**Clase 1. Variables**

Las variables son contenedores de memoria para almacenar datos que pueden ser consultados y/o modificados durante el ciclo de vida del programa.

Al generar una variable existen dos estados:

1. Declarar Variable: Consiste en identificar el nombre de una variable, y estos nombres son llamados identificadores.
2. Inicializar la variable: Asignarle un valor a una variable.

Actualmente en JavaScript existen diferentes maneras de declarar una variable:

* **VAR:** Declara la variable que estarán disponibles en cualquier parte del código.
* **LET:** Declara variables con alcance limitado a un bloque de código.
* **CONST:** Declara variables de solo lectura. También son limitados a un bloque de código.

Recordad que JavaScript es un lenguaje débilmente tipado, esto significa que el valor de una variable puede variar de string a número, booleano, objeto o arreglos.

**Clase 2. Variables: String**

Los strings son cadenas de texto. Para indicar que estamos usando una cadena de texto debemos de colocar las comillas simples. Existen funciones propias para este tipo de variables como, por ejemplo:

* toUpperCase: Sirve para transformar un string a mayúsculas.
* toLowerCase: Sirve para transformar el string a minúsculas.

También tiene atributos como por ejemplo **length**, que nos indica la cantidad de caracteres que tiene un string.

Para concatenar dos strings se utiliza el símbolo (+)



También existe una forma mas nueva de concatenar variables que se conoce como interpolación de texto. En el cual las variables se deben colocar dentro de comillas invertidas con los signos ${}.



**Clase 3. Variables: Números**

En JavaScript se realizan distintas operaciones con los números, ya sean con sumas, restas, multiplicación y división. Existen muchas funciones que ayudan al desarrollador, tales como el redondeo de decimales, convertir un string a tipo numéric, etc.

**Clase 4. Funciones**

Las funciones son fracciones de código reutilizable. Para definir una función utilizaremos la palabra reservada **function**, delimitamos el cuerpo de la función usando llaves { }. Los parámetros de la función son variables que se pasan a la función escribiéndolos entre paréntesis ().

Definir funciones nos sirve para reutilizar código. JavaScript es un lenguaje interpretado, esto quiere decir que intentará ejecutar el código sin importar si los parámetros que le pasemos a la función estén invertidos o incluso incompletos.

**Clase 5. Alcance de las funciones**

Si una variable no está definida dentro del cuerpo de una función hablamos de una variable global. Por el contrario, una variable definida dentro de una función es una variable local.

Para que la ejecución de una función no modifique una variable global usamos parámetros en lugar de pasar directamente la variable. Es posible utilizar el mismo nombre para una variable global y para el parámetro de una función con un alcance local.

**Clase 6. Objetos**

Los objetos se definen delimitados mediante llaves {}.

Un atributo se compone de una clave (key) y un valor (value), que se separan entre sí por dos puntos **“:”**. Los valores pueden ser de tipo string, número, booleano, etc. Cada atributo está separado del siguiente por una coma. Un objeto puede tener todos los atributos que sean necesarios.

Escribir el nombre de un objeto separado por un punto del nombre de un atributo, nos permite acceder al valor de dicho atributo para ese objeto. Un objeto también se puede pasar como atributo en una función.

Las últimas versiones de JavaScript nos permiten desglosar el objeto para acceder únicamente al atributo que nos interesa. Esto se consigue encerrando el nombre del atributo entre llaves { }.

**Clase 7. Desestructurar Objetos**

Existe otra forma de acceder a los atributos de los objetos que es la desestructuración de los mismos. Para no duplicar las variables introducir el nombre de la variable como parámetro de la segunda variable: Ej:



**Clase 8. Parámetros como referencia o como valor**

JavaScript se comporta de manera distinta cuando le pasamos un objeto como parámetro. Cuando los objetos se pasan como una referencia, estos se modifican fuera de la función. Para solucionar esto se puede crear un objeto diferente. Esto lo podemos hacer colocando tres puntos antes del nombre.

**Clase 9. Comparaciones en JavaScript**

Existen varias maneras de comparar variables u objetos dentro de JavaScript. En el primer ejemplo le asignamos a X un valor numérico y a Y un string. Para poder compararlos debemos agregar dos signos de igual (==). Esto los convierte al mismo tipo de valor y permite que se puedan comparar.

Cuando realizamos operaciones es recomendable usar tres símbolos de igual (===). Esto permite que JavaScript no iguale las variables que son de distinto tipo. Te recomendamos que uses el triple igual siempre que estés comparando variables.

Existen cinco tipos de datos que son primitivos:

1. Boolean
2. Null
3. Undefined
4. Number
5. String

**Importante:** Si se comparan objetos, lo más probable es que la comparación sea false. Esto se debe a que los objetos apuntan a un lugar en la memoria RAM a diferencia de la comparación entre variables de valores primitivos ya que, en ese caso, si se compara dato y valor.

**Clase 10. Condicionales**

Las estructuras de control nos permiten decidir el flujo de nuestro código. Una de ellas, es son los condicionales, los cuales nos permiten decidir si un código se ejecuta o no. Para estos casos se utilizan el valor primitivo booleano, que determina si un valor es falso o verdadero. Mediante un condicional (if) decidiremos si se ejecuta una parte de nuestro código cuando se cumpla o no cierta condición.

**Clase 11. Funciones que retornan valores**

En esta clase seguiremos trabajando con condicionales para desglosar las funciones en funciones más pequeñas que retornen un valor.

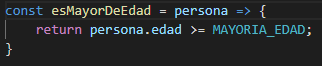
Debemos de tener en cuenta que el número 18 está incluido dentro del rango de edad. Para ello utilizamos los símbolos >=.

Return detiene la ejecución de una función y devuelve el valor de esa función.

Las variables definidas con **const** se comportan como las variables, excepto que no pueden ser reasignadas. Las constantes pueden ser declaradas en mayúsculas o minúsculas. Pero por convención, para distinguirlas del resto de variables, se escribe todo en mayúsculas.

**Clase 12. Arrow functions**

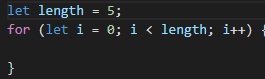
Las Arrow Functions nos permiten una nomenclatura más corta para escribir expresiones de funciones. Este tipo de funciones deben definirse antes de ser utilizadas. Al escribir las Arrow Functions no es necesario escribir la palabra function, la palabra return, ni las llaves. Ej:



**Clase 13. Estructuras repetitivas: For.**

El bucle for, se utiliza para repetir una o más instrucciones un determinado número de veces. Para escribir un bucle for se coloca la palabra for seguida de paréntesis y llaves, dentro de los paréntesis irán las condiciones para ejecutar el bucle, y dentro las llaves irán las instrucciones que se deben repetir.

En este ejemplo la variable i la utilizamos como contador:



**Clase 14. Estructuras repetitivas: While.**

While se ejecuta únicamente mientras la condición que se está evaluando es verdadera. En ocasiones nuestro código puede fallar por errores de syntaxis o errores lógicos. En caso de que quieras verificar tu código, debes utilizar un **debugger**. El código se detiene cada vez que lee esta palabra.



**Clase 15. Estructuras repetitivas: do-while.**

A diferencia de la instrucción while, un bucle do…while se ejecuta una vez antes de que se evalúe la expresión condicional.

**Clase 16. Condicional múltiple: switch.**

Switch se utiliza para realizar diferentes acciones basadas en múltiples condiciones. Prompt, muestra un cuadro de mensaje que le pide al usuario que ingrese alguna información. Break, sirve para que el browser se salte un bucle.

**Clase 17. Introducción a los arrays**.

Los arrays, también conocido como arreglos, son estructuras que nos permiten organizar elementos dentro de una colección. Estos elementos pueden ser números, strings, booleanos, objetos, etc. A través de las colecciones se pueden realizar ciertas funciones.

**Clase 18. Filtrar un array.**

Para filtrar siempre necesitamos establecer una condición. El método **filter** ( ) crea una nueva matriz con todos los elementos que pasan la prueba implementada por la función proporcionada. Es importante mencionar que, si no hay elementos que pasen la prueba, filter devuelve un array vacío.

**Clase 19. Transformar un array.**

El método **map**() itera sobre los elementos de un array en el orden de inserción y devuelve array nuevo con los elementos modificados.

**Clase 20. Reducir un Array.**

El método **reduce**() nos permite reducir, mediante una función que se aplica a cada uno de los elemento del array, todos los elementos de dicho array, a un valor único.

**Clase 21. Cómo funcionan las clases en JavaScript.**

Las clases son funciones cuya sintaxis tiene dos componentes:

* expresiones
* declaraciones

Dentro de una función, el valor de this depende de cómo es llamada ésta.

**Clase 22. Modificando un prototipo.**

JavaScript funciona con una estructura orientada a objetos y cada objeto tiene una propiedad privada que mantiene un enlace a otro objeto llamado prototipo.

El prototipo es objeto más de JavaScript, si se modifica en una cierta parte del código, este será distinto desde ese punto.

**Clase 23. El contexto de las funciones: quién es this.**

Dentro de las arrow function, this hace referencia al espacio global, esto quiere decir que apunta al windows (navegador).

**Clase 24. La verdad oculta sobre las clases en JavaScript.**

Los objetos en JavaScript son “contenedores” dinámicos de propiedades. Estos objetos poseen un enlace a un objeto prototipo. Cuando intentamos acceder a la propiedad de un objeto, la propiedad no sólo se busca en el propio objeto sino también en el prototipo del objeto, en el prototipo del prototipo, y así sucesivamente hasta que se encuentre una propiedad que coincida con el nombre o se alcance el final de la cadena de prototipos.

**Clase 25. Clases en JavaScript**

Las clases de JavaScript son introducidas en el ECMAScript 2015 y son una mejora en la sintaxis sobre la herencia basada en prototipos de JavaScript.

La palabra clave extends se usa en declaraciones de clase o expresiones de clase para crear una clase que es hija de otra clase.

El método constructor es un método especial para crear e inicializar un objeto creado a partir de una clase.

**Clase 26. Funciones como parámetros.**

En JavaScript, los parámetros de funciones son por defecto **undefined**. De todos modos, en algunas situaciones puede ser útil colocar un valor por defecto diferente que lo evalúe como verdadero.

**Cómo funciona el asincronismo en JavaScript.**

JavaScript sólo puede hacer una cosa a la vez, sin embargo; es capaz de delegar la ejecución de ciertas funciones a otros procesos. Este modelo de concurrencia se llama **EventLoop**.

JavaScript delega en el navegador ciertas tareas y les asocia funciones que deberán ser ejecutadas al ser completadas. Estas funciones se llaman **callbacks**, y una vez que el navegador ha regresado con la respuesta, el **callback** asociado pasa a la cola de tareas para ser ejecutado una vez que JavaScript haya terminado todas las instrucciones que están en la pila de ejecución.

Si se acumulan funciones en la cola de tareas y JavaScript se encuentra ejecutando procesos muy pesados, el **EventLoop** quedará bloqueado y esas funciones pudieran tardar demasiado en ejecutarse.

**Clase 27. Cómo funciona el tiempo en JavaScript**

En principio, cualquier tarea que se haya delegado al navegador a través de un **callback**, deberá esperar hasta que todas las instrucciones del programa principal se hayan ejecutado. Por esta razón el tiempo de espera definido en funciones como **setTimeout**, no garantizan que el **callback** se ejecute en ese tiempo exactamente, sino en cualquier momento a partir de allí, sólo cuando la cola de tareas se haya vaciado.

**Clase 28. Callbacks.**

Un **callback** es una función que se pasa a otra función como un argumento. Esta función se invoca, después, dentro de la función externa para completar alguna acción.

**Clase 29. Haciendo múltiples requests.**

Es importante saber que si nosotros ponemos los servicios en el orden que queremos, no garantiza que las respuestas lleguen en el mismo orden, ya que esto dependerá netamente del servicio back que se está consumiendo.

**Clase 30. Manejando el Orden y el Asincronismo en JavaScript.**

Una manera de asegurar que se respete la secuencia en que hemos realizado múltiples tareas es utilizando callbacks, con lo que se ejecutará luego, en cada llamada. Lo importante es que el llamado al callback se haga a través de una función anónima. Sin embargo, al hacerlo de esta manera generamos una situación poco deseada llamada **CallbackHell.**

**Clase 31. Manejo de errores con callbacks**

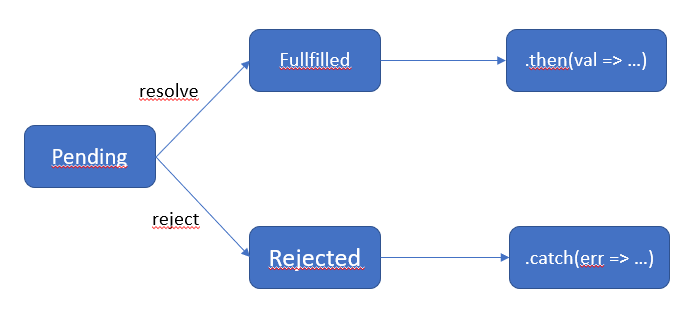
Para solucionar el problema de quedarnos sin conexión, u otro error similar, en medio de una sucesión de callbacks utilizamos el método fail().

