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 Od Things) es un concepto que se refiere a una interconexion 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.

- FRONT: Tiene opciones como Angular, React y VUE

- BACK: Tiene NodeJS

-MOVIL: Tiene React Native (Aplicaciones de Android y IOS)

- WEB DESKTOP: Tiene Electron

¿Qué es un framework?

Un framework está formado por conceptos, artefactos, o prácticas que ayudan a organizar los recursos que requiere el programador para hacer su trabajo. Un framework sirve tanto para escribir código fuente como para desarrollar toda una aplicación o software. Permite almacenar y organizar recursos esenciales para el desarrollo como el código fuente, los ficheros de configuración o las librerías de la aplicación.

De esta forma, el programador tiene todos los recursos a la vista y en un solo lugar y el entorno de trabajo se vuelve una estructura que hay que completar, en vez de partir de cero.

¿Qué es una librería?

Una librería es un conjunto de implementaciones funcionales, un pedazo de código, y funcionalidad que se deben invocada.

A diferencia de un programa ejecutable, una biblioteca no espera ser utilizada de forma autónoma (un programa sí: tiene un punto de entrada principal), sino que su fin es ser utilizada por otros programas, independientes y de forma simultánea.

VALORES

PRIMITIVOS: son los valores básicos que se utilizan para poder generar códigos.

Número: data sólo de tipo numérico, ejem: 1 2 3 4, etc.

Strig: texto, se utiliza entre comillas dobles. “Haciendo esto”, “Diego Hernández”

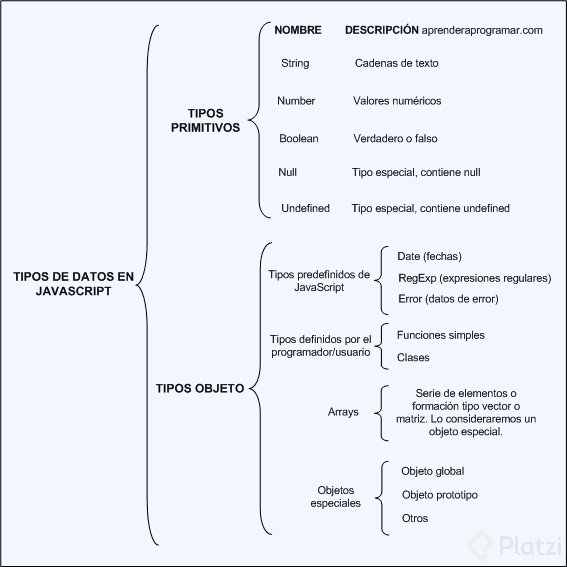
Boolean o Booleano: son valores creados por George Boolean dentro de lenguajes de programación para poder validar cosas y/o desiciones, por ello se utilizan: true (1) o false (0).

Empty values: null o undefined, se les conoce como valores “placeholders”, pero son valores reservados para un tipo de valor que queda faltante en memoria. También pueden ser tomados como valores de errores.

NO PRIMITIVOS O VALORES TIPO OBJETOS:

