

1.- 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 mas 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.

Sitios web basados en:

1. Angular: [Forbes](https://www.forbes.com/)
2. React: [Airbnb](https://www.airbnb.com/)
3. Vue: [GitLab](https://about.gitlab.com/" \t "_blank)

Aplicaciones basadas en React Native:

1. [UberEats](https://www.ubereats.com/)
2. [Discord](https://discord.com/)
3. [Instagram](https://www.instagram.com/)  
   Fuente: [Enlace](https://reactnative.dev/showcase)

Aplicaciones para Escritorio basados en Electron:

1. Visual Studio Code
2. WhatsApp
3. Twitch  
   [Ver más aplicaciones de Electron JS](https://www.electronjs.org/apps)

Compañías que usan Node.JS para parte de su backend:

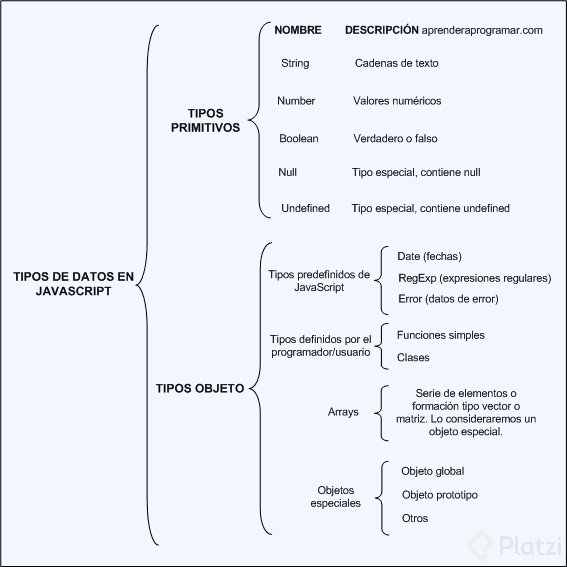
1. Netflix
2. Linkedin
3. PayPal  
   Fuente: [Enlace](https://blog.techmagic.co/companies-that-use-node-js-for-backend-how-do-big-players-benefit-from-it/)