**Clase 32. Promesas.**

Pensemos en las promesas como valores que aún no conocemos. Es la promesa que habrá un valor cuando una acción asíncrona suceda y se resuelva.

Las promesas tienen 3 estados:

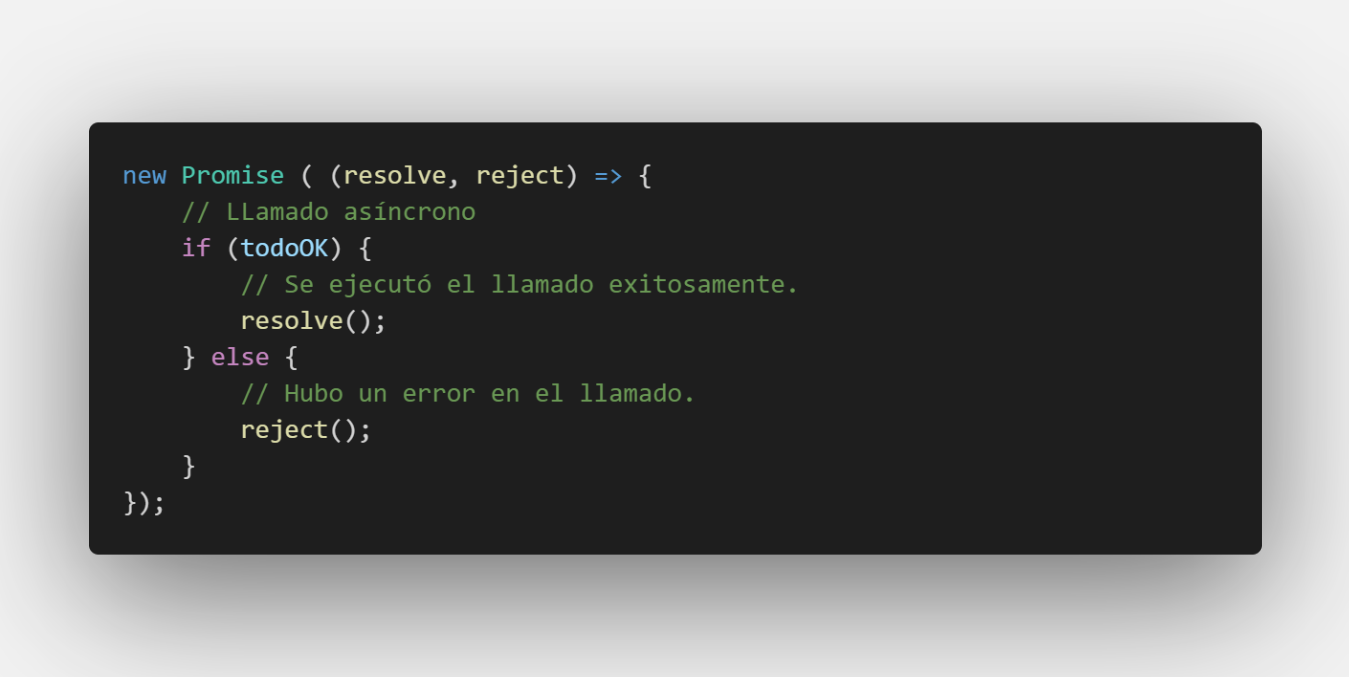
1. **Pending:** Una promesa se inicia en este estado.
2. **Fulfilled:** Cuando se resuelve exitosamente.
3. **Rejected:** Si sucede algún error.



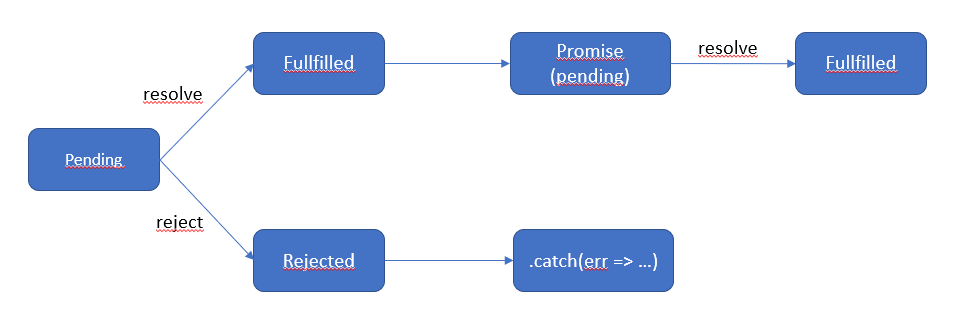
Es importante saber que las promesas también pueden resolverse de manera síncrona.