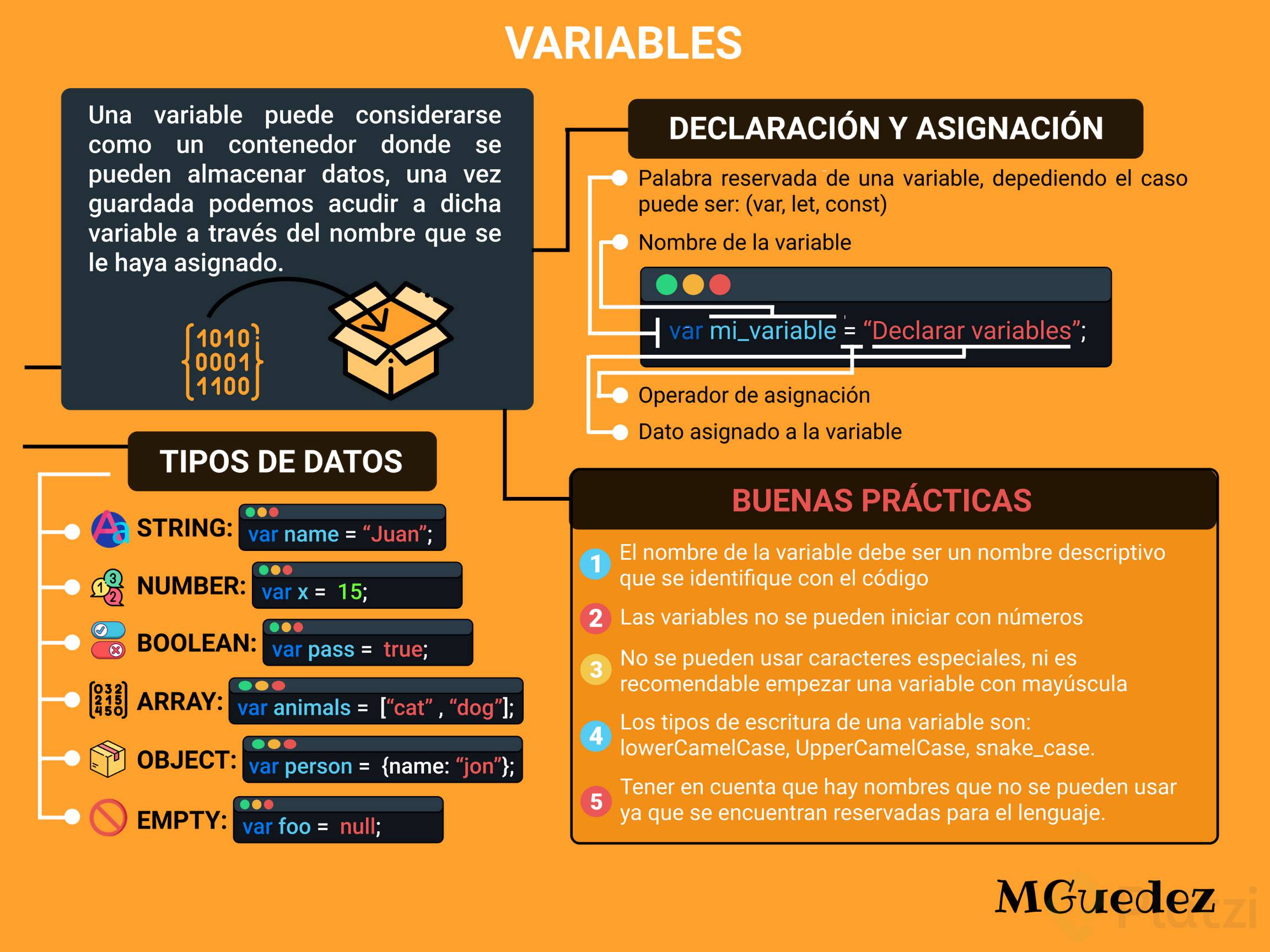
Array: se genera ocupando corchetes, los cuales dentro ocupan valores primitivos, ejem: [1,2,3] convirtiéndolos en valores tipo objeto.

Valor tipo Objeto: se generan con corchetes, los cuales dentro ocupan una data que se transforma en objeto, ejem de sintaxis: { nombre: “Diego”}.



VARIABLES (LET Y CONST)

¿Qué es una variable?



const: sirve para declarar variables que nunca van a ser modificadas:

No se puede reinicilizar: es una const única no puede haber otra inicializada con el mismo nombre. const pokemonType = ‘electric’ no puede haber:

const pokemonType = ‘grass’.

No se pude re asignar: una vez que la hayamos inicializado no la podemos reasignar solo con su nombre: const pokemonType = ‘electric’ no puede ejecutarse:

pokemonType = ‘grass’.

No es inmutable: osea no puede cambiar con objetos:

Let: Son variables que pueden ser modificadas, se pueden cambiar:

No se puede reinicilizar: es una const única no puede haber otra inicializada con el mismo nombre. let pokemonType = ‘electric’ no puede haber:

let pokemonType = ‘grass’.

Se puede reasignar: Osea la variable ya inicializada le reasignamos otro valor por ejemplo: inicializamos la variable: let pokemonType = ‘electric’ ahora la reasignamos pokemonType = ‘grass’.