- FRONT: Tiene opciones como Angular, React y VUE

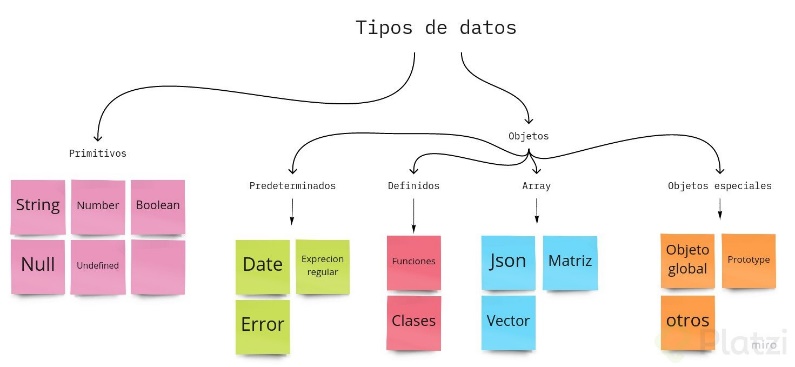
- BACK: Tiene NodeJS

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

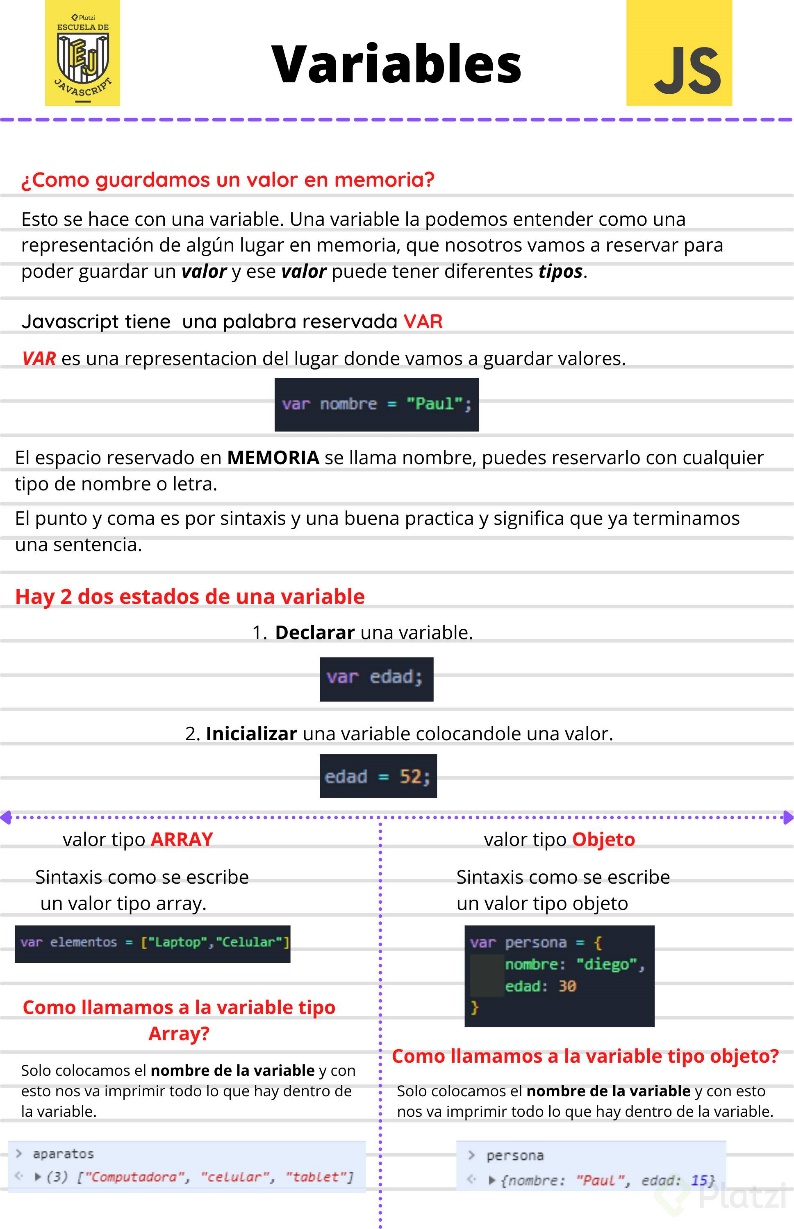
- WEB DESKTOP: Tiene Electron



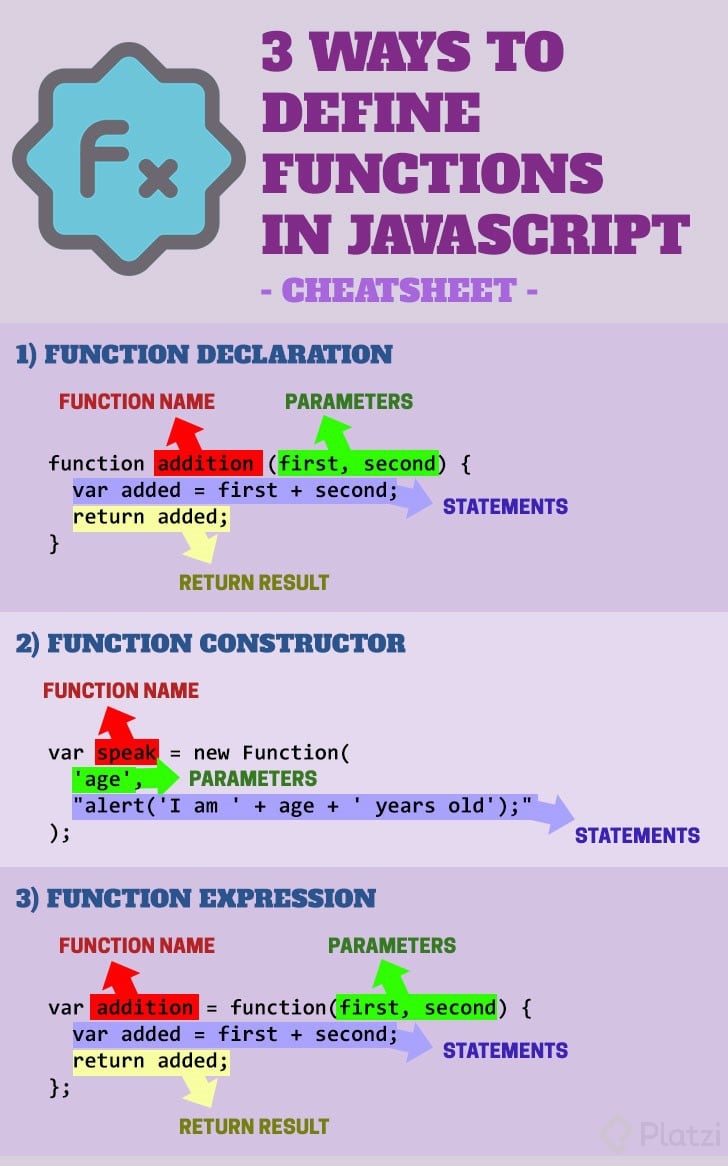


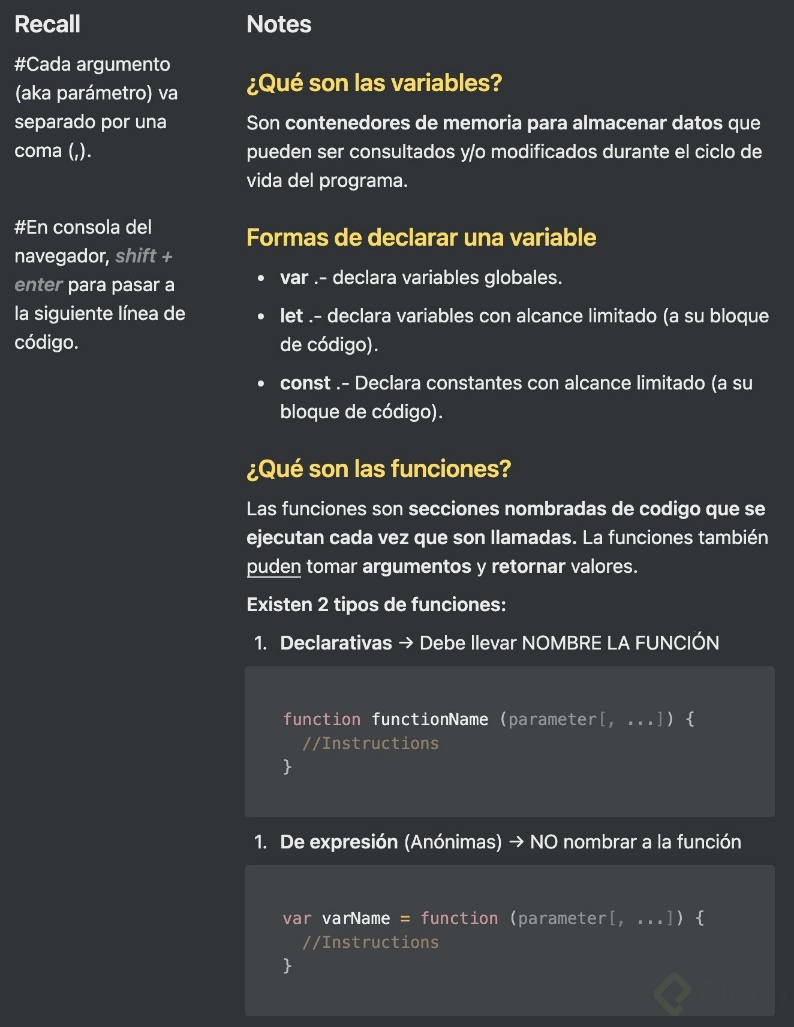












¿Qué es una función declarativa y una expresiva?

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:

**function** **saludar**(nombre) { console.log(`Hola ${nombre}`); } saludar('Diego');

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.