Para obtener el valor de la resolución de la promesa podemos llamar a la función **then** donde el primer parámetro que recibirá la función es el valor que nosotros esperábamos. En caso de que falle podemos llamar a la función **catch** donde pasamos una función que pasamos como parámetro el error que sucedió.

Las promesas se crean de la siguiente forma:



Luego de llegar al estado de Fullfilled podemos retornar otra promesa dentro de la función **then** de esa manera ir encadenándolas en sucesivas acciones asíncronas (Promesas encadenadas) y cada una de ellas puede ser rechazada o resuelta en una nueva promesa que terminará en el estado de Fullfilled.



**Clase 33. Promesas Encadenadas.**

A diferencia de los callbacks en el CallbackHell, que terminan estando anidados unos dentro de otros, cuando se usan Promesas la ejecución de las llamadas no se hacen de manera anidada sino de manera encadenada, al mismo nivel una debajo de la otra, lo que hace que el código sea mucho más legible y mantenible.

**Clase 34. Múltiples promesas en paralelo.**

Para hacer el llamado a múltiples promesas, nos apoyamos en un array de ids con el que luego construimos otro arreglo de Promesas, que pasaremos como parámetro a Promise.all (arreglo de promesas), con las promesas podemos encadenar llamadas en paralelo, algo que no es posible usando callbacks.

**Clase 35. Async-wait: Lo último en asincronismo.**

Async-await es la manera más simple y clara de realizar tareas asíncronas. Await detiene la ejecución del programa hasta que todas las promesas sean resueltas. Para poder utilizar esta forma, hay que colocar async antes de la definición de la función, y encerrar el llamado a Promises.all() dentro de un bloque try … catch.

**Clases Complemetarias.**

**Clase 36. Var, let y const: Las diferencias entre ellos.**

Una de las recomendaciones para trabajar en JavaScript es reducir siempre al mínimo el alcance de nuestras variables, por lo que se debe usar let en lugar de var mientras sea posible. A continuación, se explicará los distintos tipos de declaraciones que existen actualmente en el lenguaje:

**VAR:** Es la manera más antigua de declarar variables. No es muy estricta en cuanto al alcance, ya que, al declarar variables de esta forma, dichas variables podrán ser accedidas, e incluso modificadas, tanto dentro como fuera de los bloques internos de una función.

**LET:** El alcance se reduce al bloque (las llaves) en el cual la variable fue declarada. Fuera de este bloque la variable no existe. Una vez declarada la variable con let, no se puede volver a declarar en ninguna otra parte de la función.

**CONST:** Al igual que **let** se define en el contexto o alcance de un bloque, a diferencia de let y var, las variables definidas como constantes (const), no podrán ser modificadas ni declaradas nuevamente en ninguna otra parte de la función o el contexto en el que ya existen.

**Clase 37. ¿Hace cuántos días naciste?**

Con las variables del tipo Date, se pueden realizar operaciones de suma y resta similares a las que se realizan con los números. El resultado que se obtiene está en milisegundos, por lo que luego hay que hacer algunas operaciones adicionales para llevarlos a días, meses o años según queramos. También aplica para horas, minutos, segundos y milisegundos.

**Clase 38. Funciones Recursivas.**

La recursividad es un concepto muy importante en cualquier lenguaje de programación. Una función recursiva es básicamente aquella que se llama (o se ejecuta) a sí misma de forma controlada, hasta que sucede la condición base).

**Clase 39. Memorización: Ahorrando cómputo.**

La memorización es una técnica de programación que nos permite ahorrar cómputo o procesamiento en JavaScript, al ir almacenando resultado invariable de una función para que no sea necesario volver a ejecutar todas las instrucciones de nuevo, cuando se vuelva a llamar con los mismos parámetros. Es similar a usar memoria cache.

**Clase 40. Entiende los closures de JavaScript.**

Un closure, básicamente es una función que recuerda el estado de las variables al momento de ser invocada, y conserva este estado a través de reiteradas ejecuciones. Un aspecto fundamental de los closures es que son funciones que retornan otras funciones.

**Clase 41. Estructuras de datos inmutables.**

Las estructuras de datos inmutables forman parte de los principios de la programación funcional y nos permiten evitar tener efectos colaterales en los datos. En otras palabras, que haya modificaciones en las variables sin nuestro consentimiento, produciendo comportamientos inesperados en el programa.