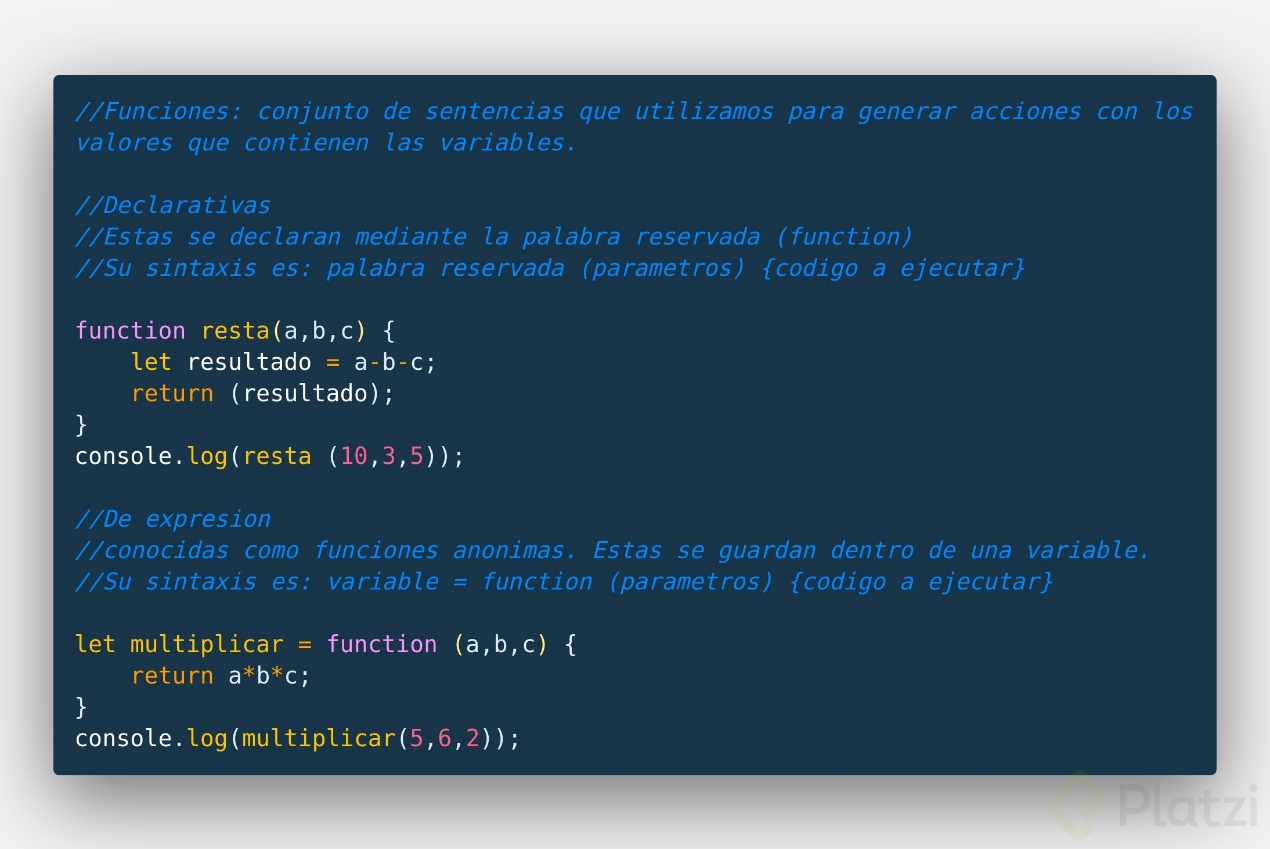
Su contexto es de bloque: Solo funciona dentro de un bloque {}, fuera de ello no.

FUNCIONES

**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 let, donde se generará una variable que guardará una función anónima.





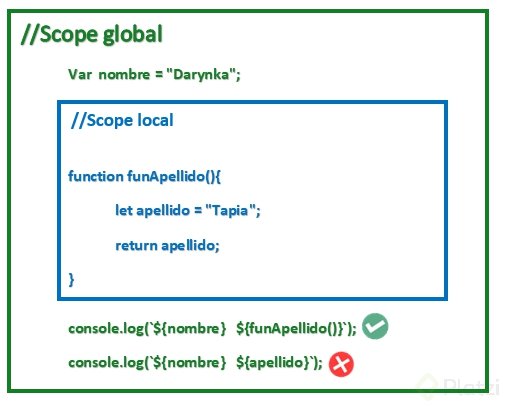
SCOPES

Scope local:

Son aquellas variables definidas dentro del cuerpo de la función, estas son solo accedidas desde dentro de la misma función.

Scope Global: variables que pueden ser accedidas y procesadas por cualquier función dentro del código.

*variables globales:* Pueden ser accedidas desde un scope local o global. las variables globales son definidas fuera de las funciones (Scope global).