**var** nombre = **function**(nombre){ console.log(`Hola ${nombre}`) } nombre(‘Diego’);

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.

function solution(valor) {

typeof (valor);

return typeof valor

}

solution(5);

function soluction (valor) {

return typeof valor;

}

solution(5);

**Scope**

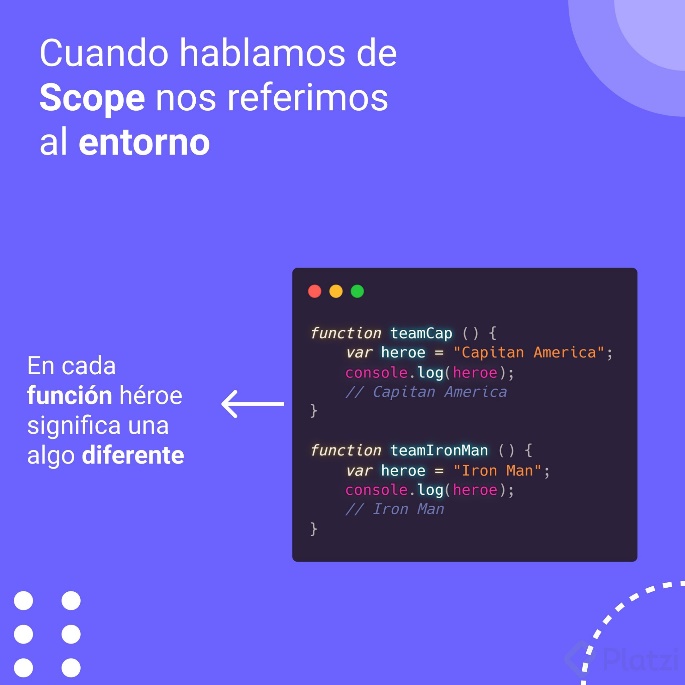
Scope Global : todo el mundo.

Scope Local : Las Vegas, y lo que pasa en las vegas, se queda en Las Vegas.







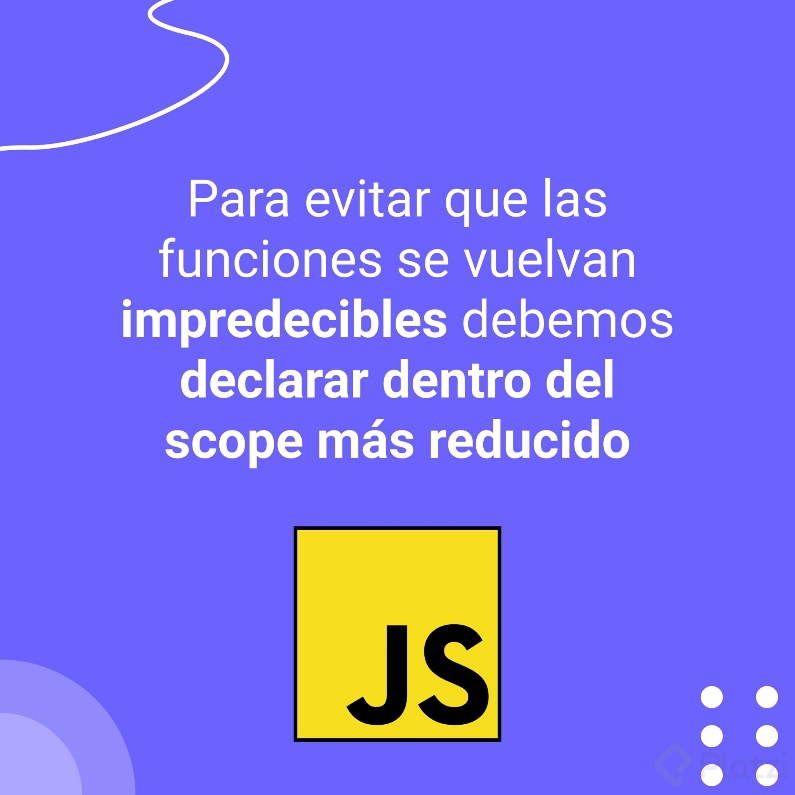










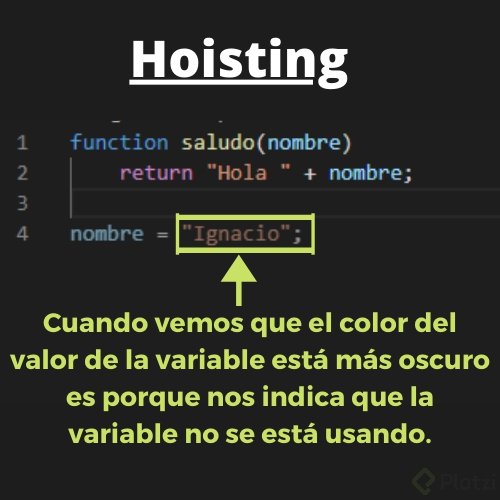


**Hoisting**

En JavaScript, las declaraciones (por ejemplo, de variables o funciones) se mueven al principio de su scope o ámbito. Este comportamiento se conoce como hoisting y es muy importante tenerlo en cuenta a la hora de programar para prevenir posibles errores.

Las funciones siempre se mueven arriba del scope. Por lo tanto, podemos elegir donde declararlas y usarlas.

La declaración de las variables se mueven arriba del scope, pero no la asignación. Antes de usar una variable, habrá que crearla y asignarla.



**Qué son los valores falsy**

Un valor falsy es aquel que es falso en un contexto booleano, estos son: 0, "" (string vacío), false, NaN, undefined o null, (!).

// Coersión explícita

Boolean(0) // false

Boolean("") // false

Boolean(null) // false

Boolean(undefined) // false

Boolean(NaN) // false

Boolean(false) // false

// Coersión implícita (**no la uses**)

!!0 // false

!!"" // false

!!null // false

!!undefined // false

!!NaN // false

!!false // false

**Qué son los valores truthy**

Un valor truthy es aquel que es verdadero en un contexto booleano, son todos los valores que no sean falsy, que especificamos en la anterior sección.

// Coersión explícita

Boolean(12) // true

Boolean("hola") // true

Boolean(true) // true

Boolean([1, 2, 3]) // true Boolean(function hola() {}) // true Boolean({ a: 1, b: 2 }) // true

Cabe recalcar que en JavaScript todo valor que no sea falsy es truthy incluyendo las estructuras vacías de array y objetos.

Boolean([]) // true

Boolean({}) // true

**Qué son los operadores aritméticos**

Los operadores aritméticos se utilizan para efectuar operaciones matemáticas. Para realizar las operaciones básicas, como suma, resta, multiplicación y división; utiliza los siguientes operadores:

// Suma

2 + 3 // 5

// Resta

5 - 3 // 2

// Multiplicación

4 \* 2 // 8

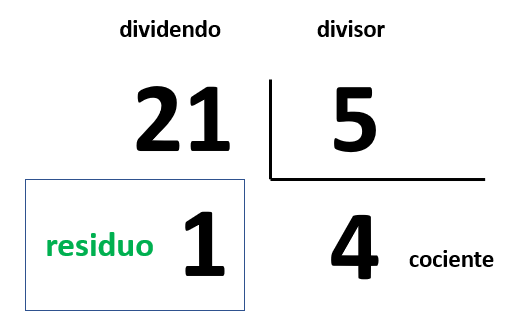
// División

6 / 2 // 3

Recuerda que no existe la división entre 0. En ese caso JavaScript devolverá el valor Infinity.

