Índice

[**¿Qué es JavaScript?** 1](#_Toc82974079)

[**¿Realmente es Javascript un lenguaje interpretado?** 2](#_Toc82974080)

[**Diferencias entre Forwards vs Basckwards** 2](#_Toc82974081)

[**Javascript es Basckwards Compatible** 2](#_Toc82974082)

[**¿Por qué JavaScript?** 2](#_Toc82974083)

[**Tipos de datos en JavaScript** 4](#_Toc82974084)

[**Variables en JavaScript** 6](#_Toc82974085)

[**Tipos de declaración en JavaScript** 7](#_Toc82974086)

[**Funciones en JavaScript** 8](#_Toc82974087)

[**¿Qué es una función declarativa y una expresiva?** 9](#_Toc82974088)

[**Funciones Declarativas** 9](#_Toc82974089)

[**Expresión de función** 10](#_Toc82974090)

[**Diferencias** 10](#_Toc82974091)

[**Scope** 11](#_Toc82974092)

[**Hoisting** 13](#_Toc82974093)

[**Coerción** 13](#_Toc82974094)

[**Truthy y Falsy** 15](#_Toc82974095)

[**Tipos de datos raros de js** 16](#_Toc82974096)

[**Operadores: Asignación, Comparación y Aritméticos** 17](#_Toc82974097)

[**Condicionales:** 19](#_Toc82974098)

[**If, Else, else if** 19](#_Toc82974099)

[**Switch** 19](#_Toc82974100)

[**Arrays** 21](#_Toc82974101)

[**Algunos métodos de array** 21](#_Toc82974102)

[**Loops: For y For...of** 25](#_Toc82974103)

[**Loops: while** 25](#_Toc82974104)

[**Sintaxis** 26](#_Toc82974105)

[**Objects** 27](#_Toc82974106)

[**Sintaxis** 27](#_Toc82974107)

[**¿Quién es this?** 28](#_Toc82974108)

[**Importante** 28](#_Toc82974109)

[**Objects: Función constructora** 30](#_Toc82974110)

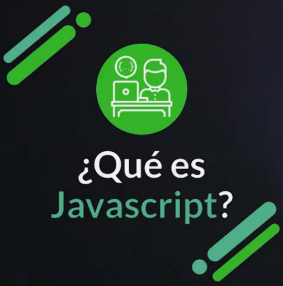
[**Eliminando elementos de un Array** 31](#_Toc82974111)

[**.push()** 31](#_Toc82974112)

[**.shift()** 32](#_Toc82974113)

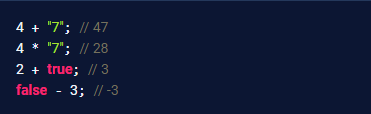
[**Bonus Track** 33](#_Toc82974114)

# **¿Qué es JavaScript?**

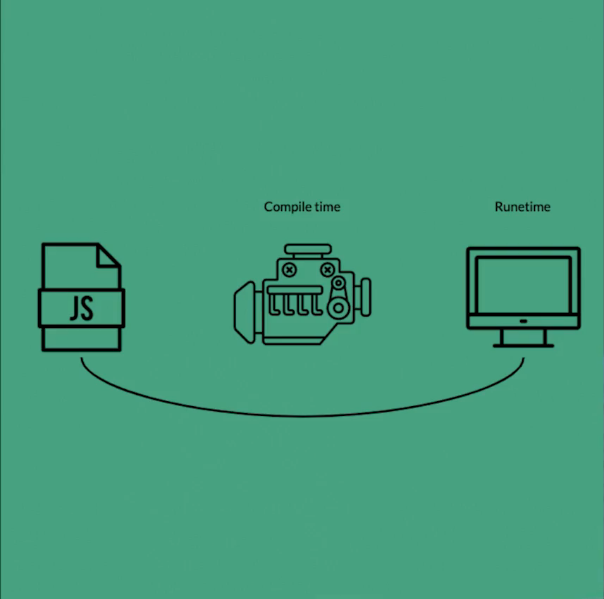
**JavaScript** nace por la necesidad de generar dinamismo en las páginas web y que a su vez personas y empresas pudieran interactuar.

Es un lenguaje interpretado, orientado a objetos, débilmente tipado y dinámico.

* **Débilmente tipado**: Se refiere a que JavaScript ejecutará operaciones entre distintos tipos de datos. Ejemplo: sumar un número con un carácter. Otros lenguajes dan un error al intentar esto.



* **Dinámico**: JavaScript no necesita ser compilado, lo que facilita probar nuestro código. También hace que los errores se vean hasta que se ejecuta dicho código, es decir, el navegador irá leyendo línea por línea el código y cuando encuentre un error, es precisamente ahí donde la ejecución se detendrá porque no puede seguir ejecutándose.



