

# La procrastinación se termina hoy.

*"Con orden y tiempo se encuentra el secreto de hacerlo todo, y de hacerlo bien."*

Pitágoras – Filósofo y matemático

## Por hacer

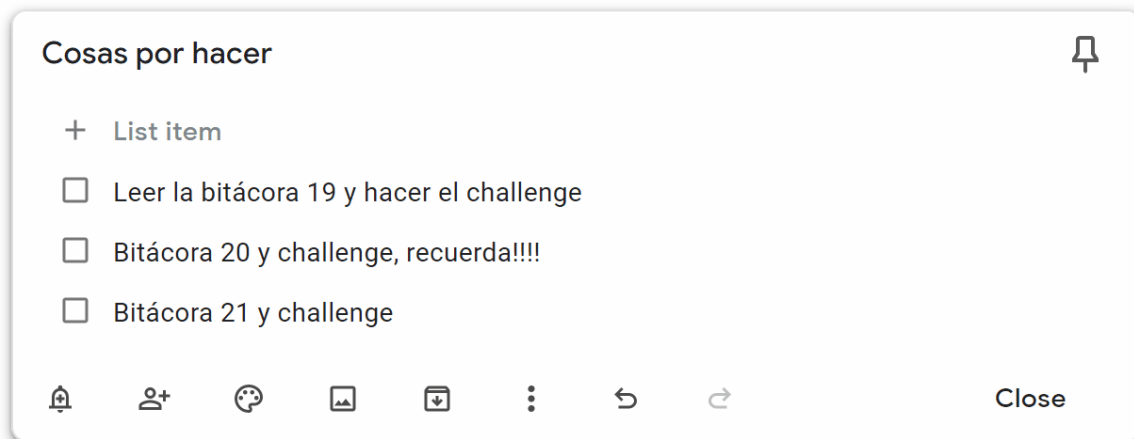
La productividad, entre muchas cosas, es un acto de balance. Organizar nuestro tiempo y lo que hacemos con él puede ser una tarea muy demandante. Hay días en los que nos balanceamos de tarea en tarea, como si estuviéramos atravesando una jungla, de liana en liana.



Por suerte, hay excelentes herramientas y metodologías que nos ayudan a solventar el problema de la organización. De seguro has intentado algunas de ellas para mantener organizado este proceso de aprendizaje en el que te encuentras ahora.

El **to-do list** (o *lista de cosas por hacer*) es la pieza por excelencia para registrar lo que tenemos pendiente, así como para marcar lo ya hecho. A nivel gráfico consiste en una lista de ítems, con un título descriptivo, así como el mecanismo para ingresar nuevas tareas, y marcarlas como terminadas. Hay algo increíblemente satisfactorio en ir marcando tarea a tarea como “completada”.





Hay una plétora de aplicaciones que solventan el problema de administrar tareas pendientes. En el ejemplo de arriba te mostramos [Google Keep](#), una aplicación web para tomar notas.

Quizás tu instinto de developer esté empezando a salir y te estés preguntando, ¿cómo puedo programar algo así? Si es así, estás de suerte, porque en esta toolbox te vamos a dar una primera aproximación a cómo construir listas de elementos, con la estructura de datos llamada [array](#).

## (+) variables

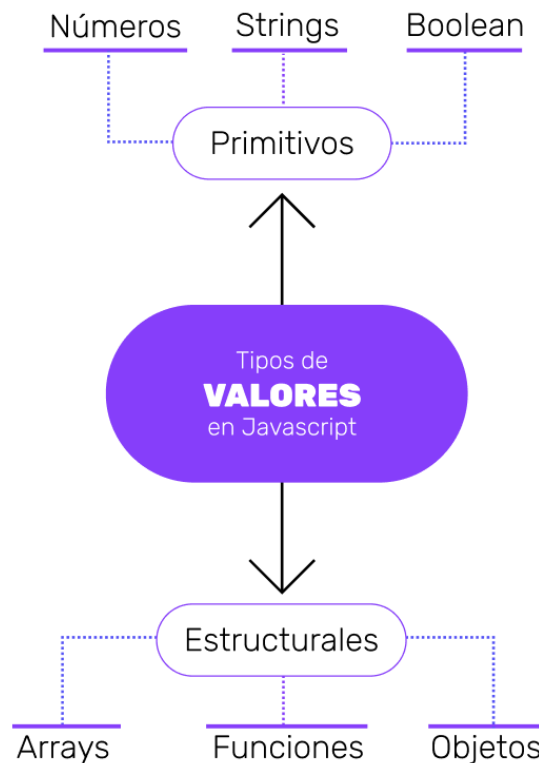
Ahora que sabes qué son las variables y cómo manipularlas, ¡retomemos! ¿Recuerdas los tipos de datos que hemos visto hasta ahora?