**Qué es el operador de residuo**

El operador de residuo ( % ), el signo de porcentaje, devuelve el residuo de una división.



//Residuo

21 % 5 // 1

Esto es importante para saber los múltiplos de cualquier número o si un número es par.

**Qué es el operador de concatenación**

El operador de concatenación consiste en unir dos o más strings.

"Hola" + "Platzi" // "Hola Platzi"

**Cómo utilizar el operador de incremento y decremento**

El operador de incremento (++) y decremento (--) consiste en aumentar o disminuir una unidad a una variable, respectivamente. Estos operadores se pueden emplear antes y después de la variable.

var a = 3

var b = 3

console.log(a++) //3

console.log(++b) //4

console.log(a) //4

console.log(b) //4

**Qué son los operadores de asignación**

En la clase de variables aprendiste un operador de asignación (=). Este operador es diferente a los operadores de igualdad (== y ===).

El operador de asignación (=) consiste en asignar un valor a una variable.

var saludo = "Hola Mundo"

**Operadores de asignación combinada**

En ciertos casos, reasignarás la misma variable más otro valor. Estas variables pueden utilizarse como acumuladores o contadores.

var contador = 1

contador = contador + 1

contador = contador + 1

console.log(contador) // 3

Una forma para evitar estar repitiendo la variable en la reasignación, es combinarlas con los operadores aritméticos antes del operador de asignación.



var contador = 1

contador += 2

contador -= 1

contador \*= 5

contador /= 2

console.log(contador)

**Qué son los operadores de comparación**

Un operador de comparación compara dos o más valores y devuelve un valor lógico (verdadero o falso).

**Qué son los operadores de igualdad**

Existen dos tipos de igualdad:

* Igualdad por valor (==): compara dos variables solamente por su valor. Por ejemplo: "3" de tipo string y 3 de tipo número son iguales.
* Igualdad por valor y tipo de dato (===): compara dos variables por su valor y tipo de dato. Por ejemplo: "3" de tipo string y 3 de tipo número no son iguales. Solamente 3 y 3, ambos de tipo número son iguales.

//Igualdad

"3" == 3 // true

3 == 3 // true

// Igualdad estricta

"3" === 3 // false

3 === 3 // true

**Qué son los operadores de desigualdad**

Igualmente que los operadores de igualdad, existen dos tipos:

* Desigualdad por valor (!=)
* Desigualdad por valor y tipo de dato (!==)

//Desigualdad

"3" != 3 // false

3 != 3 // false

// Desigualdad estricta

"3" !== 3 // true

3 !== 3 // false

**Qué son los operadores de mayor o menor**

Los operadores de mayor o menor evalúan intervalos, dependiendo si el valor especificado está incluido o no incluido.

// Menor que

3 < 5 // true

// Mayor que

3 > 5 // false

// Mayor o igual a que

3 >= 3 // true

3 >= 5 // false

// Menor o igual a que

3 <= 3 // true

3 <= 5 // true

**Qué son los operadores lógicos**

Los operadores lógicos comparan dos o más expresiones y devuelve un valor lógico (verdadero o falso). Las expresiones son comparaciones entre valores, se utiliza en conjunto con los operadores de comparación.

**Qué son los operadores lógicos**

Los operadores lógicos comparan dos o más expresiones y devuelve un valor lógico (verdadero o falso). Las expresiones son comparaciones entre valores, se utiliza en conjunto con los operadores de comparación.

**Qué es el operador disyunción lógico**

El operador de disyunción o AND (&&) devuelve verdadero, si y solo si ambas expresiones son verdadero. Se lee de la siguiente manera: “La expresión 1 es verdadero Y la expresión 2 es verdadero, entonces es verdadero”.

var a = 15

var b = 5

(a >= 10) && (a <= 20) // true

(b >= 10) && (b <= 20) // false

**Qué es el operador unión lógico**

El operador de unión u OR (||) devuelve verdadero, si y solo si, alguna expresión es verdadero. Se lee de la siguiente manera: “La expresión 1 es verdadero O la expresión 2 es verdadero, entonces es verdadero”.

var a = 15

var b = 5

(a <= 10) || (a >= 20) // false

(b <= 10) || (b >= 20) // true

**Qué es el operador negación lógico**

El operador de negación o NOT (!) devuelve el valor lógico contrario a la expresión. Se lee de la siguiente manera: “La expresión es verdadero, entonces es falso”.

var a = 5

!(a < 0) // true

**TAREA**

function solution(secreto) {

if (secreto > 5) {

return true

}

else {

return false

}

}

Solution(2);

function solution(secreto) {

return !(secreto <= 5);

}

Solution(2);

**Condicionales: If, Else, else if**

Los condicionales son estructuras de control que te permiten evaluar diferentes expresiones y realizar determinadas acciones en JavasScript.

**Cómo utilizar el condicional ifen JavaScript**

Un condicional evalúa si una expresión o condición es verdadera. Por ejemplo, si mi edad es mayor o igual que 18, puedo conducir

if (edad >= 18){

console.log("Puedes conducir")

}

La palabra reservada else evalúa cuando la expresión del if es falsa, pero no es obligatorio colocarlo. En el ejemplo anterior, la condición contraria del if es la edad menor que 18, entonces no puedes conducir.

if (edad >= 18){

console.log("Puedes conducir")

} else {

console.log("No puedes conducir")

}

**Cómo anidar condicionales al programar**

if (condicion1){

// Bloque 1

} else if (condicion2){

// Bloque 2

} else if (condicion3){

// Bloque 3

} else {

// Bloque else

}

Por ejemplo, un cliente solicita un descuento según el número de artículos que ha comprado, la tienda ofrece 3 descuentos: 10% si ha comprado más de 5 artículos, 15% si son más de 10 artículos, y todo por encima de 15 artículos recibe un 20% de descuento.

var precio = 10

if (articulos >= 5 && articulos < 10){

precio = precio \* (1-0.10)

} else if (articulos >= 10 && articulos < 15){

precio = precio \* (1-0.15)

} else {

precio = precio \* (1-0.20)

}

**Operador ternario**

El operador ternario consiste en evaluar si una expresión es verdadera o falsa. Parecido a un condicional, pero en una línea de código. Esto sirve para evaluar una condición de manera rápida. La estructura que sigue es la siguiente y se lee como: "La condición es verdadera (?), si es así ejecuta el “Bloque verdadero”, caso contrario (:), ejecuta el “Bloque falso”.