## **¿Realmente es Javascript un lenguaje interpretado?**

Si, y la razón es que le navegador lee línea por línea nuestro código el cual le indica lo que tiene que hacer, sin la necesidad de compilar. Todo esto es controlado por el motor de Javascript V8 (en Google Chrome) del navegador.



## **Diferencias entre** **Forwards vs Basckwards**

* **Forwards**: En los lenguajes de programación, si sale una nueva versión, en ese momento ya se puede utilizar en el entorno de trabajo y eso no va a romper todo el trabajo desarrollado en versiones anteriores.
* **Basckwards**: Todas las funciones nuevas o módulos que van saliendo no romperán todo el código ya hecho, pero no se puede utilizar en el entorno de trabajo en ese momento en el que sale esa nueva versión. Hay un proceso en la que la W3C debe trabajar para hacerlo estándares, recién cuando se vuelve un estándar, automáticamente se puede utilizar y no se tendrá ningún tipo de problemas.

## **Javascript es Basckwards Compatible**

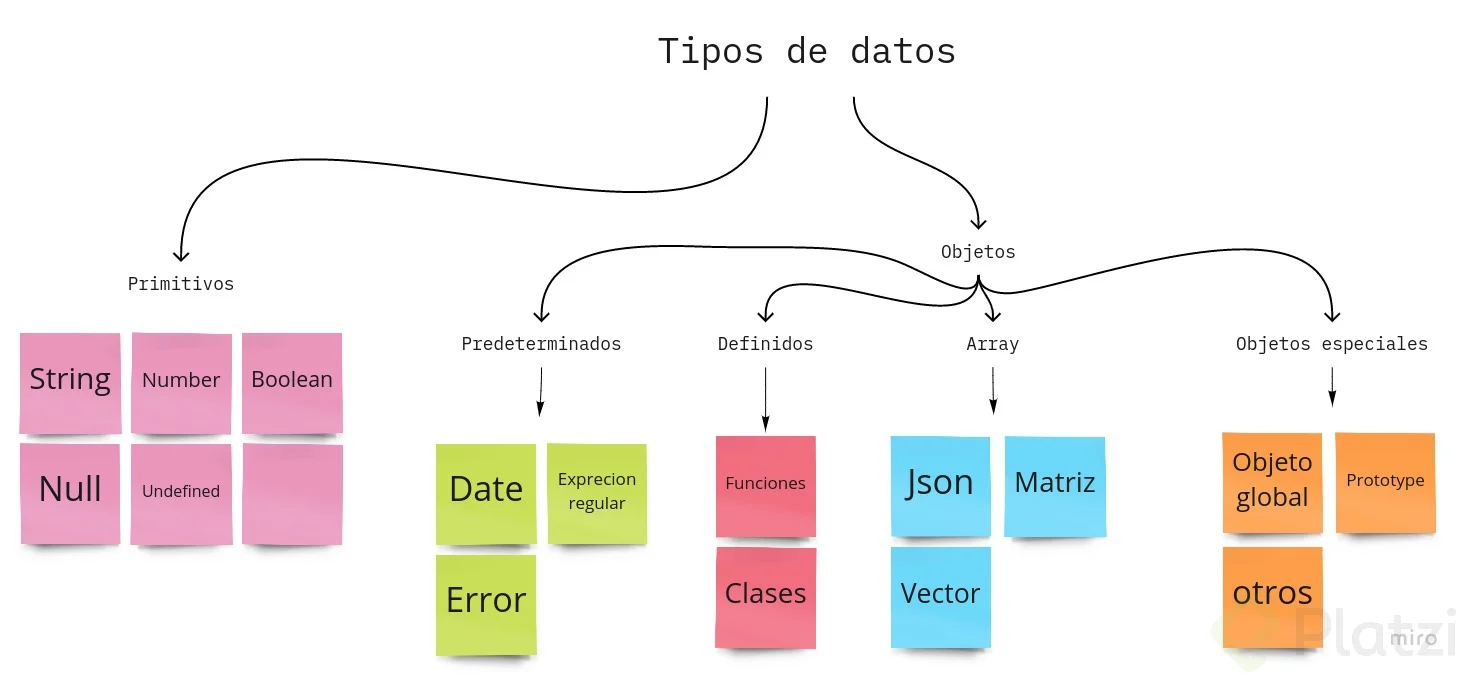
Todas las funciones nuevas que salen de Javascript no dañarán el trabajo ya hecho, pero no se podrá utilizar en nuestro entorno de trabajo inmediatamente. Para solucionar esto está **Babel** que permite utilizar las nuevas características del lenguaje, pero lo transforma a una versión que el navegador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