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 asignó un valor hasta después de su ejecución.

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 asignó el valor.

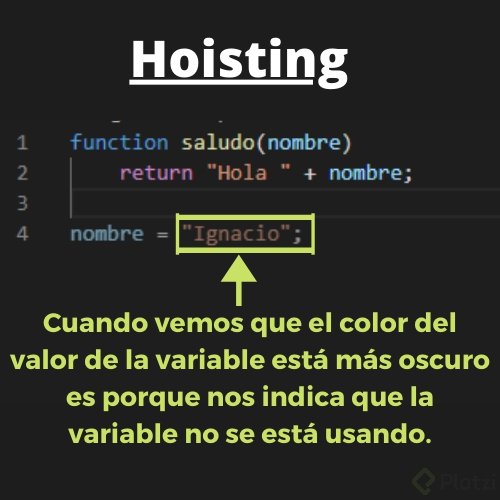
Este comportamiento se puede entender fácilmente si se comprenden estos dos puntos esenciales:

* 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.

En base al segundo punto, fue por eso que se cometió el error de usar la variable antes de inicializarla, pues sin problema el compilador le asigna memoria, pero no el valor hasta después.

***Es por eso que se tiene como buena práctica declarar e inicializar tanto variables como funciones al inicio de nuestro programa, sin importar donde sean utilizadas, pues de esta manera se evita usarlas antes de ser inicializadas.***

Debemos saber que el hoisting solo sucede con las palabras claves **var** y **function**, por lo tanto, esto nos dice que solo se da en las versiones de ECMAScript 5 o Inferiores, ya que en la versión 6 y superiores se permite la declaración de variables con **let** y **const**, que son dos nuevas variables que no activan hoisting.



COERCION

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.

Boléanos (TRUST & FALSE)

**¿Qué tipo de valor es verdadero y faso?**

**Que tipos por default son verdaderos y falsos**

**La siguiente lista daremos ejemplos**

**usamos la funcion de JS que es Boolean() dentro del pàrentesis ponemos el valor**

**Boolean() —> sin ningun 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**

**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 tambien es true.**

**Boolean(true) —> true.**

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

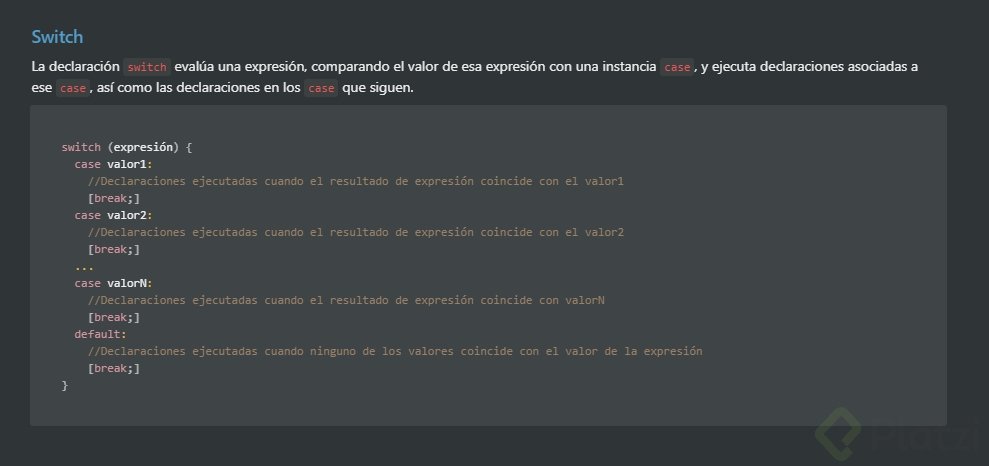
Operadores: Asignación, Comparación, Aritméticos y Lógicos.