En Javascript, las variables de tipo [string](#), [números](#) y [booleanos](#) pertenecen a una categoría llamada “**variables primitivas**”. Esto significa que el valor de estas variables representa solo una unidad de datos, expresada en su tipo. Una variable de número solo puede ser un número, y un booleano solo puede ser verdadero o falso. Este tipo de variables se consideran “atómicas” por proveer un solo tipo de unidad de datos.

Además de la primitiva, tenemos las “**variables estructurales**”, cuya característica principal es que pueden contener más de un tipo de dato en su definición. Si nos vamos con la analogía de los datos primitivos como átomos, un dato estructural sería entonces una molécula, ya que se compone de muchos átomos. Dentro de esta categoría tenemos 3 ítems, de los cuales ya conoces



uno. Presta atención al siguiente diagrama:



Dentro de la categoría de datos estructurales, tenemos las **funciones** (¡las conoces!), junto a los **objetos** y **arrays**. En la toolbox de hoy nos concentramos en los arrays.

## Arrays

Un array es una variable que te permite almacenar una lista de variables secuenciales. Si, así como leíste, una variable que guarda variables. Consideralo como un contenedor de variables primitivas, con superpoderes. Como recordarás, para declarar una variable primitiva, podrás escribir lo siguiente:

...

```
let nombre = "Cristian";
```

...

Pero, supongamos que nuestra aplicación necesita almacenar más nombres, por lo que podemos escribir lo siguiente:



```
...
```

```
let nombre1 = "Cristian";
```

```
let nombre2 = "Laura";
```

```
let nombre3 = "Juan";
```

```
...
```

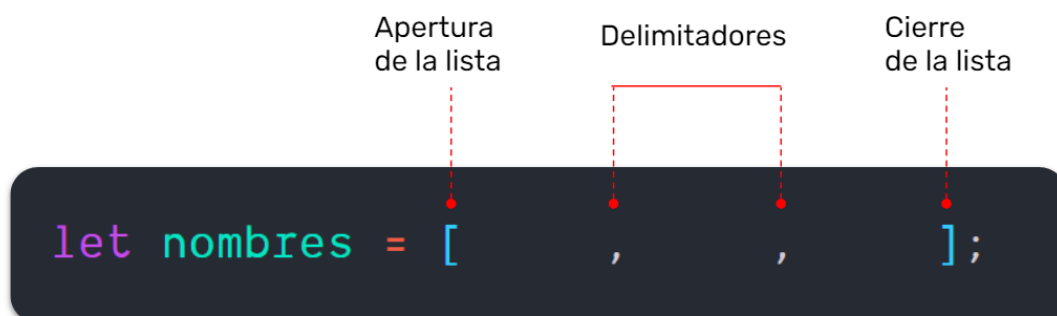
Lo anterior lo podemos expresar con un array de la siguiente forma:

```
...
```

```
let nombres = ["Cristian", "Laura", "Juan"];
```

```
...
```

¡Y *voilà*! El código anterior declara un **array**, o lista, llamada `nombres` la cual almacena 3 datos de tipo string, con nombres. Observa la sintaxis con más detenimiento:



Dentro de un mismo array podemos guardar varios tipos de datos a la vez,, primitivos o estructurales, cada uno en su posición. Imagina un array como una lista de contenedores, en el cual podremos almacenar lo que queramos.

```
let lista = [ valor, valor, valor, valor, valor, valor ]
```

Para crear un array debemos utilizar **corchetes** para iniciar y cerrar la lista, y cada ítem que esté contenido dentro de estos, los separaremos con la coma. El

array es una estructura de datos perfecta para almacenar ítems en un administrador de tareas, así que podemos aplicar lo anterior para construir lo siguiente:

...

```
let tareas = [  
  "Terminar de leer la bitácora 21",  
  "Hacer el challenge",  
  "Empezar a ver The OA en Netflix",  
  "Aprender sobre arrays"  
];  
...
```

El código anterior difiere de la declaración de arrays que te mostramos antes, pero esencialmente es el mismo, solo que está indentado para ofrecer una mayor legibilidad. Lo mejor de todo es que Javascript no tiene problemas con que ocupemos una línea para cada ítems del array.

Ahora que tenemos la lista de nuestras tareas, ¿que podemos hacer con ellas?. En esta toolbox te vamos a demostrar cómo imprimir por consola las tareas de la lista, mientras que en el próximo encuentro vamos a agregar y borrarlas.