# **¿Por qué JavaScript?**

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

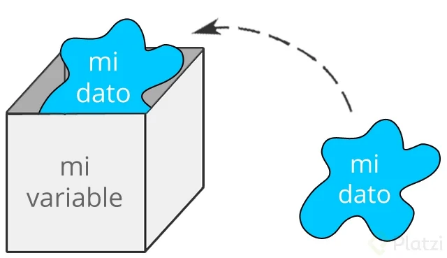
* Si solo estuvieras interesado en trabajar aplicaciones web tienes muchos frameworks y librerías construidas en JavaScript que te van a ayudar a hacer proyectos de forma mucho más rápida, eficiente y robusta (Angular, View, React, entre otros)
* Si no quieres trabajar solo en aplicaciones Web puedes utilizar JavaScript con un framework que se llama **React Native** para poder construir aplicaciones nativas como Android y IOS.
* Puedes construir aplicaciones de escritorio con JavaScript, usando un framework llamado **Electron**, pueden correr en Mac o Windows.
* 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.

# **Tipos de datos en JavaScript**



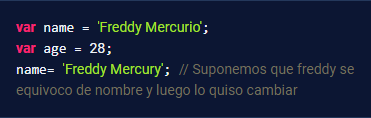


# **Variables en JavaScript**

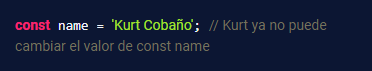
Una variable **es la representación de algún lugar en memoria que vamos a reservar para poder guardar un valor.** Ese valor puede tener diferentes tipos.

Dentro de JavaScript tenemos **tres** formas de declarar una variable las cuales son: **var**, **const** y **let**.

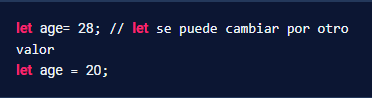
* **var**: Es una variable que puede cambiar su valor en un futuro y su alcance es local. Surgió en ECMAScript5.



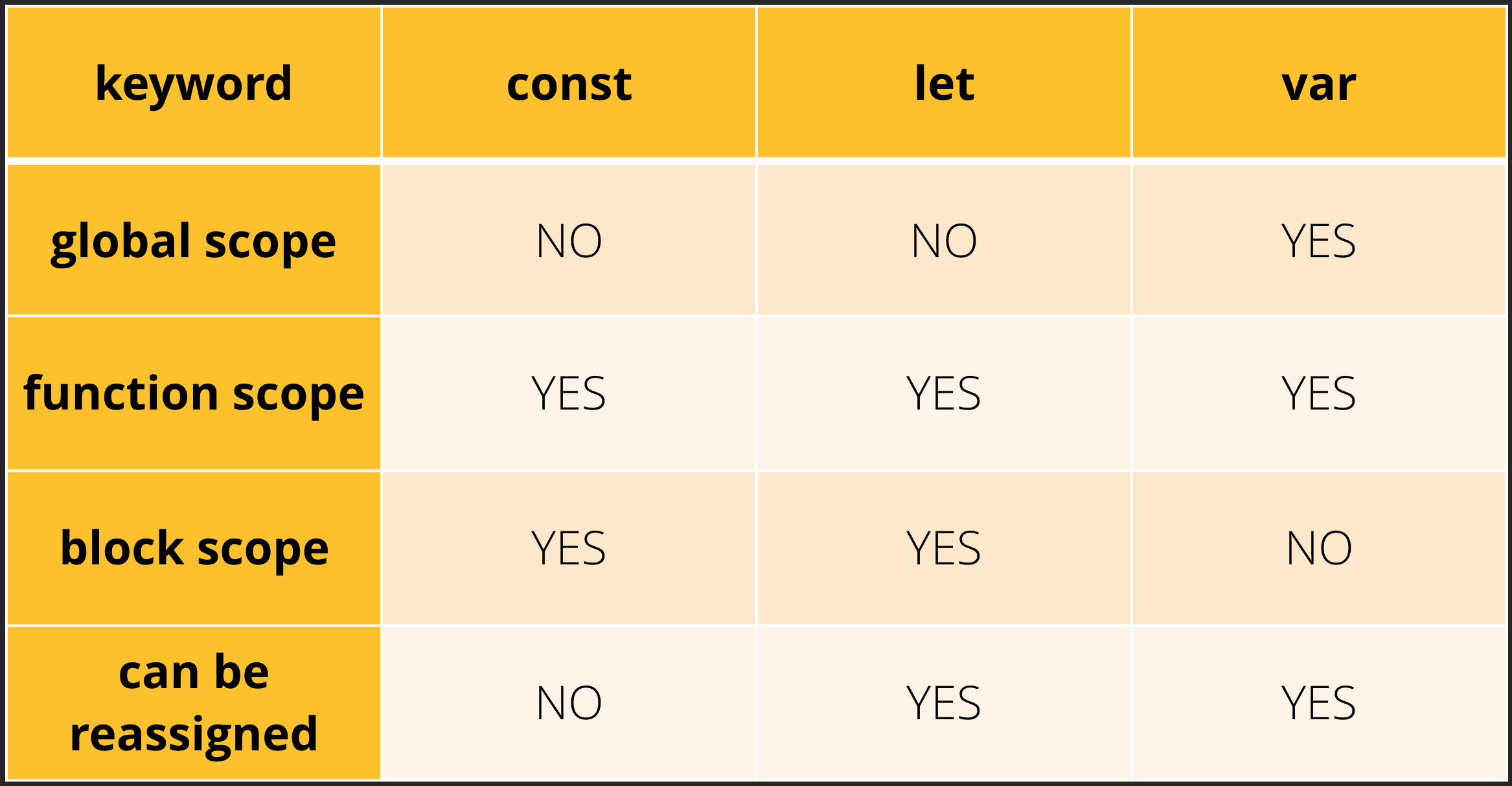
* **const**: Es una variable constante que no puede cambiar nunca su valor en un futuro.