CONDICIONALES (IF, ELSE, ELSE IF)



SWITCH



ARRAYS

Un **Array** es un tipo de estructura de datos, objeto. Puede guardar datos distintos dentro, guarda los datos en forma de lista.  
**.lenght** devuelve la longitud del array.  
**.push()** agrega elementos al final de array.  
**.pop()** elimina el último elemento del array no es necesario pasarle un parámetro.  
**.unshift()** agrega un elemento al array, pero lo agrega en primer lugar.  
**.shift()** elimina el primero , no es necesario pasarle un parámetro.  
**.indexOf** devuelve la posición de un elemento del array.

para el caso de arrays y objetos, un console.table() luce mejor que un clásico console.log()



MAS EJEMPLOS:

<https://dev.to/leirasanchez/like-console-log-but-better-nhm?utm_source=Newsletter+Subscribers&utm_campaign=1f3aa837d6-EMAIL_CAMPAIGN_2020_03_02_05_11&utm_medium=email&utm_term=0_d8f11d5d1e-1f3aa837d6-155532545>

METODOS para recorrer ARRAYS

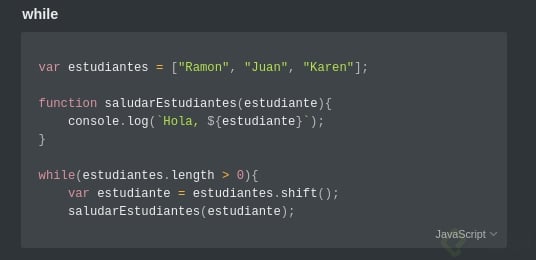
* **find()** : Devuelve el primer elemento del array que cumpla con la condición dada.
* **foreach()** : Ejecuta lo que le definamos una vez por cada elemento de nuestro array.
* **some()** : Comprueba si al menos un elemento del array cumple con la condición que le damos. Y devuelve true o false.
* **filter()** : Devuelve todos los elementos del array que cumplan con la condición dada.

VER MAS METODOS: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array>

LOOPS

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. EJ:

var estudiantes = ["Maria", "Sergio", "Rosa", "Daniel"];

**function** **saludarEstudiante**(estudiante) {

console.log(`Hola ${estudiante}`); }

var i = 0;

**do** { saludarEstudiante(estudiantes[i]); i++; }

**while** (i < estudiantes.length)

IMPRESIÓN POR CONSOLA

//Concatenando:

console.log("Hola, "  + estudiante + " estoy usando concatenación");

//Template Literals

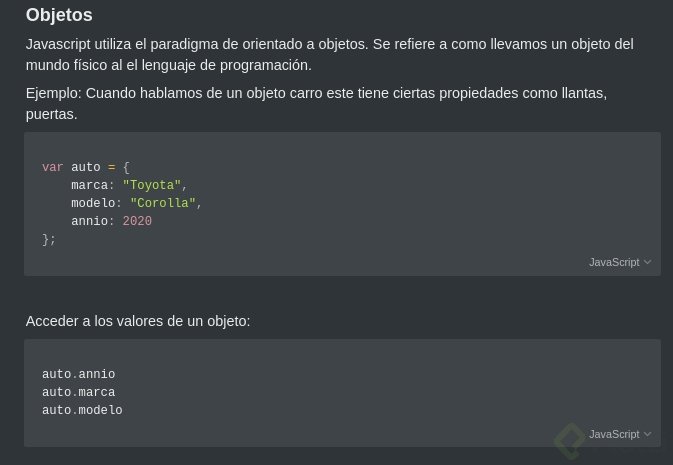
console.log(`Hola, ${estudiante} usando template literals`)

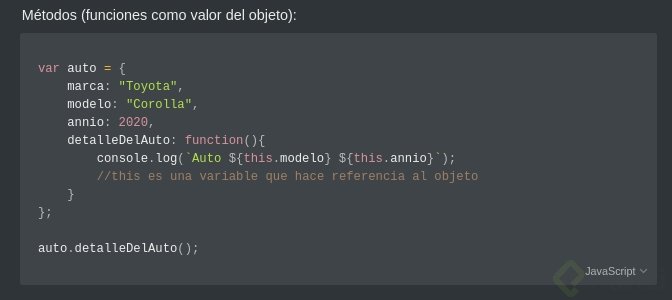
OBJETOS LITERALES

Un objeto en programación es una representación abstracta de un objeto en la vida real, sin embargo también puede entenderse como un contenedor de datos.  
Ejemplo:  
Una botella de refresco tiene ciertas características y acciones.-  
Características: altura, volumen, color, contenido, dureza etc…  
Acciones: beber, reciclar, rehusar, aventar, abrir tapa, etc…

Respectivamente a estas características y acciones se les conoce como atributos(características) y métodos(acciones) en programación.

Los objetos son usados para almacenar ciertos datos que de otra manera serían muy sensibles al hackeo, por otro lado, un objeto también puede ser el valor de una variable.





FUNCION CONSTRUCTORA