condicion ? Bloque verdadero : Bloque falso

Por ejemplo, guardemos en una variable un mensaje si un número es par o impar.

function esPar(numero){

return numero % 2 === 0 ? "Es par" : "No es par"

}

esPar(2) // "Es par"

esPar(3) // "No es par"

Ejempl0os de Condicionales:

if (true) {

console.log ("Hola a todos");

}

else {

console.log ("No hay nadie");

}

var op1 = "Piedra";

var op2 = "Papel";

var op3 = "Tijera";

var resultado = function(user, cpu){

retu(user != cpu){

if(user === op1 && cpu === op3){

console.log("el usuario GANO con "+ op1)

}else if(user === op2 && cpu === op1){

console.log( "el usuario GANO con " + op2)

}else if(user === op3 && cpu === op2){

console.log("el usuario GANO con " + op3)

}else{

console.log("La CPU Gano!!")

}

}else if(user === cpu){

console.log("Empate")

}

};

**switch**

Colocamos la palabra reservada switch y seguido de la variable o expresión a evaluar, pero sin ningún operador de comparación.

switch (expresion) {}

Después colocamos cada caso con la palabra reservada case y el valor que deberá ser igual a la expresión. Seguido colocamos el bloque de código a ejecutar y al final la palabra reservada break para que no vuelva a evaluar otra condición si ya se cumplió.

switch (expresion) {

case 1: {

// Bloque 1

break

}

case 2: {

// Bloque 2

break

}

}

Finalmente, colocamos la condición por defecto con la palabra reservada default que se ejecutará si ninguno de los casos fue el correcto. Esto es semejante al bloque else.

Por ejemplo, creemos un semáforo.

function semaforo(color) {

switch (color) {

case "verde": {

console.log("¡Sigue!")

break

}

case "amarillo": {

console.log("¡Detente!")

break

}

case "rojo": {

console.log("¡No puedes avanzar!")

break

}

default: {

console.log("¡No reconozco ese color! :(")

}

}

}

semaforo("verde") //'¡Sigue!'

var pregunta = prompt("Ingresa tu opción: piedra, papel o tijera ");

var user = pregunta.toLowerCase();

var options = ["piedra", "papel", "tijera"];

var machine = options[Math.floor(Math.random() \* 3)];

// let numero = 'a';

//con true los casos van a pasar

switch (true) {

case (user === machine):

console.log('es un empate');

break;

case (machine === 'piedra' && user === 'papel'):

console.log('Ganaste')

break;

case (machine === 'papel' && user === 'tijera'):

console.log('Ganaste')

break;

case (machine === 'tijera' && user === 'piedra'):

console.log('Ganaste')

break;

default:

console.log('¡Perdiste!');

}

var tijeras = "tijeras";

var piedra = "piedra";

var papel = "papel";

function resultadoGanador(user, cpu) {

switch (true) {

case user === cpu :

console.log("Empate")

break;

case user === "tijeras" && cpu === "papel" :

console.log("Ganaste")

break;

case user === "piedra" && cpu === "tijeras" :

console.log("Ganaste")

break;

case user === "papel" && cpu === "piedra" :

console.log("Ganaste")

break;

default:

console.log("Perdiste")

}

}

function solution(article) {

switch (true) {

case (article == "computadora"):

return "Con mi computadora puedo programar usando JavaScript";

case (article == "celular"):

return "En mi celular puedo aprender usando la app de Platzi";

case (article == "cable"):

return "¡Hay un cable en mi bota!";

default:

return "Artículo no encontrado";

}

}

solution('computadora')

solution('celular')

solution('cable')

solution('ornitorrinco')



var frutas = ["manzana", "Banano", "Pera", "Uva"];

console.log(frutas);

console.log(frutas[2]);

console.log(frutas.length);

// Como mutar o cambiar el Arrays

// push va agregar elementos al final de mi Arrays, llamado frutas

// pop permite eleiminar un elemento u objeto de mi array

var masFrutas = frutas.push("Fresa");

frutas = ["manzana", "Banano", "Pera", "Uva", "Fresa"];

var ultimo = frutas.pop("Fresa");

// Como mutar o cambiar el Arrays

// push va agregar elementos al final de mi Arrays, llamado frutas

// pop permite eliminar un elemento u objeto de mi array

// unshift se utiliza para introducir un elemento al array, colocandolo al inicio de l array

// shift eliminar el elemento que este en el inicio del array

// indexof muestra la posición de un elemento dentro del array

var masFrutas = frutas.push("Fresa");

frutas = ["manzana", "Banano", "Pera", "Uva", "Fresa"];

var ultimo = frutas.pop("Fresa");

var nuevalon = frutas.unshift("Coco");

var borrafruta = frutas.shift("Coco");

var posicion = frutas.indexOf("Fresa");

posicion;

DESAFIO

var arraySecreto = [1, "gaylord", 3, "Computador"];

b = typeof(arraySecreto[0]);

function solucion(arraySecreto) {

console.log(b===string? True: False)

}

var arraySecreto = [1, "gaylord", 3, "Computador"];

b = typeof(arraySecreto[0]);

function solucion(arraySecreto) {

return b===string? True: False

}

var arraySecreto = [1, "gaylord", 3, "Computador"];

b = typeof(arraySecreto[0]);

function sol(b) {

if (b==="string") {

console.log("True")

}

else {

console.log("False")

}

}

var arraySecreto = [1, "gaylord", 3, "Computador"];

b = typeof(arraySecreto[0]);

function sol(b) {

if (b==="string") {

return True

}

else {

return False

}

}

**LOOPS o Ciclos**

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

**Ejemplo:**

var estudiante = ["Maria", "sergio", "Rosa", "Daniel"];

function saludarEstudainte(estudiante) {

console.log(`Hola, demos la bienvenida al estudiante ${estudiante}`);

}

for(var i = 0; i < estudiante.length; i++){

saludarEstudainte(estudiante[i])

}

var estudiante = ["Maria", "sergio", "Rosa", "Daniel"];

function saludarEstudainte(estudiante) {

console.log(`Hola, demos la bienvenida al estudiante ${estudiante}`);

}

for(var estudiante of estudiante){

saludarEstudainte(estudiante)

