



Nuestro compromiso es con el *futuro*.

Introducción a la programación

Repaso

process.argv

Recordemos que el `process.argv` nos permite leer los comandos que ingresamos por terminal. Utilizando este objeto, podremos ir preparando el código para que dispare distintas funcionalidades según lea nuestros comandos. Recordemos que si queremos pasar mucho texto como un solo parámetro debemos utilizar comillas.

```
const saludar = (name) => {  
  console.log(`Hola, ${name}!`)  
}  
  
if (process.argv[2] === 'saludar') {  
  saludar(process.argv[3])  
}
```

```
$ node app saludar Juan  
Hola, Juan!
```

writeFileSync y readFileSync

Recordemos que estos métodos nos servirán tanto para **leer** como para **escribir** un archivo. En nuestro caso los utilizaremos para poder obtener y actualizar nuestros datos del **JSON de datos**, que utilizaremos para persistir nuestras tareas.

```
console.log(fs.readFileSync('./data/data.json', 'utf-8'))
```

```
fs.writeFileSync('./random/random.txt', 'esto es un texto random')
```

Clase 7

Aplicación de tareas v2

En la clase de hoy, continuaremos modificando nuestra app para que ésta pueda comenzar a leer y escribir nuestros **archivos de datos**. En los mismos almacenaremos las distintas tareas que se irán cargando por **consola**. La idea es generar una aplicación que pueda **leer**, **agregar**, **editar** o **borrar** una tarea de un listado. Todo esto de manera permanente, es decir con **datos que persistan**.

Aplicación de tareas v2

Nuestra app ahora contará con nuevas funcionalidades que crearemos en base a la última práctica y lo aprendido en la última clase, un CRUD:

- **create** (crear)
- **read** (leer)
- **update** (actualizar)
- **delete** (borrar)

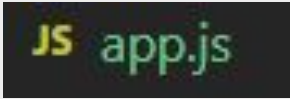
Pasos a seguir: 1

1 - Modificaremos el archivo **tasksData.js** que creamos en la clase anterior adaptándolo a un formato **JSON**:

Debemos modificar el nombre del archivo a **tasksData.json**, quitar la **constante** y la **referencia**, y simplemente dejar el contenido como **datos**. Además, deberemos agregar **comillas dobles** a todas las propiedades y modificar los valores de ser necesario.

Pasos a seguir: 2

2 - Crearemos en la raíz del proyecto un nuevo **archivo principal**, que nos servirá para esta **versión 2.0** de nuestra aplicación. Podemos llamarle a este nuevo archivo **app.js**, por ejemplo.



```
JS app.js
```

Pasos a seguir: 3

3 - Ya no utilizaremos la carpeta comandos, sino que crearemos una nueva que se va a llamar **CRUD**, donde colocaremos nuestras nuevas cuatro funciones básicas.

El **código** que irá en estas carpetas estará inspirado en el que creamos en la última práctica.

A continuación veremos cada archivo.

✓ CRUD

JS create.js

JS delete.js

JS read.js

JS update.js

Pasos a seguir: 4, read

4. a - Archivo **read.js**: esta funcionalidad deberá hacer uso del módulo **fs (file system)** de **node.js** para poder leer el **JSON** de datos, y devolver su contenido como un **Array de Objetos Literales** de **JavaScript** para que éste pueda trabajarse con normalidad.

Este código, a pesar de su simpleza, es uno de los más importantes porque será quien provea a toda nuestra aplicación del contenido del **JSON** de datos.

Pasos a seguir: 4, write

4. b - Archivo **writeJSON.js**: esta funcionalidad deberá hacer uso del módulo **fs (file system)** de **node.js** para poder recibir como parámetro un **arreglo de objetos literales** de JavaScript y (sobre)escribir nuestro archivo **JSON** de datos (**taskData.json**).

Para realizar esta tarea vamos a echar mano de la función ***writeFileSync*** y de ***JSON.parse***.

Pasos a seguir: 5

5 - Archivo **create.js**: esta funcionalidad recibirá 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.

Para ello, primero deberá **obtener el listado de tareas**, **agregar esta nueva tarea** y luego escribir (más bien, sobrescribir) el archivo con la lista actualizada.

Para realizar estas tareas haremos uso del código del archivo **read.js** y también del **FileSystem** con su **WriteFileSync**.

Pasos a seguir: 6

6 - Archivo **delete.js**: esta funcionalidad es bastante sencilla pero requiere de bastante lógica. Recibirá 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.

Pasos a seguir: 7

7 - Archivo **update.js**: esta funcionalidad es un poco más compleja, aunque muy similar al **create.js**. Recibirá 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: 8 (bonus)

8 - Modificar el código creado para modularizar nuestra aplicación agregando una nueva función al **directorio CRUD**: el **writeJson.js**.

Esta función recibirá como parámetro un array de datos (para nuestro caso un **arreglo de objetos literales**) y deberá escribir el **JSON** de datos con este contenido. Luego deberemos utilizar esta nueva funcionalidad en cada parte donde se requiera escribir los **datos**, logrando así que nuestra app sea mucho más **escalable** y **modular**.

Recordemos que tanto si hacemos este bonus como si no, cuando queramos escribir el JSON de datos, debemos volver a parsear estos datos para que vuelvan a ser un string y no más un Objeto Literal o un Array.

Pasos a seguir: 9

9 - Finalmente trabajaremos en nuestro nuevo **entry point**: el archivo que llamamos en esta ocasión **app.js**. Dentro de éste deberemos crear el código necesario para que al correr nuestra aplicación con los siguientes comandos, se produzcan los siguientes resultados:

- **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.

Pasos a seguir: 9

- `edit param1 param2` → deberá editar la tarea que tenga como *title* el contenido del `param1` modificando su *desc* con el contenido del `param2`.
- `delete param1` → deberá eliminar la tarea cuyo *title* coincida con el `param1`.

Ejemplos:

```
$ node app.js list
```

```
$ node app.js add 'nueva tarea 1' 'esta es una nueva tarea'
```

```
$ node app.js edit 'nueva tarea 1' 'nueva descripcion'
```

```
$ node app.js delete correr
```



JS

Ahora les toca a ustedes!

Éxito!

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

Felicitaciones!

Muchas gracias!



ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar