**Curso Básico de JavaScript.**

**¿Qué es JavaScript?**

**¿Cómo nace Javascript?**  
Nace con la necesidad de generar dinamismo en las páginas web y que a su vez los usuarios y las empresas pudieran interactuar unos con otros.

**¿Qué es Javascript?**  
Es un lenguaje interpretado, orientado a objetos, débilmente tipado y dinámico.

**Débilmente tipado**  
Se pueden hacer operaciones entre tipos distintos de datos (enteros con strings, booleanos con enteros, etc). Ejemplo:



**Dinámico**  
Corre directamente en la etapa de Runetime sin una etapa de compilación previa. Esto permite probar nuestro código inmediatamente; pero también es lo que hace que los errores se muestren hasta que se ejecuta el programa.

**¿Realmente es Javascript un lenguaje interpretado?**  
Si, y la razón es que el navegador lee línea por línea nuestro código el cuál le indica lo que tiene que hacer, sin la necesidad de compilar. Todo esto es controlado por el motor de Javascript V8 del navegador

**Javascript es Backwards Compatible**  
Todas las funciones nuevas que salen de Javascript no dañarán el trabajo ya hecho, pero no se podrá utilizar en nuestro entorno de trabajo inmediatamente. Para solucionar esto, está **Babel** que permite utilizar las nuevas características del lenguaje pero lo transforma a una versión que el navegador pueda entender.

# ¿Por qué JavaScript?

.JavaScript tiene una **comunidad enorme** de desarrolladores que te pueden ir ayudando a generar diferentes cosas.

1. Si solo estuvieras interesado en trabajar **aplicaciones web** tienes muchos frameworks y librerías construidas en JavaScript que te van a ayudar a hacer proyectos de forma mucho más rápida, eficiente y robusta (Angular, View, React, entre otros)
2. Si no quieres trabajar solo en aplicaciones Web puedes utilizar JavaScript con un framework que se llama React Native para poder **construir aplicaciones nativas** como Android y IOS.
3. Puedes construir **aplicaciones de escritorio** con JavaScript, usando un framework llamado Electron, pueden correr en Mac o Windows.
4. También puedes trabajar en la parte del **Back-end** o IOT (Internet of Things) es un concepto que se refiere a una interconexión digital de objetos cotidianos con Internet. Esto con un Framework llamado NodeJS, el cual es un entorno de ejecución de JavaScript que corre directamente en el Back-end.

# 

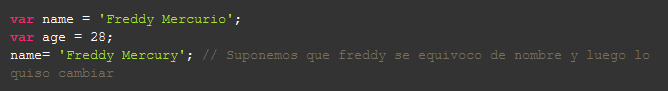
# Elementos de un Lenguaje de Programación: Variables, Funciones y Sintaxis.

# 

# 

# Variables.

**var**: Es una variable que puede cambiar su valor en un futuro y su alcance es local. Surgió en ECMAScript5.  
Ejemplo:



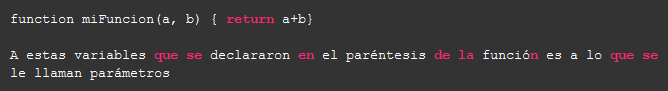
**const**: Es una variable constante que no puede cambiar nunca su valor en un futuro.  
Ejemplo:

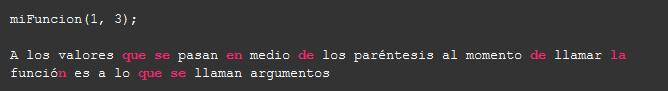


**let**: Es una variable que puede cambiar su valor, pero solo funciona en un bloque donde se declare ( **{let}** )

# 

# Funciones.





Las funciones son las tareas que va a llevar a cabo el navegador.

Existen 2 tipos de funciones  
1) Declarativas  
2) De expresión  
Ambas pueden llevar parámetros, que son los datos que necesitan para ejecutarse.  
Cada parámetro va separado por una coma.  
Cada instrucción que tenga la función debe terminar con ;   
Si queremos que una función nos dé un numero o dato tenemos que usar la siguiente sintaxis:

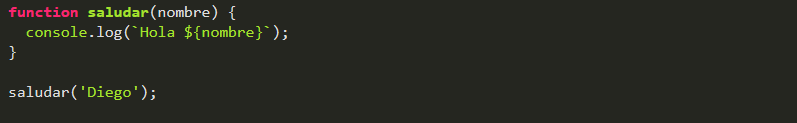
***return*** *dato;*

# ¿Cuándo utilizar una función declarativa y una expresiva?

Cuando hablamos de funciones en JavaScript, tenemos dos tipos de funciones: Funciones Declarativas (function declaration / function statement) y Expresiones de función (function expression / funciones anónimas).

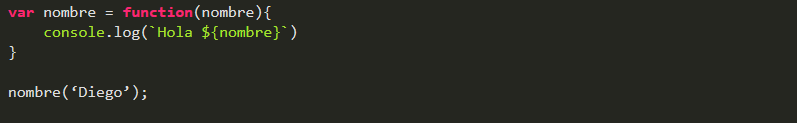
### **Funciones Declarativas:**

En las funciones declarativas, utilizamos la palabra reservada function al inicio para poder declarar la función:



#### **Expresión de función:**

En la expresión de función, la declaración se inicia con la palabra reservada var, donde se generará una variable que guardará un función anónima.



En la expresión de función, la función podría o no llevar nombre, aunque es más común que se hagan anónimas.

### **Diferencias:**

A las funciones declarativas se les aplica hoisting, y a la expresión de función, no. Ya que el hoisting solo se aplica en las palabras reservadas var y function.

Lo que quiere decir que con las funciones declarativas, podemos mandar llamar la función antes de que ésta sea declarada, y con la expresión de función, no, tendríamos que declararla primero, y después mandarla llamar.

# Scope.

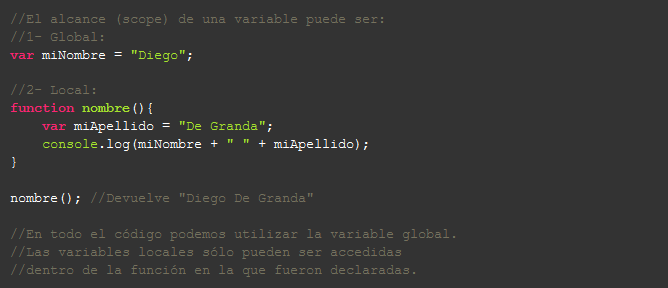
# Scope: Alcance que tienen las variables en el código.

# Existen 2 tipos de sopes.

# Global: Puede ser llamada a lo largo de nuestro programa.

# Local: Solo puede ser llamada dentro del bloque de código en el que se declaro

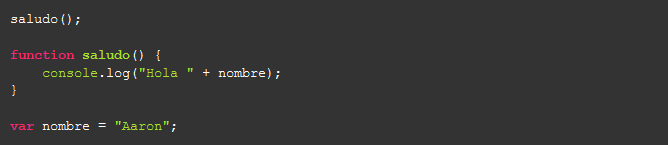
# 



# Hoisting.

El **Hoisting** es un proceso del compilador de JavaScript, que consiste en que la *declaración de las variables* y las *funciones* son llevadas al inicio del código, sin importar su posición, para su procesamiento, sin embargo, la inicialización de las variables no es llevada al inicio del código para su compilación, sino solo su declaración, lo cual suele dar espacio a errores cuando se declara una variable sin inicializarla y se procesa en el código antes de haber llegado a su inicialización, lo cual nos suele dar una variable con valor ***undefined***, ya que la variable sí fue almacenada en memoria, pero no se le asigno un valor hasta después de su ejecución.

Aquí un ejemplo de esto:



El output de este código sería el siguiente:



Debido a que como lo hemos dicho, la variable se tomó en cuenta y se le asigno memoria, sin embargo, el compilador no la inicializo y se le dio el valor de *undefined*, y con ese valor se ingresó a la función, y ya después de correr la función se le asigno el valor.

Este comportamiento se puede entender fácilmente si se comprenden estos dos puntos esenciales:

* Las funciones siempre se mueven arriba del Scope. Por lo tanto, podemos elegir donde declararlas y usarlas.
* La declaración de las variables se mueve arriba del Scope, pero no la asignación. Antes de usar una variable, habrá que crearla y asignarla.

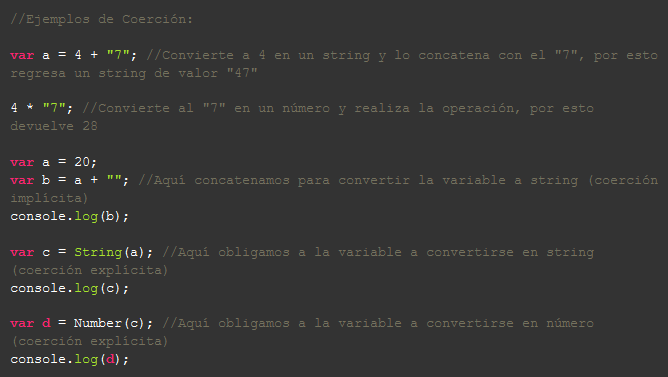
# Coerción.

# 

# Coerción es la forma en la que podemos cambiar un tipo de valor a otro, existen dos tipos de coerción:

# Coerción implícita: es cuando el lenguaje nos ayuda a cambiar el tipo de valor.

# Coerción explicita : es cuando obligamos a que cambie el tipo de valor.



# Valores: Truthy y Falsy.

***¿Qué tipos por default son verdaderos y falsos?***

Usamos la función de JS que es ***Boolean()*** dentro del paréntesis ponemos el valor y nos dice si el mismo el False o True.

–> ***Falsy***

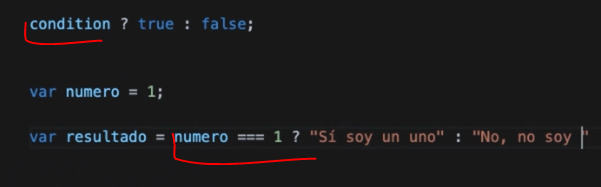
* Boolean() —> sin ningún valor es false
* Boolean(0) —> false
* Boolean(null) —> false
* Boolean(NaN) —> false // NaN es Not and Number
* Boolean(Undefined) —> false
* Boolean(false) —> false
* Boolean("") —> false

–> ***Truthy***

* Boolean(1) —> true //cualquier número que no sea igual a cero es true
* Boolean(“a”) —> true
* Boolean(" ") —> true // siendo un espacio el valor es true
* Boolean([]) —> true // un array nos da un true
* Boolean({}) —> true // un objeto nos da el valor de true
* Boolean(function() {}) —> true //una funcion también es true
* Boolean(true) —> true

Todo esto lo vamos a usar en condiciones esto valida si es verdadero o falso para ejecutar cierta acción.

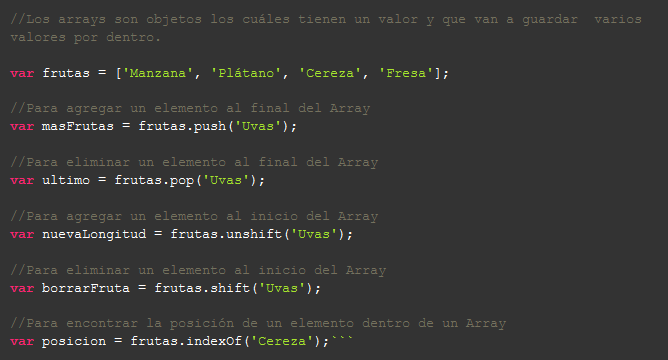
# Otra forma de realizar un If.



# Operadores: Asignación, Comparación y Aritméticos.

# 

# Arrays.



**Mayor documentación:**

<https://devcode.la/tutoriales/manejo-de-arrays-en-javascript/>

# Loops: For y For...of



# Objects.

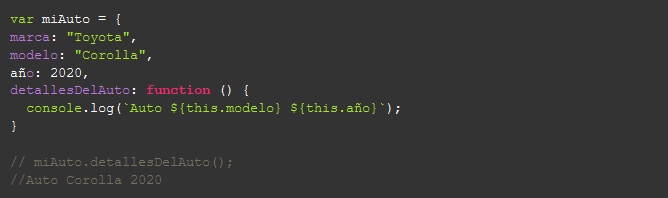
# Para definir un objeto hacemos uso de {}

# 

# Acceder a una propiedad del objeto:

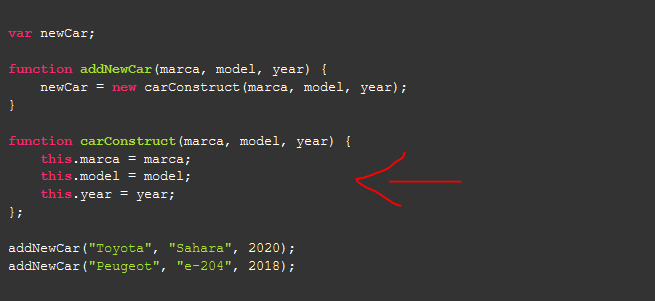


Se pueden agregar propiedades que van a ser una función, se les llama métodos de objetos.

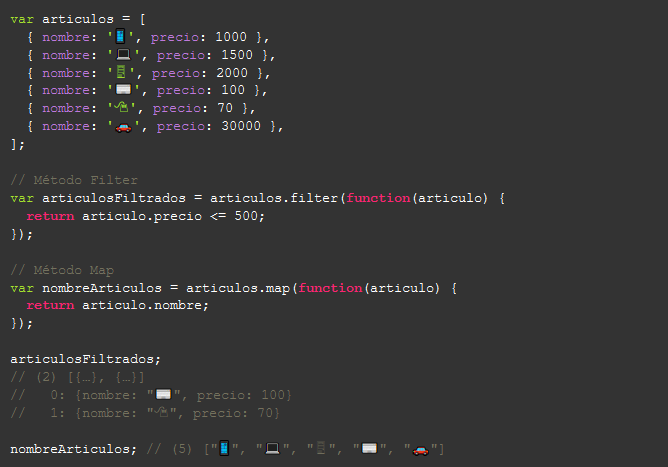


¿Qué es *this*?  
Es una variable que hace referencia al objeto. En este caso: this = miAuto.