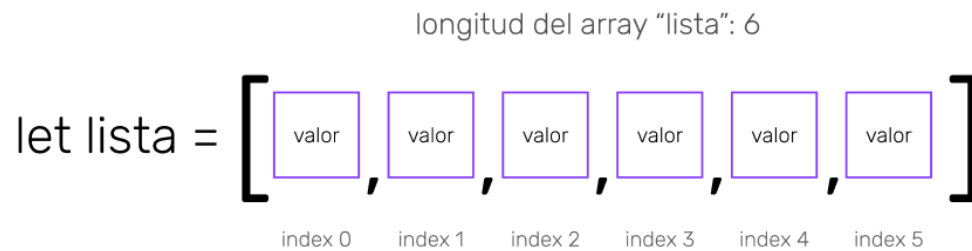
## Leer elementos de un array

No queremos sonar trillado, porque ya hemos usado y reusado la siguiente frase en las toolboxes y durante en nuestros encuentros, pero **"existen varias formas"** de leer elementos de un array. Cuando decimos "leer" nos referimos a la forma de poder consultar el valor de alguno de sus ítems internos.

Al ser una lista, un array tiene una "longitud" expresado en un número. Este representa la cantidad de elementos, variables o ítems que se encuentran dentro de él. El array `tareas` tiene una longitud de 4, por que hay cuatro ítems en el.

Para leer un elemento del array, podemos especificar su número de `index`, que sería la posición en la que se encuentra dentro de la lista, pero **atención**, los arrays empiezan a contar desde cero.





Para imprimir un valor del array por consola, debemos escribir el nombre del array junto a un par de corchetes `[ ]`, y dentro de estos, especificar el número de index.

...

```
let tareas = [
  "Terminar de leer la bitácora 21",
  "Hacer el challenge",
  "Empezar a ver The OA en Netflix",
  "Aprender sobre arrays"
];
console.log(tareas[0]);

// imprime "Terminar de leer la bitácora 21"

console.log(tareas[3])

// imprime "Aprender sobre arrays"
```

...

Si intentamos especificar un index que no existe en el array, obtendremos un valor indefinido a través del mensaje `undefined`.

## Ciclos

Si queremos mostrar todas las tareas del array, podríamos escribir cuatro `console.log` indicando en cada uno, el index de la tarea a mostrar. Pero con Javascript, tenemos una mejor forma de hacer esto empleando **ciclos** o procesos iterativos. **Un ciclo es una instrucción que se repite tantas veces según una condición.**

En Javascript existen distintos tipos de ciclos, con más y menos características especiales. El primer ciclo que vamos a ver en este sprint es el `for`. Este ciclo repetirá el código que está dentro del bloque que se forma entre con las llaves



tantas veces según se cumpla una condición lógica, observa el siguiente ejemplo.

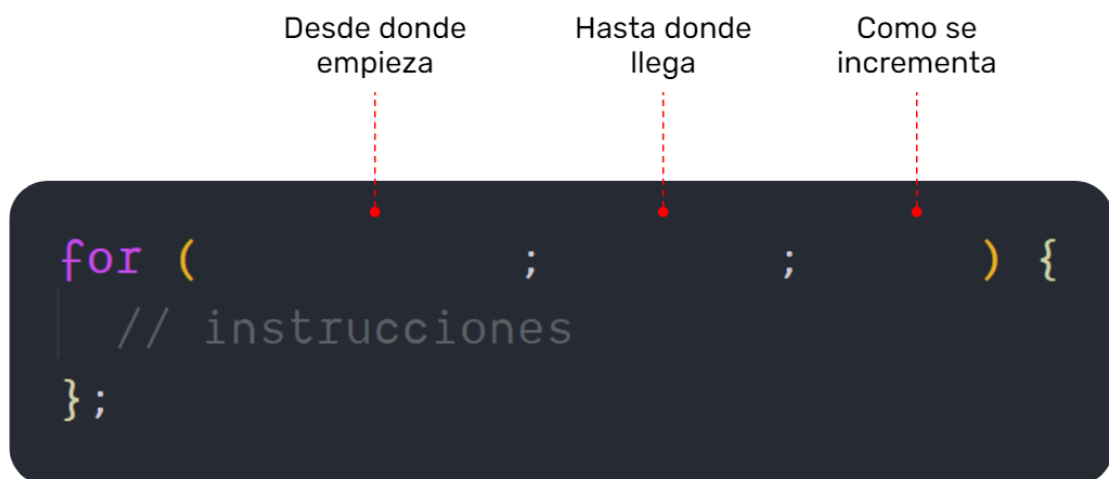
...

```
for (let index = 0; index < 4; index++) {  
  console.log(index)  
}  
...
```

El código anterior va a imprimir por consola los números del 0 al 3. Observa el siguiente sandbox para ver el ejemplo en vivo:

Sandbox: [ciclo for](#)

El ciclo `for` se define utilizando 3 expresiones dentro de un parentesis, separadas por punto y coma `;`:



1. El número que indica **desde donde empieza** el ciclo.
2. La condición booleana que indica **hasta donde llega**
3. Y finalmente, que tanto se incrementa el número que indica desde **donde empieza**

En este video te contamos un poco más sobre cómo trabaja este ciclo:

[Ciclo for](#)



## Combinando conceptos

En base a lo anterior, si queremos plasmar en código un ciclo que imprima por consola los elementos de un array, podemos hacer lo siguiente:

```
...

let tareas = [
  "Terminar de leer la bitácora 21",
  "Hacer el challenge",
  "Empezar a ver The OA en Netflix",
  "Aprender sobre arrays"
];

for (let index = 0; index < 4; index++) {
  console.log(tareas[index])
}
...
```

Si recuerdas, para leer un ítem del array, debemos escribir `tareas[n]` donde `n` es el número del index, o posición dentro del array. Dentro del ciclo, tenemos acceso al valor del número de la iteración, la cual podemos usar para darle valor a `n`, devolviendonos así el elemento perteneciente a ese index del array.

## Longitud del array

La lista de tareas irá modificándose durante el tiempo, tomando una longitud más o menos larga. Irás completando algunas, mientras que otras desaparecerán. Para adaptar el ciclo a estos cambios, los arrays tienen una propiedad especial llamada `length` la cual indica la longitud del mismo. Para acceder a esta propiedad solo debemos agregar `.length` al nombre de un array, de esta forma:

```
...

let nombres = ["Hansel","Gretel"];
console.log(nombres.length) // 2
...
```

Si incorporamos esto al ciclo anterior, podríamos tener lo siguiente:

```
...

let tareas = [
  "Terminar de leer la bitácora 21",
  "Hacer el challenge",
  "Empezar a ver The OA en Netflix",
```





```
"Aprender sobre arrays"  
];  
  
for (let index = 0; index < tareas.length; index++) {  
  console.log(tareas[index])  
}  
...
```

De esta forma cada vez que la aplicación se ejecute, la duración del ciclo va a adaptarse dinámicamente a la longitud del array, mostrando por consola los ítems que se encuentren guardados en él.

## Cerrando

En programación es común (y necesario) trabajar con estructuras de datos que nos permitan definir valores de variables más extensos. El array es una estructura que presenta una lista para almacenar valores de cualquier tipo. En esta toolbox te presentamos un ejemplo donde se guardan varias cadenas de caracteres, o strings, en un array, pero en un mismo array podemos guardar varios tipos de datos en simultáneo.

Los arrays están presentes en las aplicaciones más sencillas, permitiendo guardar listas de “cosas”. Una playlist de Spotify es un array de canciones, las stories de instagram es un array de videos, los hoteles resultantes de una búsqueda de Airbnb, están dentro de una lista por igual.

Al trabajar con listas, tenemos que emplear mecanismos iterativos que nos permitan con una sola instrucción, mostrar un conjunto total o parcial de estos elementos listados, y en la toolbox de hoy te comentamos del ciclo for, una forma condicional de mostrar elementos según un rango numérico.

## ¡Prepárate para la próxima meeting!





### Comunidad

La red de developers de Mozilla ofrece una de las documentaciones más completas que existen sobre Javascript. Es muy posible que cuando busques un tópico de Javascript en Google, la primera coincidencia sea de MDN.

[Está sección de MDN](#) tiene mas información sobre los tipos de ciclos que se ofrecen en Javascript, incluyendo sobre el que te contamos hoy: el ciclo for.




**Challenge**

-  1. Escribe un ciclo que muestre por consolas los números del 1 al 10.
-  2. Escribe un ciclo que muestre por consolas los números del 10 al 1.
-  3. Escribe un ciclo que vaya desde 0 hasta 30 y, si el número de iteración es par, imprimir por consola “emparedado”.
-  4. Crea la función `repetir(n)` la cual ejecute un ciclo que empezará siempre desde 0, pero que termina en `n`, mostrando por consola cada iteración. Por ejemplo:

```
repetir(10);
```

Deberá mostrar por consola los números del 1 al 10.

-  5. Crea la función `repetirMensaje(n, mensaje)` la cual ejecuta un ciclo que empezará siempre desde 0, pero que termina en `n`, repitiendo por consola un mensaje especificado. Por ejemplo:

```
repetirMensaje(5, "debo aprender sobre ciclos");
```

Deberá mostrar por consola el string “debo aprender sobre ciclos” cinco veces.

**Potencia tu Talento - Oportunidades de desarrollo**

¿Recuerdas que antes hablamos de referentes y comunidades? Ahora que has avanzado un poco más en el programa, puedes comenzar a pensar hacia qué área te gustaría especializarte y conectar con los referentes de esa área.

¿Todavía no lo tienes claro? ¡No te preocupes! Anímate a compartir todas tus dudas en la comunidad y contactar a quienes trabajan en distintas especializaciones. ¿Sabes qué especialidad tiene tu squad lead? Puedes empezar por conversar sobre su recorrido si aún no lo han hecho.

 [Oportunidades de carrera en Desarrollo Web](#)