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



## **Tipos de declaración en JavaScript**



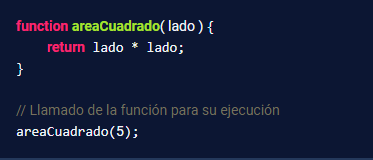
# **Funciones en JavaScript**

Una función es **un conjunto de sentencias que realizan una tarea o calculan un valor**. Para usar una función, debe definirla en algún lugar del ámbito desde el cual desea llamarla.

La declaración de una función consiste de la palabra reservada **function**, seguida por:

* El nombre de la función (opcional).
* Una lista de argumentos para la función, encerrados entre paréntesis y separados por comas.
* Las sentencias JavaScript que definen la función, encerradas por llaves.

Ejemplo de una función básica:

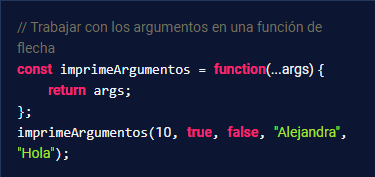


Existen diferentes tipos de funciones como lo son:

* Funciones Anónimas.
* Funciones de Flecha.

Las funciones que no tienen especificado el retorno con la palabra reservada **return**, regresarán **undefined**.

Existe un tipo de declaración en los parámetros de una función en el cual no se necesitan especificar el número de parámetros que se van a recibir, eso se hace de la siguiente manera:



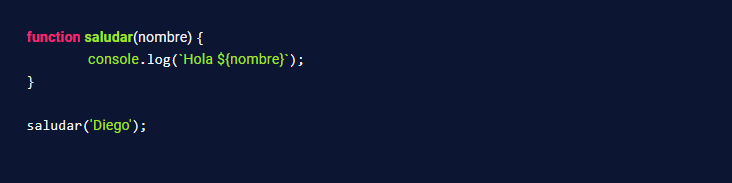
# **¿Qué es una función declarativa y una expresiva?**

Cuando hablamos de funciones en JavaScript, tenemos dos tipos de funciones:

* Funciones Declarativas (function declaration / function statement).
* 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 una **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**

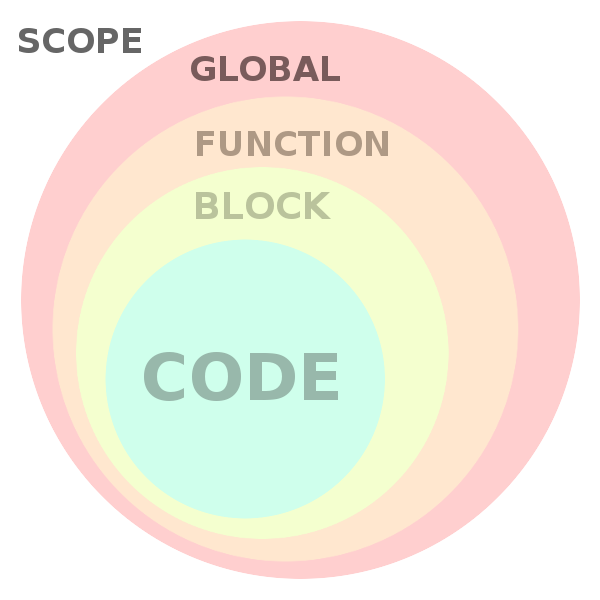
Es el alcance que tienen las variables en el código.