# Objects: Función constructora



# Métodos para recorridos Arrays.



Otra forma también valida de usar el *filter* es la siguiente:



Es la misma forma, pero un poco más recortada.

# Recorriendo Arrays con .find(), .forEach() y .some()

# 

Por si a alguien le quedó alguna duda con respecto a la diferencia entre find y filter:

El método *find ()* devuelve el primer valor que coincide de la colección. Una vez que coincida con el valor en los resultados, no verificará los valores restantes en la colección de matriz.

El método *filter ()* devuelve los valores coincidentes en una matriz de la colección. Verificará todos los valores de la colección y devolverá los valores coincidentes en una matriz.

A modo de recopilación:

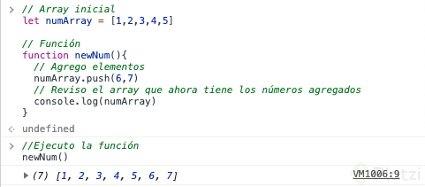
* **find()** : Devuelve el primer elemento del array que cumpla con la condición dada
* **foreach()** : Ejecuta lo que le definamos una vez por cada elemento de nuestro array
* **some()** : Comprueba si al menos un elemento del array cumple con la condición que le damos
* **filter()** : Devuelve todos los elementos del array que cumplan con la condición dada  
  Acá te dejo la documentación de cada uno: [**find()**](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/find) - [**foreach()**](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/forEach) - [**some()**](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/some) - [**filter()**](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/filter)

# Eliminando elementos de un Array.

# .push()

El método .push() nos permite agregar uno o más elementos al final de un array.

A continuación, veremos un ejemplo aplicado con un array que contiene números:



Como podemos ver, al momento de ejecutar la función se agregan los números 6 y 7 al array.

Ahora revisemos un ejemplo con strings:



Como podemos ver, agregamos dos cadenas de strings al ejecutar la función donde tenemos txtArray.push(). Es decir, indico el array al que voy a agregar elementos, uso el método .push(), y dentro de .push() indico los elementos que quiero agregar al string. Finalmente, el console.log() lo uso para revisar en la consola si esto sucedió o no.

# .shift()

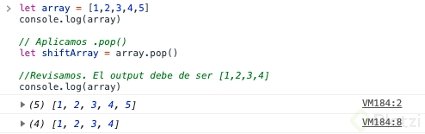
Ahora pasemos a la otra cara de la moneda donde necesitamos eliminar un elemento del array. .shift() eliminar el primer elemento de un array, es decir, elimina el elemento que esté en el índice 0.



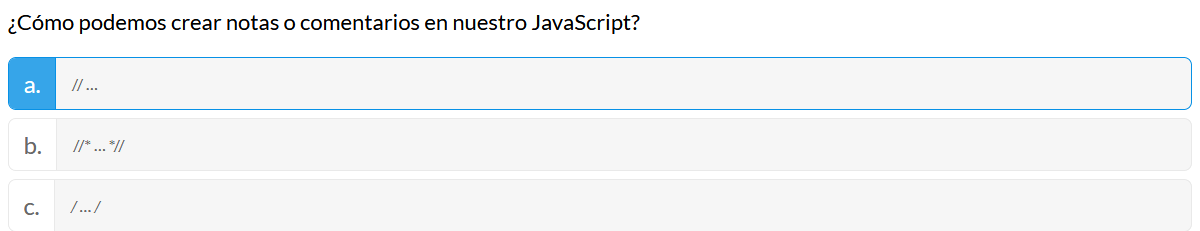
Como vemos, luego de aplicar .shift() se eliminó exitosamente el primer elemento del array. ¿Y si quisiéramos eliminar el último elemento? Pasemos al bonus track de esta clase 🙌🏼.

# Bonus Track

Si ya entendiste cómo funciona .shift() aplicar .pop() te será pan comido 🍞. El método .pop() eliminará el último elemento de un array. En este sentido, si tenemos un array de 5 elementos, pop() eliminará el elemento en el índice 4. Usemos el mismo ejemplo, pero usando este método.



***Questions and Answers***



¿Por qué decimos que JavaScript es un lenguaje dinámico?