}

var estudiante = ["Maria", "sergio", "Rosa", "Daniel"];

for(var i = 0; i < estudiante.length; i++){

console.log(`Hola, demos la bienvenida al estudiante ${estudiante[i]}`);

}

// WHILE

var estudiante = ["Maria", "sergio", "Rosa", "Daniel"];

function saludarEstudiante(estudiante){

    console.log(`Hola, demos la bienvenida al estudiante ${estudiante}`);

}

while(estudiante.length>0){

    var estudiantex = estudiante.shift();

    saludarEstudainte(estudiantex);

}

// DESAFIO \*\*\*\*\*

var estudiantes = ["Yuli", "Garyn", "Jose", "Dairo"];

count = 0;

var nuevo = "Dissu";

function solucion(estudiantes, count){

estudiantes.push(nuevo);

console.log(estudiantes);

}

if (count===0){

solucion(estudiantes);

}

else if(count>0){

for(i=0; i<count; i++){

estudiantes.pop(i);

console.log(i);

//console.log(estudiantes);

solucion(estudiantes);

}

}

**OBJETOS**

Un objeto es una estructura de datos que permite almacenar valores mediante propiedad - valor a través de la sintaxis de llaves ({}) y separados por comas.

En las propiedades del objeto es opcional colocar las comillas. En el caso de que haya espacios, es obligatorio.

var objeto = {

clave1: "valor1",

"clave 2": "valor2",

}

Excepto por los primitivos y las funciones, todo es un objeto en JavaScript.

Qué son los atributos y métodos

En programación orientada a objetos, un objeto es una representación de la realidad, en el cual sus características propias se definen como atributos y sus acciones se definen como métodos.

En otras palabras, los atributos son las variables y los métodos son las funciones propias de cada objeto.

Por ejemplo, definamos el objeto miAuto. Se coloca entre comillas la propiedad año porque el lenguaje no admite caracteres especiales del español. Aunque en ciertas situaciones si admite.

var miAuto = {

marca: "Toyota",

modelo: "Corolla",

"año": 2020,

detalle: function () {

console.log("Es un auto")

}

}

s propiedades marca, modelo y "año" son los atributos del objeto miAuto. La propiedad detalle es un método del objeto miAuto.

**Cómo acceder a los valores de un objeto**

Mediante la notación de corchetes

Mediante la notación de punto

**Qué es la notación de corchetes**

miAuto["marca"] // "Toyota"

miAuto["modelo"] // "Corolla"

miAuto["año"] // 2020

miAuto["detalle"] // ƒ detalle()

Observa que cuando accedes a un método, únicamente muestra la función, esto sucede porque la propiedad guarda dicha función que aún no es ejecutada. Para ejecutarla hay que utilizar los paréntesis.

miAuto["detalle"]() // "Es un auto"

Finalmente, ten cuidado con las comillas, si nos las usas, estás haciendo referencia a una variable. En este caso existirán tres posibilidades:

var propiedad1 = "marca"

miAuto[propiedad1] // "Toyota"

var propiedad2 = "nombre"

miAuto[propiedad2] // undefined

miAuto[modelo] // ReferenceError: modelo is not defined

Qué es la notación de punto

La notación de punto indicamos con un punto la propiedad del objeto. Si existen espacios, la única forma de acceder a esa propiedad es mediante la notación de corchetes.

objeto.propiedad

Por ejemplo, accedamos a las propiedades del objeto miAuto creado anteriormente.

miAuto.marca // "Toyota"

miAuto.modelo // "Corolla"

miAuto.añó // 2020

miAuto.detalle // ƒ detalle()

Igualmente, para ejecutar el método hay que utilizar los paréntesis.

miAuto.detalle() // "Es un auto"

**Los arrays también son objetos**

La notación punto te debe de parecer familiar, ya que así usábamos los diferentes atributos y métodos de los arrays, como length o map.

Esto es debido a que los arrays también son objetos en JavaScript. Por esta razón, también podemos utilizar la notación de corchetes, pero no es recomendable.

var array = [1, 2, 3]

array["length"] // 3

var newArray = array["map"](function (el) {

return el \* 2

})

newArray // [2,4,6]

**Cómo añadir propiedades de un objeto**

Para añadir propiedades de un objeto, utilizamos la notación de corchetes o de punto con la nueva propiedad, asignándole su respectivo valor.

Por ejemplo, añadamos la propiedad color del objeto miAuto.

miAuto["color"] = "rojo"

// o

miAuto.color = "rojo"

console.log(miAuto)

/\* {

marca: 'Toyota',

modelo: 'Corolla',

'año': 2020,

detalle: ƒ detalle(),

color: 'rojo' <---- nueva propiedad

} \*/

Cómo modificar propiedades de un objeto

Para modificar propiedades de un objeto, utilizamos la notación de corchetes o de punto con la propiedad específica, asignándole su nuevo valor.

Por ejemplo, modifiquemos la propiedad marca, de "Toyota" a "Ford", del objeto miAuto.

miAuto["marca"] = "Ford"

// o

miAuto.marca = "Ford"

console.log(miAuto)

/\* {

marca: 'Ford', <--- Cambió de valor

modelo: 'Corolla',

'año': 2020,

detalle: ƒ detalle(),

} \*/

**Cómo eliminar propiedades de un objeto**

Para eliminar propiedades de un objeto, utilizamos la palabra reservada delete seguido de la propiedad del objeto.

Por ejemplo, eliminemos la propiedad marca del objeto miAuto.

delete miAuto["marca"]

// o

delete miAuto.marca

console.log(miAuto)

/\* {

modelo: 'Corolla', <--- No existe la propiedad marca

'año': 2020,

detalle: ƒ detalle(),

} \*/

**El objeto contexto this**

En JavaScript, el objeto contexto this hace referencia a diferentes valores según su contexto de ejecución. Como es un tema complejo de programación orientada a objetos, no profundizaré.

En objetos, el contexto this hace referencia al propio objeto. Esto sirve para acceder a los atributos y métodos propios del objeto.

var miAuto = {

//...

detalle: function () {

console.log(`Auto ${this.modelo} del ${this.año}.`)

},

}

miAuto.detalle() //'Auto Corolla del 2020.'

¡Funcionó! Ahora creemos otro objeto.

var otroAuto = {

// ...

detalle: function () {

console.log(`Auto ${this.modelo} del ${this.año}.`)

},

}

otroAuto.detalle() //'Auto Corolla del 2020.'

**Función constructora**

Existe un problema al momento de construir varios objetos a partir de un código base, los atributos deben cambiar con respecto a la nueva información. Para esto se utiliza una función constructora.