Existen 2 tipos de scopes.

* **Scope Global**: Todo el mundo.
* **Scope Local**: Las Vegas, y lo que pasa en las vegas, se queda en Las Vegas.

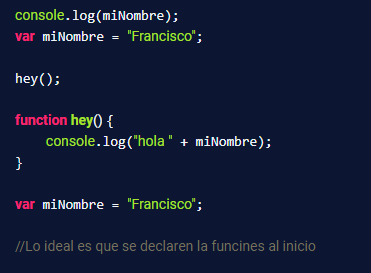


El **scope local** puede tener acceso a las variables que se encuentran en el **scope global**, pero no viceversa, es decir, las variables que se encuentran en el scope global no pueden acceder a las variables del scope local.



# **Hoisting**

Es cuando las **variables** o **funciones** son usadas antes de ser declaradas.

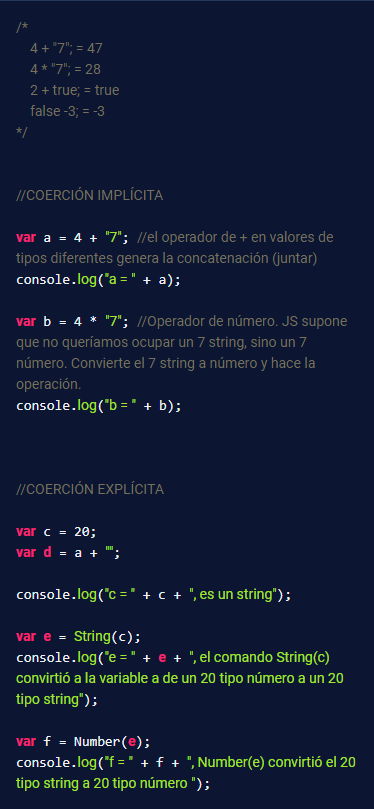


Esto sucedía antes de EcmaScript 6, desde ES6 en adelante ya no ocurre porque se crearon los keyword **const**, **let**, pero si ocurre con las funciones.

# **Coerción**

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

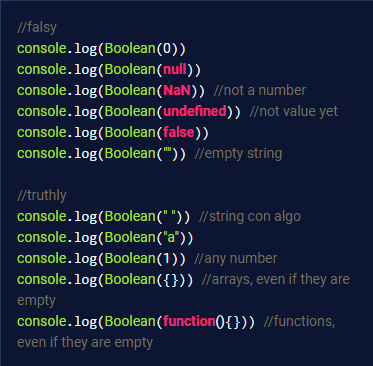
* **Coerción implícita**: Es cuando el lenguaje nos ayuda a cambiar el tipo de valor.
* **Coerción explicita**: Es cuando obligamos a que cambie el tipo de valor.



# **Truthy y Falsy**

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

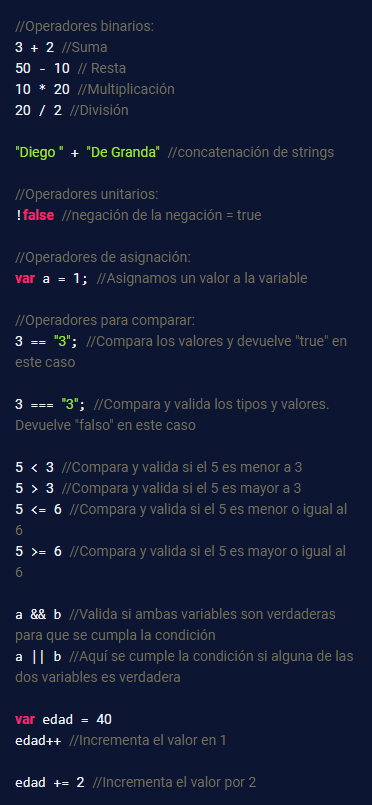
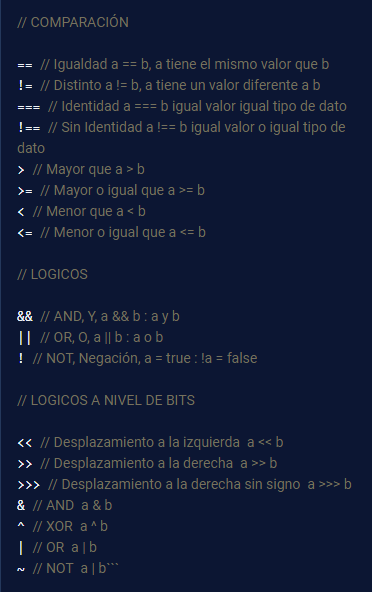
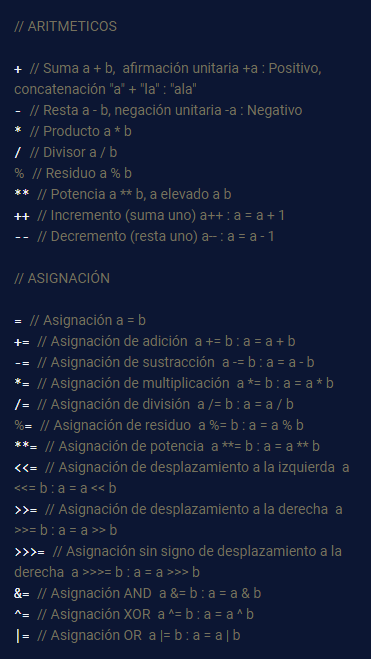
* **Falsy**
  + Boolean() —> Sin ningún valor es **false**.
  + Boolean(0) —> **false**.
  + Boolean(null) —> **false**.
  + Boolean(NaN) —> **false** // **NaN** => Not and Number.
  + Boolean(Undefined) —> **false**.
  + Boolean(false) —> **false**.
  + Boolean("") —> **false**.
* **Truthy**
  + Boolean(1) —> **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**.



