



Nuestro compromiso es con el *futuro*.

# Introducción a la programación

# Repaso

# forEach

Método que recibirá un **callback** que se ejecutará por cada uno de los elementos del array. Este **callback**, a su vez, recibirá al menos un **parámetro**, que representará literalmente el elemento que es contenido por ese **Array** que está siendo iterado. También disponemos de un segundo **parámetro** opcional que funciona como **índice**, es decir, la **posición** del **Array** en la que nos encontramos.

```
let nombres = ['Virginia', 'Josefina', 'Juan', 'Francisco']

let funcionCallback = (elemento, indice) => {
  console.log(`El elemento en la posición ${indice} es ${elemento}`)
}

nombres.forEach(funcionCallback)
```

```
$ node index
El elemento en la posición 0 es Virginia
El elemento en la posición 1 es Josefina
El elemento en la posición 2 es Juan
El elemento en la posición 3 es Francisco
```

# map

El método **map** es muy parecido al **forEach** en cuanto a sus parámetros (son exactamente los mismos) pero como vimos recién difieren en el retorno. El **map** junto con el **reduce** y el **filter** retornan algo, y es esto que retornan lo que nos resulta de utilidad. El **map**, por ejemplo, retorna un **Array** con las condiciones que le pasemos dentro del **callback**. Por ejemplo:

```
let array1 = ['nombre1', 'nombre2', 'nombre3']
let array2 = array1.map((element, index) =>
  `${element} ${index}`)
```

```
console.log(array1)
console.log(array2)
```

```
$ node index
[ 'nombre1', 'nombre2', 'nombre3' ]
[ 'nombre1 0', 'nombre2 1', 'nombre3 2' ]
```

# filter

El **filter** funciona muy parecido al **map**, en que retorna algo, solo que es mucho más sencillo, ya que nos devolverá un **Array** con los elementos del **Array** original que cumplan con la condición que le pasemos en el **callback**.

Por ejemplo:

```
let letras = ['asd', 'qwe', 'zxc', 'zxe', 'ase']
let filteredLetras = letras.filter(x => x.includes('e'))
console.log(filteredLetras)
```

```
$ node index
[ 'qwe', 'zxe', 'ase' ]
```

# Clase 9

# Aplicación de tareas v3

En la clase de hoy terminaremos de darle algunos toques a nuestra aplicación de tareas.

Limpiaremos un poco el código y actualizaremos algunos métodos y partes de la misma.



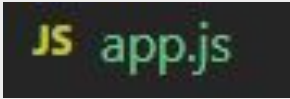
# Aplicación de tareas v2

Nuestra app ahora contará varias funcionalidades, pero actualizadas para ser más modulares y modernas (utilizaremos funciones de orden superior en vez de simples bucles).

- **create** (crear)
- **read** (leer)
- **update** (actualizar)
- **delete** (borrar)
- **return** (devolver)
- **filter** (filtrar)
- **find** (encontrar)

# Pasos a seguir: 1

1 - Continuaremos utilizando el archivo `app.js` para esta **versión 3.0** de nuestra aplicación. Seguirá siendo nuestro entry-point, pero le añadiremos las funcionalidades adicionales que veremos a continuación.



```
JS app.js
```

## Pasos a seguir: 2

2 - Utilizaremos tanto la carpeta **CRUD**, como la carpeta **funcionalidades** o **comandos** que utilizamos en la primera versión.  
El **código** que irá en estas carpetas será basado en las versiones anteriores, pero modificándolo.

## Pasos a seguir: 3

3 - Archivo `read.js`: esta funcionalidad deberá seguir haciendo uso del módulo `fs (file system)` de `nodejs` para poder leer el `JSON de datos`, y retornar su contenido como un `Array de Objetos Literales` de `JavaScript` para que éste pueda trabajarse con normalidad.

## Pasos a seguir: 4

4 - Archivo `create.js`: esta funcionalidad seguirá recibiendo como parámetros un título y una descripción (ambos en formato string), y deberá agregar una nueva tarea (`Objeto Literal` con un *title* y *desc* que serán las propiedades que contendrán a los parámetros que llegan a la función) al **JSON de datos**.

Recordemos **obtener el listado de tareas**, **agregarle esta nueva tarea** y luego escribir el **archivo** con la lista actualizada.

## Pasos a seguir: 5

5 - Archivo **delete.js**: esta funcionalidad seguirá recibiendo como parámetro un título, obtendrá el listado de tareas, y eliminará de ese listado la tarea cuyo *title* coincida con el parámetro que recibe. Finalmente deberá escribir el **JSON de datos** con el listado actualizado.  
Ya no es de carácter opcional.

## Pasos a seguir: 6

6 - Archivo **edit.js**: Seguirá recibiendo como parámetros un título y una descripción. Deberá obtener el listado de tareas y, recorriéndolo, modificar la *desc* del elemento cuyo *title* coincida con el recibido por parámetro. Finalmente deberá escribir el **JSON de datos** con el listado actualizado.

# Pasos a seguir: 7

7 - Archivo `find.js`: esta funcionalidad seguirá recibiendo un parámetro y deberá retornar un **true** o **false** dependiendo de si encuentra o este parámetro dentro del listado de tareas (cómo título).



## Pasos a seguir: 8

8 - Archivo `return.js`: esta funcionalidad continuará recibiendo un *title* por parámetro (*string*) y en caso de que ese título coincida con alguno de los que existen dentro del listado. Deberá retornar la *desc* que corresponde a esa tarea.

## Pasos a seguir: 9

- 9 - Archivo `filter.js`: esta funcionalidad retornará si encontró el valor que se le pasa por parámetro o no, esta función deberá retornar todas las tareas que contengan (en su título) lo que se le pasa por parámetro a la función. Recordemos que ahora disponemos del método `filter`. Que debería facilitarnos muchísimo esta tarea.

# Pasos a seguir: 10

10 - Finalmente debemos actualizar el entry point.

Actualizaremos nuestro código para recibir los siguientes comandos:

- `list` → deberá imprimir por consola todas las tareas
- `add param1 param2` → deberá agregar una nueva tarea que tenga como *title* el contenido del `param1` y como *desc* el `param2`. Recordemos que estos `param1` y `param2` serán *strings* con los *datos* de la tarea a cargar.
- `edit param1 param2` → deberá editar la tarea que tenga como *title* el contenido del `param1` modificando su *desc* con el contenido del `param2`.

# Pasos a seguir: 10

- **delete param1** → deberá eliminar la tarea cuyo *title* coincida con el param1.
- **filter param1** → deberá retornar las tareas cuyos *title* contengan el param1 (sólo que lo contengan, aunque sea una parte del *title*).
- **return param1** → deberá retornar la *desc* de la tarea que tenga como *title* el contenido del param1.
- **find param1** → deberá retornar true o false si encuentra la tarea que tenga como *title* el contenido del param1.



**JS**

**¡Desafíos!**

# Éxito!

Si todo funcionó correctamente, deberíamos tener una aplicación que nos permita realizar leer comandos de terminal y ejecutar código según corresponda, pero con una sintaxis moderna y mejorada!

Felicitaciones!

# Algunas consideraciones

# Muchas gracias!



ICARO Asociación Civil  
CUIT 30716564815  
[info@icaro.org.ar](mailto:info@icaro.org.ar)  
[www.icaro.org.ar](http://www.icaro.org.ar)