
ICARO
Nuestro compromiso es con el futuro.

Introducción a la programación

Repaso

Veracidad

Conocemos los valores booleanos (**true** or **false**). Pero **JavaScript** también ejecuta una evaluación sobre los otros tipos o estructuras de dato, a fin de determinar si son verdaderos o falsos.

```
// Datos forzados a true
true
' '          // cualquier string que no esté vacío
1            // cualquier número distinto de cero
['']         // un arreglo, vacío o no
{}           // un objeto
function name() {}  // una función

// Datos forzados a false
false
0
undefined
null
''                  // un string vacío

// verificamos el valor de veracidad
// de un dato con la doble negación: !!
```

Operadores lógicos

- **AND** &&
  Devuelve true si y sólo si **ambas** expresiones son verdaderas.

- **OR** ||
  Le basta con que **apenas una** de las expresiones sea true para devolver true.

- **NOT** !
  Invierte el valor booleano de la expresión.

```
1   if (100 > 10 && 10 > 1) {
2       // ambas expresiones son
3       // correctas (true)
4       // se ejecutará el código
5   }
6
7   if (100 > 10 || 10 === 1) {
8       // la primera expresión es true
9       // la segunda, no
10      // se ejecutará el código
11  }
12
13  if (!(100 < 10)) {
14      // la expresión es false
15      // pero el operador de
16      // negación la transforma a true
17      // se ejecutará el código
18  }
```

Incremental y decremental

- El operador **incremental** ++ incrementa en uno el valor de una variable.
- El operador **decremental** -- decrementa en uno el valor de una variable

```
4   var x = 1;
5   x = x + 1;   // a x se le suma 1, por ende x = 2
6   x += 1;      // ¡así también le sumamos 1! x = 3
7   x++;         // la forma más común de incrementar
8                // por uno: ahora x = 4
9
10  x = x - 1;  // x = 3
11  x -= 1;     // x = 2
12  x--;        // x = 1
```

Clase 4

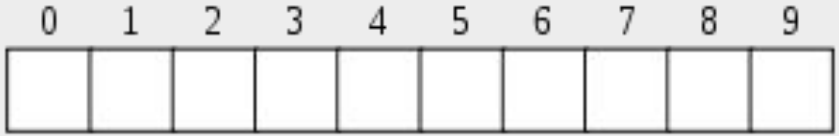
Arrays

Dentro de la programación le llamamos **array**, **vector** o **matriz** a una zona de almacenamiento contiguo en memoria que contiene una serie de elementos del mismo tipo… o no.
A fines prácticos, en JavaScript, estos **arrays** son un tipo de dato que nos permiten guardar más de un valor dentro de sí mismos. Podríamos verlos como una variable con múltiples posibles contenidos.

```
products = ['product 1', 'product 2', 'product 3']
```

Arrays

Es importante destacar, que en **JavaScript**, comenzamos siempre a contar los **arrays** desde la posición 0 en adelante, esto quiere decir que la primer posición es la 0, la segunda la 1, la tercera la 2, etc.
En definitiva, el primer elemento tiene lo que se llama **índice** 0, que no es más que su posición dentro del array; el segundo elemento tiene un **índice** de 1, etc.



Arrays

Para poder acceder a un elemento particular podemos utilizar los **corchetes**. Lo que haríamos es llamar al **array** e indicar entre **corchetes** la **posición** a la cual queremos acceder. Esto retornará el valor almacenado en tal **posición**.

```
products = ['product 1', 'product 2', 'product 3']

console.log(products[1])
```

Recordemos que si la posición no existe **JavaScript** de todos modos retornará algo. ¿Se les ocurre qué será?

Métodos de arrays

Métodos de arrays

A la hora de trabajar con los **arrays**, necesitamos saber que existen métodos que nos permiten realizar muchas funcionalidades predeterminadas por **JavaScript**, para no tener que crear nosotros las funciones correspondientes.

Esto nos facilitará muchísimo la manipulación de los **datos** y nos permitirá desarrollar funcionalidades avanzadas sin tener que detenernos a pensar en el cómo de cosas más sencillas o pequeñas.

Por supuesto que para poder lograr eso, necesitamos conocer cómo funcionan estos métodos predeterminados.

Métodos de arrays

Algunos de los métodos de arrays más utilizados son:

- **.push()** permite agregar un elemento en la última posición de un array
- **.pop()** permite quitar el último elemento de un array
- **.join()** retorna un string que une los elementos del array según algún criterio (por defecto, una coma)
- **.indexOf()** retorna el índice del primer elemento que coincida con lo que se le pasa por parámetro
- **.lastIndexOf()** retorna el índice del último elemento que coincida con lo que se le pasa por parámetro
- **.includes()** retorna true/false dependiendo de si encuentra en el array lo que se le dé como parámetro
- **.slice()** retorna “una porción” del array
- **.splice()** permite quitar o agregar elementos en cierta parte de un array
- **.shift()** permite quitar el primer elemento de un array
- **.unshift()** permite agregar un elemento en la primera posición de un array

Métodos de strings

Muchos de estos mismos métodos pueden ser utilizados para trabajar con **strings**, ya que, en muchos sentidos, para **JavaScript** los **strings** son **arrays** de caracteres simplemente.