**Clase 42. Cambiando de contexto al llamar a una función.**

El contexto (o alcance) de una función es por lo general, **window**. Así que, en ciertos casos, cuando intentamos referirnos a **this** en alguna parte del código, es posible que tengamos un comportamiento inesperado, porque el contexto quizás no sea el que esperamos.

Existen al menos tres maneras de cambiar el contexto de una función:

* Usando el método .**bind**, enviamos la referencia a la función sin ejecutarla, pasando el contexto como parámetro.
* Usando el método **.call**, ejecutamos inmediatamente la función con el contexto indicado
* Usando el método .**apply**, es similar a .**call** pero los parámetros adicionales se pasan como un arreglo de valores

**Clase 43. ¿Cuándo hace falta poner el punto y coma al final de la línea?**

El punto y coma es opcional es JavaScript, excepto en algunos casos:

* Cuando usamos varias instrucciones en una misma línea.
* Al comenzar la próxima con un array.
* Al comentar la próxima línea con un template string.

**Distintas funciones para recorrer, filtrar, agregar o quitar data en los arrays.**

Teniendo el siguiente arreglo como ejemplo, aplicaremos los diferentes métodos que se pueden aplicar en los arrays en JavaScript con el fin de repesar conceptos del curso.



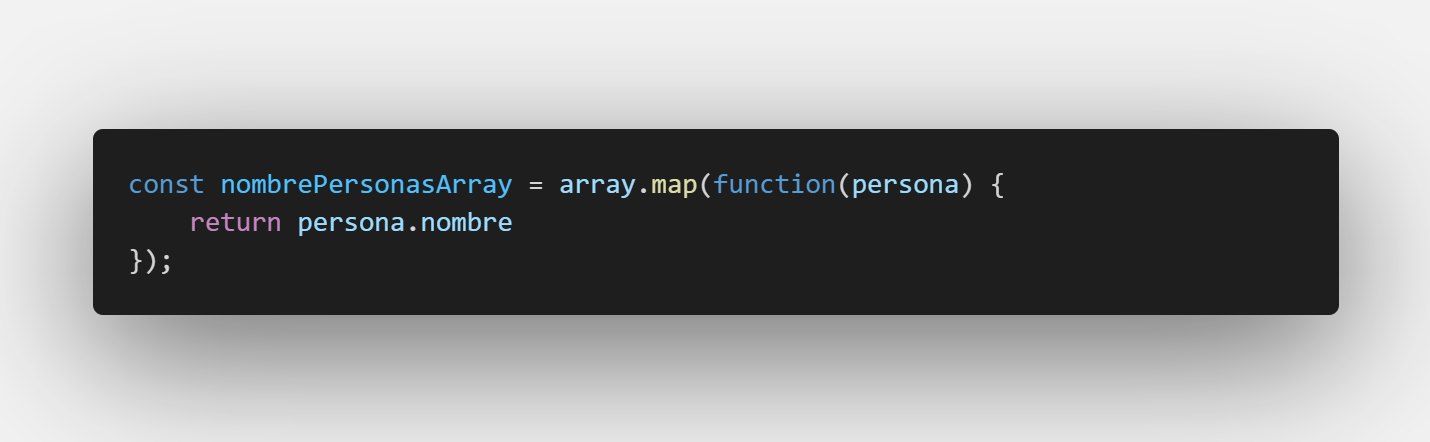
**Método Filter:**

Ayuda a filtrar ciertas cosas en un arreglo, validando si algo es verdadero o falso agregándolo a un nuevo array, por lo que no modifica el array con el que estás trabajando.



**Método Map:**

Ayuda a mapear ciertos datos. Por ejemplo, en el siguiente ejemplo obtendremos un nuevo array utilizando la función **map**, la cuál recuperará los nombres de las personas que están dentro del array.



**Método Filter:**

Tiene la finalidad de encontrar una variable solicitada dentro del arreglo, al igual que las funciones anteriores crea un nuevo array.



**Método ForEach:**

Ejecuta la función indicada una vez por cada elemento en el array. Este método no genera un nuevo array, solo lo filtra y devuelve las variables sobre la base de los parámetros otorgados.



Otros métodos que se utilizan constantemente:

* Some: Genera un nuevo array entregando la validación de true o false.
* Push: Agrega elementos al final del array.
* Pop: Elimina el último elemento del array.
* Shift: Elimina el primer elemento del array.
* Unshift: Agrega un elemento al comiendo del array