Una función constructora sirve para crear varios objetos a partir de nueva información, esto es recibido argumentos.

**Cómo generar varios objetos a partir de una función constructora**

Para crear una función constructora, debemos definir los parámetros correspondientes, que serán los atributos del objeto, que cambiarán con la nueva información mediante argumentos. Estos argumentos deben hacer referencia a cada uno del nuevo objeto, esto mediante el objeto contexto this.

Ten en cuenta que los parámetros de la función son diferentes a los atributos del objeto 😄.

function Auto(brand, model, year){

this.marca = brand;

this.modelo = model;

this.año = year;

this.detalle = function () {

console.log(`Auto ${this.modelo} del ${this.año}.`)

}

}

Si ejecutamos la función Auto mostrará un error, necesitamos especificar que vamos a construir una instancia mediante la palabra reservada new.

var miAuto = new Auto("Toyota", "Corolla", 2020)

/\* Auto {

marca: 'Toyota',

modelo: 'Corolla',

'año': 2020,

detalle: ƒ ()

}\*/

var otroAuto = new Auto("Tesla", "Model 3", 2021)

var otroAuto2 = new Auto("Suzuki", "K-20", 2019)

var otroAuto3 = new Auto("Ferrari", "Model N", 2018)

**// DESAFIO**

function auto (MARCA, MODELO, ANNIO){

this.marca = MARCA;

this.modelo = MODELO;

this.annio = ANNIO;

}

var autos = [];

for(let i = 0 ; i < 30 ; i++){

var marca = prompt("Ingresa la marca del auto");

var modelo = prompt("Ingresa el modelo del auto");

var annio = prompt("Ingresa el año del auto");

autos.push(new auto (marca, modelo, annio));

}

for(let i = 0 ; i < autos.length ; i++){

console.log(autos[i]);

}



//RETO

//Paso 1: Creamos una lista vacía de autos

var listaAutos = [];

//Paso 2: Creamos la función constructora

function auto(marca, modelo, anio){

this.marca = marca;

this.modelo = modelo;

this.anio = anio;

};

//Paso 3: Creamos una función que agregue un auto nuevo a la lista

function agregarAuto(marca, modelo, anio){

var nuevoAuto = new auto(marca, modelo, anio);

listaAutos.push(nuevoAuto);

};

//Paso 4: Creamos una función para que el usuario agregue un nuevo carro con sus parámetros

function registrarAutoNuevo(){

var marca = prompt("Ingresa la marca: ");

var modelo = prompt("Ingresa el modelo: ");

var anio = prompt("Ingresa el año: ");

agregarAuto(marca, modelo, anio);

console.log("¡Agregaste tu nuevo carro con éxito!");

};

//Paso 5: Mostramos el arreglo actualizado

listaAutos;

//RETO

solution({

color: 'red',

brand: 'Kia',

});

solution({

color: 'gray',

brand: 'Chevrolet',

licensePlate: 'AAA111',

});

solution({ licensePlate: 'RGB255' });

function solution(car) {

if (car.licensePlate !== undefined) { //valida que exista la propiedad licensePlate

car.drivingLicense = true; //añade una nueva propiedad llamada drivingLicense al objeto car

return car;

}

else {

car.drivingLicense = false; //añade una nueva propiedad llamada drivingLicense al objeto car

return car;

}

}

function solution(car) {

if (car.licensePlate) {

car.drivingLicense = true;

} else {

car.drivingLicense = false;

}

return car

}

**Métodos de recorridos de Arrays**

Existen métodos de arrays para recorrerlos, y devolver un valor o un array con nuevos resultados. Entre estos están dos muy importantes: map y filter.

Utilizaremos el siguiente array de objetos para los ejemplos de los métodos:

var articulos = [

{ nombre: "Bici", costo: 3000 },

{ nombre: "TV", costo: 2500 },

{ nombre: "Libro", costo: 320 },

{ nombre: "Celular", costo: 10000 },

{ nombre: "Laptop", costo: 20000 },

{ nombre: "Teclado", costo: 500 },

{ nombre: "Audifonos", costo: 1700 },

]

**Cómo utilizar el método filter**

El método filter consiste en crear un nuevo array a partir de los elementos originales filtrados mediante una función (callback) que indica la condición a cumplir y es inmutable. Si la condición se cumple, retorna el elemento completo.

**El método filter recibe dos argumentos:**

La función que itera y evalúa si cada elemento del array si cumple con la condición especificada (obligatorio).

Un objeto al que puede hacer referencia el contexto this en la función. Si se lo omite, será undefined. Recuerde que this es diferente según el lugar donde sea invocado.

var otherArray = array.filter(function(), thisArg)

var articulosFiltrados = articulos.filter(function (articulo) {

return articulo.costo <= 500

})

console.log(articulosFiltrados)

/\* [

{ nombre: 'Libro', costo: 320 },

{ nombre: 'Teclado', costo: 500 }

] \*/

// Este ejemplo es imaginando que recibimos una lista de 30 carros diferentes y se nos solicita que se almacene esta información, como estamos aprendiendo a usar js por que es un lenguaje cool y no queremos tipiar todo de manera normal y aburrida, usaremos este lenguaje c: así hacemos un pequeño trabajo de separar todas las marcas, modelos y años respectivamente en distintos arrays para así lograr juntarlos en uno solo usando una función constructiva la cual indicara cada auto ingresado con su marca, modelo y año respectivamente.

var marca = ["Audi", "Subaru", "Lexus", "Porsche", "BMW", "Mazda", "Buick", "Toyota", "Kia", "Honda", "Hyundai", "Volvo", "Mini", "Mercedes-Benz", "Volkswagen", "Ford", "Lincoln", "Scion", "Acura", "Chevrolet", "Nissan", "Infiniti", "GMC", "Cadillac", "Dodge", "Land", "Rover", "Mitsubishi", "Jeep", "Fiat"];

var modelo = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30];

var annio = [2020,2019,2018,2020,2020,2020,2018,2018,2020,2020,2020,2018,2018,2020,2020,2019,2020,2020,2019,2019,2020,2020,2019,2019,2019,2020,2019,2019,2018,2020];

var listaAutos = [];

function autoN(marca, modelo, annio){

this.marca = marca;

this.modelo = modelo;

this.annio = annio;

}

for (var i = 0; i<30; i++){

var autoNuevo = new autoN(marca[i],modelo[i],annio[i]);

listaAutos.push(autoNuevo);