Esto nos permite hacer muchas cosas con los **strings**, aunque veremos que éstos también tienen sus limitaciones.

Objetos Literales

Objetos en JavaScript

En **JavaScript** los objetos son un **tipo de dato** que permite almacenar pares de datos llamados ***key*** (llave o propiedad) y ***value*** (valor). Por lo que también se los conoce como **key-value pairs**.

```
2   const futbolista = {
3       nombre: 'Lionel',
4       apellido: 'Messi',
5       edad: 34,
6       retirado: false
7   };
```

Objetos en JavaScript

Nosotros podemos acceder a los datos almacenados en estos **objetos** utilizando la nomenclatura del punto, ya sea para acceder a sus **métodos** (funciones dentro de los mismos) o a sus propiedades.
Para esto podríamos simplemente llamarlos con un ***console.log*** que retornará el dato correspondiente.

```
11    futbolista.nombre       // Lionel
12    futbolista.apellido     // Messi
13    futbolista.edad         // 34
14    futbolista.retirado     // false
```

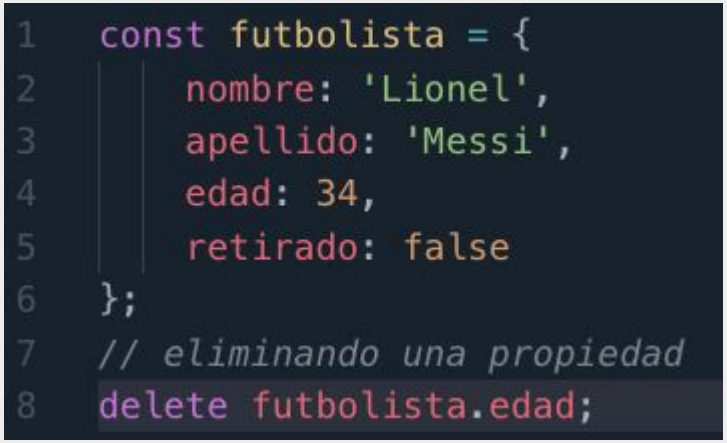
Objetos en JavaScript

¡También podemos **añadir** propiedades!
Invocando al **objeto** por su **nombre**, colocando un **punto** y, a continuación, el **nombre de la nueva propiedad**, a la cual se le asigna un **valor**.

```
1  const futbolista = {
2      nombre: 'Lionel',
3      apellido: 'Messi',
4      edad: 34,
5      retirado: false
6  };
7  // añadiendo una propiedad
8  futbolista.nacionalidad = 'Argentina';
```

Objetos en JavaScript

¿Podemos **eliminar** propiedades?
¡Claro!
Colocando la palabra reservada **delete** invocando al **objeto** por su **nombre**, colocando un **punto** y, a continuación, el **nombre de la nueva propiedad**, a la cual se va a eliminar.



```
1  const futbolista = {
2      nombre: 'Lionel',
3      apellido: 'Messi',
4      edad: 34,
5      retirado: false
6  };
7  // eliminando una propiedad
8  delete futbolista.edad;
```

Objetos en JavaScript

¿Y si no queremos **eliminar** una propiedad, pero modificar su valor?
¡Claro!
Colocando la palabra reservada **delete** invocando al **objeto** por su **nombre**, colocando un **punto** y, a continuación, el **nombre de la nueva propiedad**, a la cual se va a eliminar.

```
1  const futbolista = {
2      nombre: 'Lionel',
3      apellido: 'Messi',
4      edad: 34,
5      retirado: false
6  };
7  // modificando una propiedad
8  futbolista.edad = 35;
```

Objetos y Arreglos

A diferencia de los **arreglos**, que nos permiten guardar elementos **relacionados entre sí**, los **objetos** nos permiten almacenar información **sobre un mismo elemento**.

```
2   const futbolista = {
3       nombre: 'Lionel',
4       apellido: 'Messi',
5       edad: 34,
6       retirado: false
7   };
8
9   const futbolistas = ['Lionel Messi', 'Cristiano Ronaldo', 'Kevin De Bruyne'];
```

