datos se asignen directamente a nuestro arreglo usuarios y mostrar un mensaje de éxito o error en el html.
Como lo puedes hacer??? (**piensa un instante antes de continuar**), debes considerar que si pones código debajo de loadJSON() el código debajo se va a ejecutar antes porque loadJSON está realizando una petición a un servidor (**es asíncrono**!!). Además, no tienes la respuesta de los usuarios.
Una estrategia es utilizar callbacks que se ejecutarán una vez que se reciba la respuesta ya sea exitosa o de error. Para ello añade 2 argumentos en la función de loadJSON: cbOk, cbErr, el primero ejecútalo donde todo resultó correcto como cbOk(datos), y el otro donde ocurrió un error como cbErr().
¡**Ahora sí**!!!, desde users.js puedes mandar llamar loadJSON y le mandas los callbacks que deseas ejecutar. Si todo es correcto muestra en el div (**info.innerHTML**) asigna al arreglo usuarios los datos y muestra algún mensaje o incluso imprime el arreglo de usuarios). En caso de error solo muestra un mensaje de que ocurrió un error. **Pruébalo** llamando initData()

# Mostrar arreglo de usuarios en HTML

Para esta parte usaremos el método **map** y el método **join** que nos permitirán convertir el arreglo a html y concatenarlo.
Crea primero en index.html una estructura para visualizar al usuario, ya que te guste copia este código y pégalo en una función en users.js dentro de la clase User. Éste método puede llamarse **userToHTML**(user), esta función recibe un usuario a convertir y regresará un string de HTML con los datos del usuario. Puedes probar que funciona desde la consola. Debes mostrar la imagen.
Crea otra función que se llama **userListToHTML**(**listaUsuarios**) que convierte todo el arreglo en HTML y lo muestra en el div (aquí usa el método map). Observa que recibe un

argumento porque esta función te permitirá mostrar no solo el arreglo de usuarios sino cualquier arreglo que guarde usuarios.

# Añadir usuarios

Crea una función "**addUser**" que recibe los atributos de un nuevo usuario y le asigna su "uid" (user id) y lo añade al arreglo. Asegúrate de que no exista otro usuario con el mismo email ni con el mismo uid. Puedes asignarle una imagen de forma automática con la información del sexo y el id para formar la url: `https://randomuser.me/api/portraits/men/0.jpg` `cambiar por men o women/uid.jpg`
Para esta función podría ayudarte los métodos **find**, string.**toUpperCase()**, **push**. Además, te podría ayudar tener una función generadora o una clase **User** que incremente su uid con forme se van creando nuevos usuarios. Al final manda llamar a **userListToHTML** para que se actualicen los datos.

Intenta agregar al menos 5 usuarios diferentes.

```
function addUser(user){}
```

# Editar usuarios

Crea una función "**updateUser**" que reciba el id y un objeto con los valores a actualizar, verifica que exista el usuario y copia los valores excepto el uid (no queremos que se modifique).
Para este método te podría servir **findIndex**, **Object.assign**.
Al final manda llamar a **userListToHTML** para que se actualicen los datos.

```
function updateUser(id,obj){}
```
donde **obj** es un objeto con las propiedades a actualizar

# Borrar

Crea una función "**deleteUser**" que reciba el id verifica que exista y se borra, puedes usar **findIndex** y **splice**
Al final manda llamar a **userListToHTML** para que se actualicen los datos.

```
function deleteUser(id){}
```

# Ordenamiento

Crea una función de "**sortUsers(cb)**", que recibe una función que indica cómo hacer el ordenamiento. Internamente usa el método sort para el arreglo usuarios que ejecuta ese callback.

Al final manda llamar a **userListToHTML con el nuevo arreglo** para que se actualicen los datos.

```
function sortUsers(cb){}
```

# Búsqueda

Esta parte es para realizar filtros, crea una función que se llame "**findUsers(nombre, email, sexo, fechaIni, fechaFin )**" solo los argumentos no **undefined** son considerados para hacer el filtro.
Ejemplos:
findUsers(undefined, undefined, "M") crea un nuevo arreglo con solo mujeres
findUsers("ar", undefined, "M") crea un nuevo arreglo con solo mujeres que **su nombre o apellido** contiene las letras "ar", para ello el nombre y el apellido comparan con toUpperCase() y también el argumento. Podría ser: Marta, Mar, María, Martínez. Puedes usar el método **.includes()** de las cadenas
En el caso de fechaIni si fechaFin es undefined buscará todos los que nacieron después de esa fecha. Si fechaIni es undefined y fechaFin no, entonces todos los menores o iguales a esa fecha, si ambos existen entonces buscará en ese rango de fechas. Podrías usar let mydate = new **Date**('2019-10-03') entre tipos fecha (Date) puedes usar el operador >= o <=
Al final manda llamar a **userListToHTML con el nuevo arreglo** para que se actualicen los datos.

# Guardar en el JSON

Similar a como se cargaron los datos ahora los vamos a guardar, para ello se te facilita el código que puedes encontrar en este enlace: https://pastebin.com/zDMAFF2K que puedes pegar en el archivo de almacenamiento.js . A este código debes añadirle la posibilidad de recibir un callback de éxito y uno de error.
Puedes añadir en el archivo index.html otro div para mensajes. En *users.js* crea una función que se llame **saveUsers()** que ejecute la función **guardarEnJSON** para guardar el arreglo de usuarios y debe mostrar un mensaje en el nuevo div de éxito o error.

# Entregables

Subir index.html, almacenamiento.js, users.js junto con un archivo .pdf conteniendo una reflexión de la práctica. Comprime todo y subelo en un archivo .zip.

El nombre del archivo comprimido deberá seguir el formato
appelidos_nombre_expediente.zip

Asegurate de utilizar los nombres de propiedades y funciones como vienen en el documento. Ya que para evaluar se utilizará un script para determinar si los casos de prueban pasan o no.

# Evaluación

| Actividad | Ponderación | Realizado SI/NO |
|---|---|---|
| Importar datos, asignar los datos cargados al arreglo de usuarios. Se obtiene el id máximo y se guarda en alguna variable. | 10 | |
| userToHTML,userListToHTML: Mostrar el arreglo de usuarios en el html con formato Bootstrap. También muestra el url | 15 | |
| addUser: añadir un usuario al arreglo de usuarios. Con las validaciones solicitadas. Que automáticamente se incremente el uid y se asigne una imagen. | 15 | |
| updateUser: actualizar usuario con las validaciones solicitadas | 10 | |
| deleteUser: eliminar usuario | 10 | |
| sortUsers: ordenar por uno o varios criterios | 15 | |
| findUsers: filtros por diferentes criterios: nombre y apellidos, email, sexo, por fechas. | 20 | |
| saveUsers: guardar el arreglo en myjson.com y mostrar un mensaje cuando sea exitoso o cuando sea error usando callbacks | 10 | |