}

// Luego se nos solicito que filtremos los carros por medio del año actual, ya que deseaban saber la cantidad de autos nuevos ingresados. Gracias a nuestro trabajo esto se podía saber de la siguiente manera:

var autosFiltrados = listaAutos.filter(function(auto){

return auto.annio === 2020;

});

console.log(autosFiltrados);

// Así logramos obtener una lista con todos los autos del año 2020 c:

// Posterior a eso, se necesitaba obtener una lista urgentemente de las marcas de los 30 autos que acababan de ingresar por razones ajenas que no nos interesan c: Obviando que nosotros ya tenemos esta lista creada xD y queremos usar js para esto, esta tarea es tan sencilla como hacer:

var marcasRecientes = listaAutos.map(function(auto){

return auto.marca;

});

console.log(marcasRecientes);

// Así obtenemos nuestra lista de marcas recientemente ingresadas c:

**Cómo utilizar el método map**

El método map es inmutable y consiste en crear un nuevo array a partir de los elementos originales transformados mediante una función (callback).

**El método map recibe dos argumentos:**

La función que itera y transforma cada elemento del array (obligatorio).

Un objeto al que puede hacer referencia el contexto this en la función. Si se lo omite, será undefined. Recuerde que this es diferente según el lugar donde sea invocado.

var otherArray = array.map(function(), thisArg)

var otherArray = array.map(function(element, index, array))

var nombreArticulos = articulos.map(function (articulo) {

return articulo.nombre

})

console.log(nombreArticulos)

/\*

[ 'Bici', 'TV', 'Libro', 'Celular', 'Laptop', 'Teclado', 'Audifonos' ]

\*/

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

var articulos = [

{ nombre: "Bici", costo: 3000 },

{ nombre: "TV", costo: 2500 },

{ nombre: "Libro", costo: 320 },

{ nombre: "Celular", costo: 10000 },

{ nombre: "Laptop", costo: 20000 },

{ nombre: "Teclado", costo: 500 },

{ nombre: "Audifonos", costo: 1700 },

]

**Cómo utilizar el método find**

El método find consiste en encontrar el primer elemento de un array que cumpla con la condición especificada en la función (callback). Si ningún elemento cumpla con la condición, retornará undefined.

**El método find recibe dos argumentos:**

La función que itera y evalúa cada elemento del array hasta encuentre uno que cumpla con la condición especificada (obligatorio).

Un objeto al que puede hacer referencia el contexto this en la función. Si se lo omite, será undefined.

array.find(function(), thisArg)

**Practiquemos el uso del método find**

Utilicemos el array articulos que definimos para encontrar algún artículo que su nombre sea Laptop.

Entonces utilizamos el método find que retorne la condición que necesitamos. Recuerda que el primer parámetro de la función callback es cada uno de los elementos del array.

var algunArticulo = articulos.find(function (articulo) {

return (articulo.nombre = "Laptop")

})

console.log(algunArticulo)

/\*

{ nombre: 'Laptop', costo: 3000 }

\*/

**Cómo utilizar el método forEach**

El método forEach de los arrays consiste en ejecutar una función (callback) para cada uno de los elementos iterados. Iterar significa repetir una acción varias veces. Este método no retorna ningún valor.

**Este método recibe dos argumentos:**

La función que itera cada elemento del array (obligatorio).

Un objeto al que puede hacer referencia el contexto this en la función. Si se lo omite, será undefined.

array.forEach(function(), thisArg)

La función, que recibe como argumento el método forEach, utiliza tres parámetros opcionales:

El valor actual del elemento iterado. Es decir, si es la primera iteración, será el primer elemento, y así sucesivamente.

El índice del elemento iterado. Es decir, si es la primera iteración, será el índice 0, y así sucesivamente.

El array que está iterando.

array.forEach(function(element, index, array))

**Practiquemos el uso del método forEach**

Utilicemos el array articulos que definimos para mostrar todos los artículos.

Entonces utilizamos el método forEach y que ejecute la función console.log para cada uno de los elementos. Recuerda que el primer parámetro de la función callback es cada uno de los elementos del array.

articulos.forEach(function (articulo) {

console.log(articulo)

})

/\*

{ nombre: 'Bici', costo: 3000 }

{ nombre: 'TV', costo: 2500 }

...

{ nombre: 'Audifonos', costo: 1700 }

\*/

**Cómo utilizar el método some**

El método somees inmutable y consiste retornar un valor lógico verdadero si existe al menos un elemento que cumpla la condición establecida en la función (callback).

El método some recibe dos argumentos:

La función que itera y evalúa cada elemento del array hasta que al menos uno cumpla con la condición especificada (obligatorio).

Un objeto al que puede hacer referencia el contexto this en la función. Si se lo omite, será undefined.

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" \t "_blank)** - **[foreach()](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/forEach" \t "_blank)** - **[some()](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/some" \t "_blank)** - **[filter()](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array/filter" \t "_blank)**

array.some(function(), thisArg)

La función, que recibe como argumento el método some, utiliza tres parámetros:

El valor actual del elemento iterado. Es decir, si es la primera iteración, será el primer elemento, y así sucesivamente.

El índice del elemento iterado. Es decir, si es la primera iteración, será el índice 0, y así sucesivamente.

El array que está iterando.

array.some(function(element, index, array))

**Practiquemos el uso del método some**

Utilicemos el array articulos que definimos para saber si existe al menos un artículo con el costo menor o igual que 700.

Entonces utilizamos el método some que retorne la condición que necesitamos. Recuerda que el primer parámetro de la función callback es cada uno de los elementos del array.

var existeArticulo = articulos.some(function (articulo) {

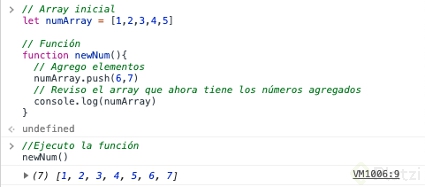
return articulo.costo <= 700

})

console.log(existeArticulo) // true

**Eliminando elementos de un Array**

l 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:

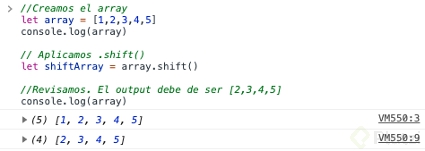


Ahora revisemos un ejemplo con strings:



**.shift()**

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.



El método .pop() eliminará el último elemento de un array.