Objetos y arreglos

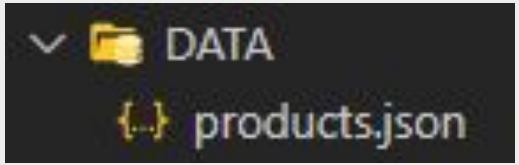
Dentro de estos **objetos** podemos almacenar todo tipo de **datos**, incluyendo otros **objetos** o **arrays**, ¡e inclusive **funciones**!, y de esta manera generar una estructura muy compleja que nos permite guardar **datos** de una manera muy organizada.

```
1  const futbolista = {
2      nombre: 'Lionel',
3      apellido: 'Messi',
4      edad: 34,
5      clubes: ['Barcelona', 'PSG'],
6      saludar: function () {
7          return 'Hola, mi nombre es Lionel Messi';
8      }
9  };
```

JSON
(JavaScript Object Notation)

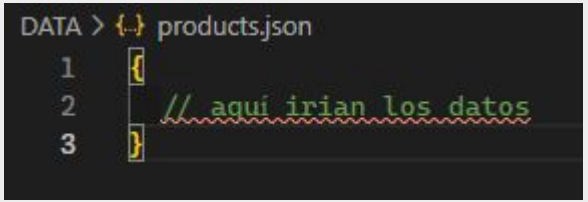
JSON

Los JSON son un formato o tipo de archivo muy similares a los objetos literales, que nos permiten almacenar información de una manera muy ordenada y metódica, ya que deben respetarse sus reglas sí o sí para que éste pueda funcionar.



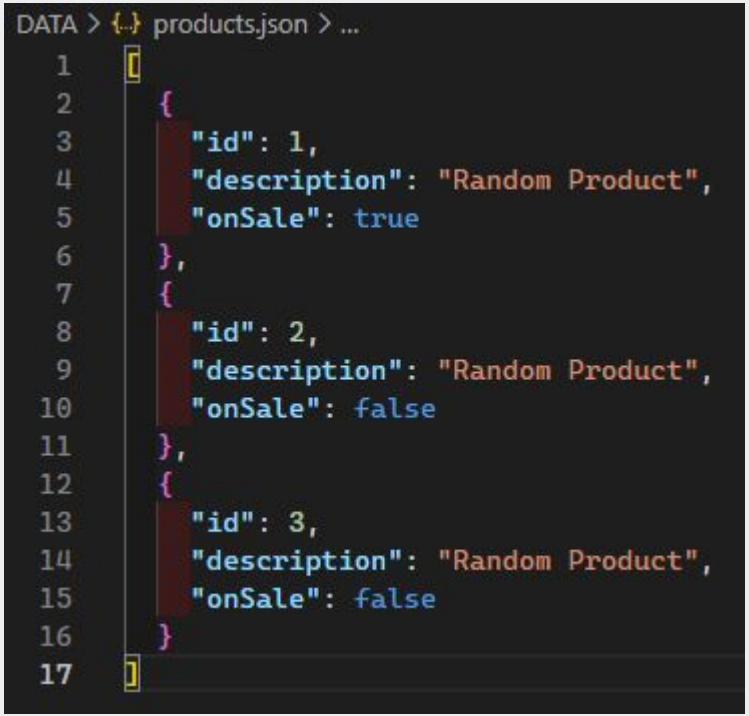
JSON

Un **JSON** puede ser o bien un **array** o un **objeto literal**, pero con algunas consideraciones: el contenido de un **JSON** no debe almacenarse en una variable o constante, sino que debe ser **literalmente** el dato a guardar, es decir el **array** u **objeto literal** puro.
Importante destacar que al ser archivos puramente de datos, no podemos cometer errores sintácticos ni utilizar comentarios.



JSON

Teniendo esto en cuenta, dentro de estos **objetos literales** o **arrays** podemos guardar cualquier dato como lo haríamos normalmente, con la diferencia de que las ***keys*** o ***propiedades*** del objeto literal deben ir **siempre** entre comillas, como **string**. Más adelante veremos cómo podemos trabajar con estos **JSON** para leerlos, editarlos, sobreescribirlos, etc.



Inicializando un proyecto

Para comenzar, lo primero que debemos hacer es crear un proyecto de Nodejs utilizando el comando:

npm init -y

Primero vemos la llamada al comando **npm**, que nos permitirá hacer uso de las funcionalidades de npm, luego le decimos qué queremos hacer utilizando estas funcionalidades de npm: **init**, es decir inicializar un proyecto, y finalmente le pasamos el *flag* **-y** que nos indica que aceptaremos (diremos *yes* → -y) toda la configuración por defecto.