## **Tipos de datos raros de js**

* **NaN**: En JS significa **Not a Number** y quiere decir que esa operación necesita de dos números o más para poder operarse.
* **undefined**: Undefined es un valor especial de JavaScript que existe cuando no se le asigna un valor a una variable. Es decir, la variable si existe o ha sido creada pero no se le ha asignado ningún valor. (Su valor es indefinido).
* **null**: null es cuando algo no existe.

# **Operadores: Asignación, Comparación y Aritméticos**



# **Condicionales:**

# **If, Else, else if**

Los condicionales son las reglas que vamos a aplicar para poder validar si algo es verdadero o falso.

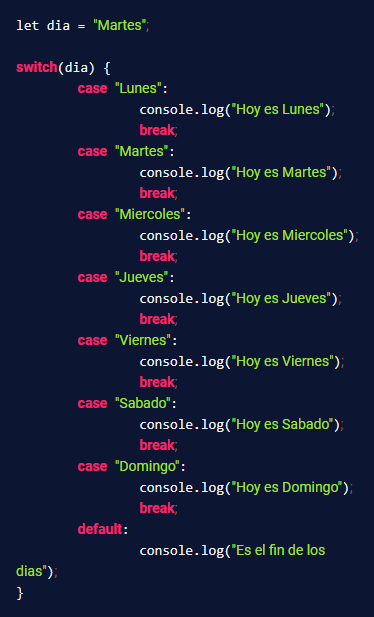
Esta expresión se le como **sí** (se cumple la **condición**) entonces dame el resultado. **Sino**, dame el otro **resultado**.

# **Switch**

Un **switch** es como un **if** pero diferente en su sintaxis, pero son muy similares.

¿Como usar switch?

La sintaxis de switch es:



El uso de **break** es importante, ya que si no lo pones accionara todas las opciones dentro del **switch**.

En el caso de que ninguna de las opciones dentro del switch sea la correcta tenemos 2 opciones por hacer:

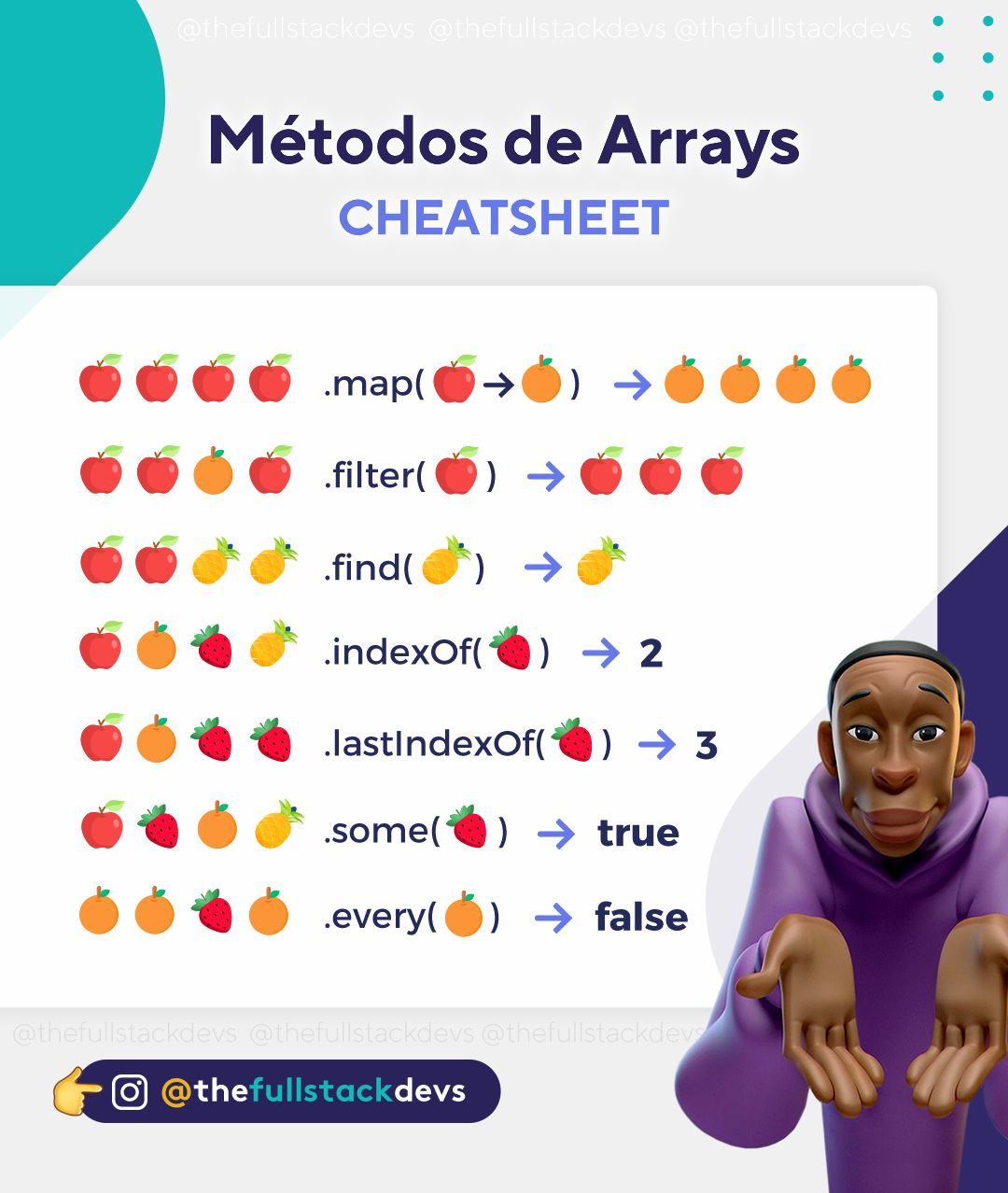
* Dejar el código seguir sin hacer nada.
* Utilizar **default** que es como si utilizáramos un **else** pero para switch.

# **Arrays**

**Es un tipo de estructura de datos, objeto**. Puede guardar datos distintos dentro, guarda los datos en forma de lista.

## **Algunos métodos de array**

* **lenght**: Devuelve la longitud del array.
* **push()**: Agrega elementos al final de array.
* **pop()**: Elimina un elemento del array.
* **unshift()**: Agrega un elemento al array, pero lo agrega en primer lugar.
* **shift()**: Elimina el elemento que está en el inicio del array.
* **indexof**: Devuelve la posición de un elemento del array.



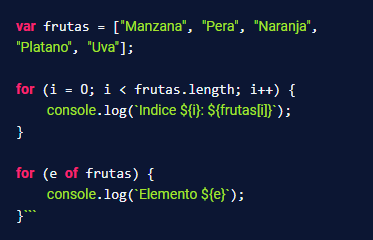




# **Loops: For y For...of**

Los bucles pueden ejecutar un bloque de código varias veces. JavaScript admite diferentes tipos de bucles:

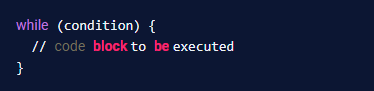
* **for** - recorre un **bloque** de código varias veces.
* **for/in** - recorre las **propiedades** de un **objeto**.
* **for/of** - recorre los **valores** de un objeto **iterable**.
* **while** - recorre un bloque de código **mientras** se cumple una condición específica.
* **do/while** - también recorre un bloque de código **mientras** se cumple una condición específica.



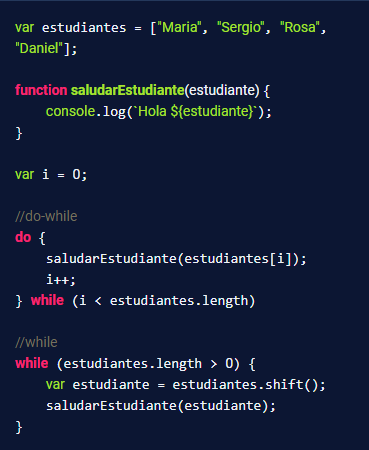
# **Loops: while**

**While** recorre un bloque de código **mientras** se cumpla una condición específica.

## **Sintaxis**



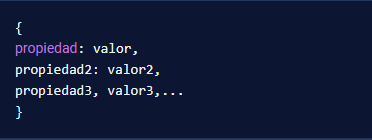
Ejemplo



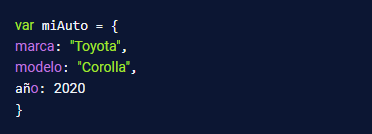
# **Objects**

Un **objeto es una estructura de datos** (al igual que los array), con la diferencia que en los objetos **no existe el índice numérico**, lo que hay son **propiedades**, con ellas accedemos a cada uno de los **valores**.

