
ICARO
Nuestro compromiso es con el futuro.

Introducción a la programación

Inicializar el proyecto

Siempre para inicializar, correremos el comando:

```
❯ npm init -y
```

Para requerir módulos de otros archivos:

```
const operaciones = require('./operaciones/operaciones.js')
```

Clase 5

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!
```

Aplicación de tareas v1

En la clase de hoy vamos a codear una app de JavaScript con Nodejs que nos permitirá interactuar con la terminal para lograr distintas funcionalidades al comunicarnos con la aplicación a través de la consola.

Utilizaremos estas funcionalidades para poder leer algunas estructuras y filtrar ciertos comportamientos.

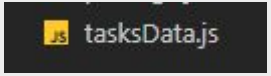
Aplicación de tareas v1

Nuestra app contará con algunas funciones básicas:

- find (encontrar)
- list (listar)
- filter (filtrar)
- edit (editar)
- return (devolver)

Pasos a seguir: 1

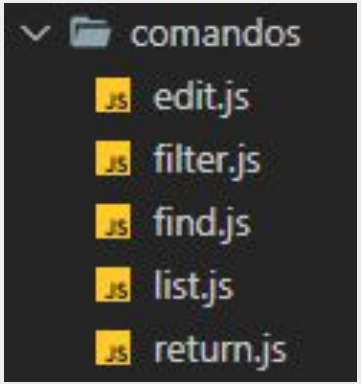
1 - Crear un archivo tasksData.js que contendrá nuestras tareas.
La idea es que contenga en una variable/constante tasks (que exportaremos para consumir en otro archivo) todas las tareas en formato de un *array* de *objetos literales*, donde cada tarea es un *objeto literal* y en su conjunto conforman el *array* de tareas.
Cada tarea deberá tener dos propiedades: title (título) y desc (descripción), ambas de tipo string.




```
const tasks = [
  {
    title: 'task1',
    desc: 'this is a test task'
  },
  {
    title: 'task2',
    desc: 'this is also a test task'
  }
]
```

Pasos a seguir: 2

2 - Crear una carpeta llamada **funcionalidades** o **comandos** donde ubicaremos las distintas funciones que dispararán las acciones de nuestra aplicación. Estas funcionalidades deberán ser un archivo con un nombre descriptivo para cada una, por ejemplo: filter.js, entre otros.



Pasos a seguir: 3

3 - Dentro de la carpeta **funcionalidades/comandos** (le llamaremos comandos de aquí en adelante para simplificar) debemos incluir el código para que cada una de estas funcionalidades cumpla con su propósito. Estas funciones serán disparadas desde nuestro entry point, archivo llamado tasks.js, que deberemos crear en la raíz de nuestro proyecto.

Pasos a seguir: 4

4 - Archivo list.js: esta funcionalidad deberá retornar simplemente el listado completo de las tareas. Para ello, deberá leer por supuesto el listado completo, recorrerlo y retornar solamente los títulos de cada tarea, no su descripción, como un array de strings.

Vale aclarar, que para que podamos leer el listado de tareas deberemos *importarlo* desde cada **archivo** donde lo necesitemos, como por ejemplo en este archivo. Esto es algo que deberemos repetir en cada archivo dentro de **comandos**.

Pasos a seguir: 5

5 - Archivo find.js: esta funcionalidad recibirá 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: 6

6 - Archivo edit.js: esta funcionalidad aún no estará performando al 100% en esta etapa. Pero ya la dejaremos creada para futuras iteraciones de nuestra app. Debería recibir dos parámetros: el primero es **el título** de la tarea a editar, y el segundo es **la nueva descripción** de la tarea a editar. Como esta funcionalidad no hará nada realmente aún, simplemente deberá devolver un objeto literal que contenga la información que a futuro se guardaría en la tarea.

Pasos a seguir: 7 (bonus)

7 - Archivo filter.js: esta funcionalidad hace algo similar al find.js, pero en vez de retornar 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.
Es decir, si tenemos tres tareas: **correr**, **comer** y **saltar**, y llamamos la función pasándole por parámetro el string **“co”**(filter(‘co’)), esta funcionalidad debería retornar un array con las primeras dos tareas(**correr** y **comer**), ya que sólo estas dos contienen el string **“co”** dentro del título.

Pasos a seguir: 8 (bonus)

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

Pasos a seguir: 9

9 - Finalmente trabajaremos en nuestro entry point: el archivo tasks.js. Dentro de este deberemos crear el código necesario para que al correr nuestra aplicación utilizando comandos adicionales puedan lanzarse las funcionalidades que fuimos creando anteriormente.
La idea es que si pasamos, por ejemplo, como **primer parámetro (adicional)** la palabra find, deberá dispararse la función del archivo find.js utilizando el **segundo parámetro (adicional)** como parámetro de búsqueda.

```
❯
❯ node tasks.js find correr
❯
```

Consejo: la estructura switch puede venirnos muy útil.

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!

Algunas consideraciones

Muchas gracias!

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