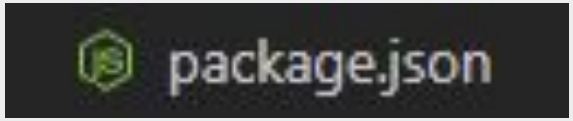
```
❯ npm init -y
```

NPM

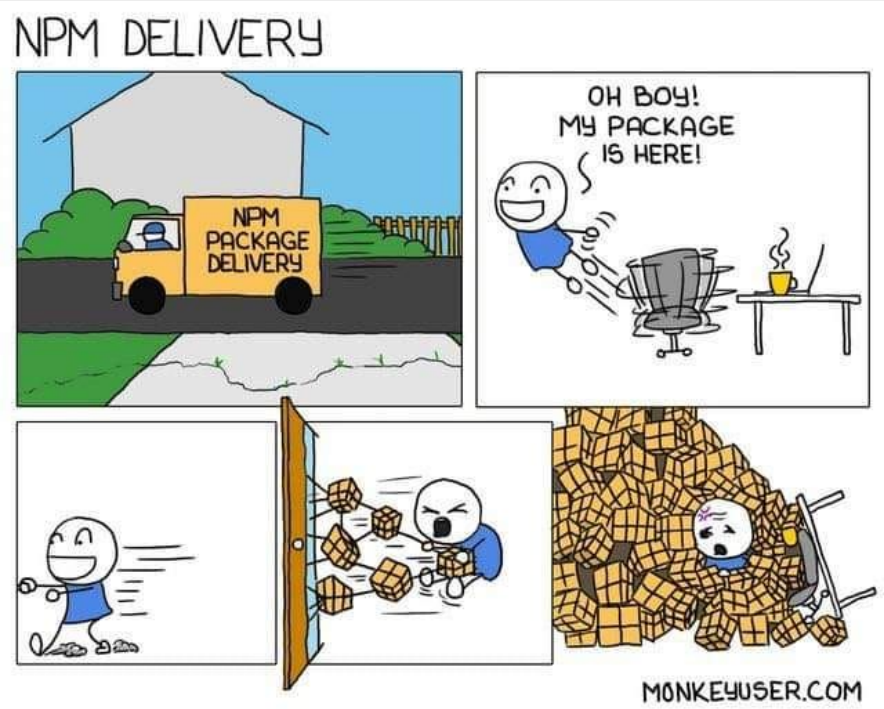
Pero, ¿qué es **NPM**?
NPM son siglas para **Node Package Manager**, o Administrador de Paquetes de Node. Básicamente, nos ayuda a inicializar (instalación) y actualización de paquetes. Los paquetes no son otra cosa que módulos que alguien más –o nosotros mismos– se ha tomado el trabajo de escribir

package.json

Pero, ¿cómo puede hacer eso? A través de **package.json**
Es un archivo automáticamente creado que guarda un listado de todas las dependencias y las versiones de los paquetes que tenemos instalados en nuestra aplicación.
Además, es donde se guarda toda la configuración de nuestro proyecto.

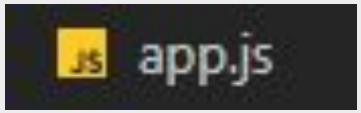


package.json



Entry point

Además necesitaremos un archivo principal que se encargará de servir cómo ***entry point*** (punto de entrada) a nuestra aplicación.
Usualmente se le suele dar el nombre de **app.js**, y es el archivo que contendrá el código principal de nuestra aplicación.
Desde aquí se referenciarán las otras partes de la misma y se harán todos los llamados necesarios para ejecutar las distintas funcionalidades.



process.argv

process

Si hacemos un **console.log** del objeto **process** (que viene incluido con **node.js**) podremos ver que contiene muchísimas cosas.
Dentro de este objeto encontraremos toda la información que tiene **node.js** al correr en nuestra computadora. Utilizando la nomenclatura del punto, podremos hacer uso de cualquier dato o método que tenga dentro.

En esta misma línea, veremos a continuación el **process.argv**.

process.argv

Ahora, si hacemos un **console.log** del objeto **process.argv** podremos encontrar que al correr nuestra aplicación usando el comando **node** y el **nombre** de nuestro archivo (por ejemplo **node app.js**), nos muestra dos rutas…

```
$ node app
[
  'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\Juan\\Desktop\\taller-git-icaro\\app'
]
```

¿De qué podrán ser estas rutas?

process.argv

Como vimos, al utilizar el **process.argv** podemos acceder a todos los argumentos (de ahí el **argv**) que se ingresaron por terminal. Ya con esta data, podríamos comenzar a utilizar esta información para disparar ciertas **funcionalidades**, por ejemplo:

```
if (process.argv[2] === 'saludar') console.log('Hola!')
```

```
$ node app saludar
Hola!
```

process.argv

Ahora sí, estamos en posición de ver qué otras opciones tenemos para poder utilizar los contenidos del **process.argv** para poder leer nuestros **comandos de terminal** y que nuestro código reaccione según lo programamos.

```
const saludar = (name) => {
  console.log(`Hola, ${name}!`)
}

if (process.argv[2] === 'saludar') {
  saludar(process.argv[3])
}
```

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

Muchas gracias!

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