## **Sintaxis**



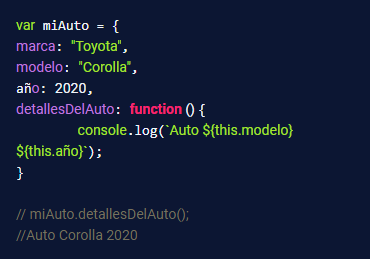
Ejemplo de Objeto:



Acceder a una propiedad del objeto.



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



## **¿Quién es this?**

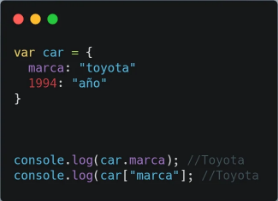
Es una variable que hace referencia al objeto. En este caso: **this = miAuto**.

## **Importante**

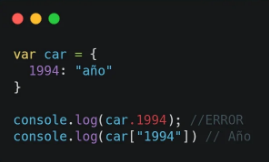
* Los **key values** se representan como **string** para JavaScript esto quiere decir que, si se coloca una propiedad, por ejemplo:

**marca : “toyota”** es lo mismo decir **“marca” : “Toyota”**

* Debido a que los **key values** Javascript los interpreta como **strings** significa que también **pueden acceder a una propiedad usando llaves** y dentro especificando el **key value**.



* Esto es importante porque si se coloca un **key value** tipo **número**, nose puede accedera esta propiedad de la forma tradicional.

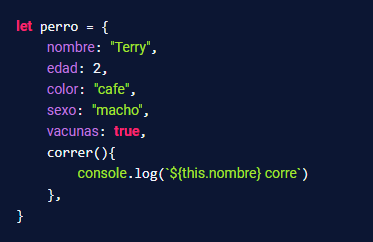


Los valores del objeto pueden ser de todo tipo, es decir, que podemos tener:

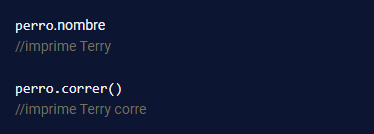
* **Number**.
* **String**.
* **Función**.
* **Array**.
* **Incluso** otro objeto que contenga **otros** valores.

Los objetos son importantes porque de algún modo imitan el comportamiento en el mundo real, los objetos tienen **propiedades** y tienen **métodos**.

* **Propiedades**:
  + Es una característica.
* **Método**:
  + Algo que el objeto puede hacer.

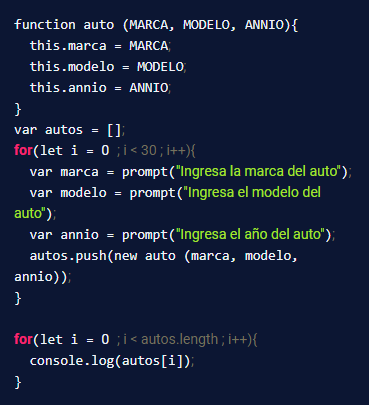


Para acceder a los **valores** y **métodos** hacemos.



# **Objects: Función constructora**

Nos ayuda a construir un objeto de forma dinámica. Esta función es posible crear una nueva instancia de sí misma.



# **Eliminando elementos de un Array**

## **.push()**

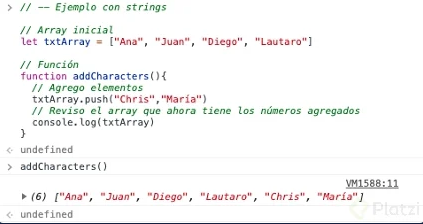
El método .push() nos permite agregar uno o más elementos al final de un array.

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.

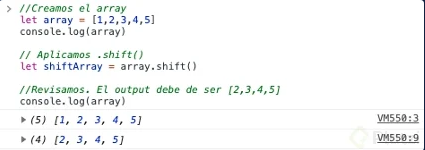
Ahora revisemos un ejemplo con strings:



Como podemos ver, agregamos dos cadenas de strings al ejecutar la función donde tenemos **txtArray.push()**. Es decir, indico el array al que voy 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?

## **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.